talend*
*open integration solutions

Talend Open Studio for Big Data Components

Reference Guide

5.2.1

Adapted for v5.2.1. Supersedes previous Reference Guide releases.

Copyleft

This documentation is provided under the terms of the Creative Commons Public License (CCPL).

For more information about what you can and cannot do with this documentation in accordance with the CCPL, please read: http://creativecommons.org/licenses/by-nc-sa/2.0/

Notices

Apache Hadoop, Hadoop, HDFS, HBase, Hive, Pig are trademarks of the Apache Software Foundation.

All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

Table of Contents

Preface xix
General information xix
Purpose xix
Audience xix
Typographical conventions xix
Feedback and Support xix
Big Data components 1
tBigQueryBulkExec 2
tBigQueryBulkExec Properties 2
Related Scenario 3
tBigQueryInput 4
tBigQueryInput Properties 4
Scenario: Performing a query in BigQuery 4
tBigQueryOutput 8
tBigQueryOutput Properties 8
Scenario: Writing data in BigQuery..... 9
tBigQueryOutputBulk 14
tBigQueryOutputBulk Properties 14
Related Scenario 14
tCassandraClose 15
tCassandraClose properties 15
Related Scenario 15
tCassandraConnection 16
tCassandraConnection properties 16
Related scenario 16
tCassandraInput 17
tCassandraInput properties 17
Scenario: Handling data with Cassandra 18
tCassandraOutput 24
tCassandraOutput properties 24
Related Scenario 25
tCassandraRow 26
tCassandraRow properties 26
Related scenario 26
tHBaseClose 27
tHBaseClose properties 27
Related scenario 27
tHBaseConnection 28
tHBaseConnection properties 28
Related scenario 28
tHBaseInput 29
tHBaseInput properties 29
Scenario: Exchanging customer data with HBase 30
tHBaseOutput 37
tHBaseOutput properties 37
Related scenario 38
tHCatalogInput 39
tHCatalogInput Properties 39
Related scenario 40
tHCatalogLoad 41
tHCatalogLoad Properties 41
Related scenario 42
tHCatalogOperation 43
tHCatalogOperation Properties 43
Scenario: HCatalog table management on Hortonworks Data Platform 46
tHCatalogOutput 56
tHCatalogOutput Properties 56
Related scenario 57
tHDFSCompare 58
tHDFSCompare properties 58
Related scenario 59
tHDFSConnection 60
tHDFSConnection properties 60
Related scenario 60
tHDFSCopy 61
tHDFSCopy properties 61
Related scenario 62
tHDFSDelete 63
tHDFSDelete properties 63
Related scenario 63
tHDFSExist 64
tHDFSExist properties 64
Scenario: Checking the existence of a file in HDFS 65
tHDFSGet 69
tHDFSGet properties 69
Scenario: Computing data with Hadoop distributed file system 70
tHDFSInput 76
tHDFSInput properties 76
Related scenario 77
tHDFSList 78
tHDFSList properties 78
Scenario: Iterating on a HDFS directory 80
tHDFSOutput 85
tHDFSOutput properties 85
Related scenario 86
tHDFSProperties 87
tHDFSProperties properties 87
Related scenario 88
tHDFSPut 89
tHDFSPut properties 89
Related scenario 90
tHDFSRename 91
tHDFSRename Properties 91
Related scenario 92
tHDFSRowCount 93
tHDFSRowCount properties 93
Related scenario 94
tHiveClose 95
tHiveClose properties 95
Related scenario 95
tHiveConnection 96
tHiveConnection properties 96
Related scenario 97
tHiveInput 98
tHiveInput properties 98
Related scenarios 100
tHiveRow 101
tHiveRow properties 101
Related scenarios 103
tMongoDBClose 104
tMongoDBClose properties 104
Related scenario 104
tMongoDBConnection 105
tMongoDBConnection properties 105
Related scenario 105
tMongoDBInput 106
tMongoDBInput Properties 106
Related scenarios 107
tMongoDBOutput 108
tMongoDBOutput Properties 108
Scenario 1: Creating a collection and writing data to it 109
Scenario 2: Upserting records in a collection 114
tMongoDBRow 122
tMongoDBRow Properties 122
Scenario: Using Mongo functions to create a collection and write data to it 123
tPigAggregate 128
tPigAggregate Properties 128
Related scenario 128
tPigCode 129

tPigCode Properties 129
Scenario: Selecting a column of data from an input file and store it into a local file 129
tPigCross 133
tPigCross Properties 133
Related scenario 133
tPigDistinct 134
tPigDistinct Properties 134
Related scenario 134
tPigFilterColumns 135
tPigFilterColumns Properties 135
Related Scenario 135
tPigFilterRow 136
tPigFilterRow Properties 136
Scenario: Filtering rows of data based on a condition and saving the result to a local file 136
tPigJoin 140
tPigJoin Properties 140
Scenario: Joining two files based on an exact match and saving the result to a local file 141
tPigLoad 147
tPigLoad Properties 147
Related Scenario 148
tPigSort 149
tPigSort Properties 149
Scenario: Sorting data in ascending order 149
tPigStoreResult 153
tPigStoreResult Properties 153
Related Scenario 154
tSqoopExport 155
tSqoopExport Properties 155
Related scenario 156
tSqoopImport 157
tSqoopImport Properties 157
Related scenario 158
tSqoopImportAllTables 159
tSqoopImportAllTables Properties 159
Related scenario 160
Business components 161
tAlfrescoOutput 162
tAlfrescoOutput Properties 162
Scenario: Creating documents on an Alfresco server 167
tMarketoInput 172
tMarketoInput Properties 172
Related Scenario 173
tMarketoListOperation 174
tMarketoListOperation Properties 174
Scenario: Adding a lead record to a list in the Marketo DB 175
tMarketoOutput 178
tMarketoOutput Properties 178
Scenario: Data transmission between Marketo DB and an external system.... 179
tMicrosoftCrmInput 184
tMicrosoftCrmInput Properties 184
Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows 185
tMicrosoftCrmOutput 191
tMicrosoftCrmOutput Properties 191
Related Scenario 192
tOpenbravoERPInput 193
tOpenbravoERPInput properties 193
Related Scenario 193
tOpenbravoERPOutput 194
tOpenbravoERPOutput properties 194
Related scenario 194
tSageX3Input 195
tSageX3Input Properties 195
Scenario: Using query key to extract data from a given Sage X3 system..... 196
tSageX3Output 200
tSageX3Output Properties 200
Scenario: Using a Sage X3 Web service to insert data into a given Sage X3 system 201
tSalesforceBulkExec 205
tSalesforceBulkExec Properties 205
Related Scenario: 206
tSalesforceConnection 207
tSalesforceConnection properties 207
Related scenario 207
tSalesforceGetDeleted 208
tSalesforceGetDeleted properties 208
Scenario: Recovering deleted data from the Salesforce server 209
tSalesforceGetServerTimestamp 211
tSalesforceGetServerTimestamp properties 211
Related scenarios 212
tSalesforceGetUpdated 213
tSalesforceGetUpdated properties 213
Related scenarios 214
tSalesforceInput 215
tSalesforceInput Properties 215
Scenario: Using queries to extract data from a Salesforce database 216
tSalesforceOutput 221
tSalesforceOutput Properties 221
Scenario 1: Deleting data from the Account object 222
Scenario 2: Gathering erroneous data while inserting data to a module at Salesforce.com 224
Scenario 3: Inserting AccountIDs from an Excel File to the Contact Module 227
tSalesforceOutputBulk 232
tSalesforceOutputBulk Properties 232
Scenario: Inserting transformed bulk data into your Salesforce.com 232
tSalesforceOutputBulkExec 237
tSalesforceOutputBulkExec Properties 237
Scenario: Inserting bulk data into your Salesforce.com 238
tSAPBWInput 242
tSAPBWInput Properties 242
Scenario: Reading data from SAP BW database 242
tSAPCommit 246
tSAPCommit Properties 246
Related scenario 246
tSAPConnection 247
tSAPConnection properties 247
Related scenarios 247
tSAPInput 248
tSAPInput Properties 248
Scenario 1: Retrieving metadata from the SAP system 249
Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function 256
tSAPOutput 262
tSAPOutput Properties 262
Related scenario 263
tSAPRollback 264
tSAPRollback properties 264

Related scenarios 264
tSugarCRMInput 265
tSugarCRMInput Properties 265
Scenario: Extracting account data from SugarCRM 265
tSugarCRMOutput 268
tSugarCRMOutput Properties 268
Related Scenario 268
tVtigerCRMInput 269
tVtigerCRMInput Properties 269
Related Scenario 269
tVtigerCRMOutput 270
tVtigerCRMOutput Properties 270
Related Scenario 270
Business Intelligence components 271
tDB2SCD 272
tDB2SCD properties 272
Related scenarios 273
tDB2SCDELT 274
tDB2SCDELT Properties 274
Related Scenario 275
tGreenplumSCD 276
tGreenplumSCD Properties 276
Related scenario 277
tInformixSCD 278
tInformixSCD properties 278
Related scenario 279
tIngresSCD 280
tIngresSCD Properties 280
Related scenario 281
tMSSqlSCD 282
tMSSqlSCD Properties 282
Related scenario 283
tMysqlSCD 284
tMysqlSCD Properties 284
Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) 287
tMysqlSCDELT 295
tMysqlSCDELT Properties 295
Related Scenario 296
tOracleSCD 297
tOracleSCD Properties 297
Related scenario 298
tOracleSCDELT 299
tOracleSCDELT Properties 299
Related Scenario 300
tPaloCheckElements 301
tPaloCheckElements Properties 301
Related scenario 302
tPaloConnection 303
tPaloConnection Properties 303
Related scenario 303
tPaloCube 304
tPaloCube Properties 304
Scenario: Creating a cube in an existing database 305
tPaloCubeList 308
tPaloCubeList Properties 308
Discovering the read-only output schema of tPaloCubeList 309
Scenario: Retrieving detailed cube information from a given database 310
tPaloDatabase 312
tPaloDatabase Properties 312
Scenario: Creating a database 313
tPaloDatabaseList 315
tPaloDatabaseList Properties 315
Discovering the read-only output schema of tPaloDatabaseList 316
Scenario: Retrieving detailed database information from a given Palo server 316
tPaloDimension 319
tPaloDimension Properties 319
Scenario: Creating a dimension with elements 322
tPaloDimensionList 327
tPaloDimensionList Properties 327
Discovering the read-only output schema of tPaloDimensionList 328
Scenario: Retrieving detailed dimension information from a given database 329
tPaloInputMulti 331
tPaloInputMulti Properties 331
Scenario: Retrieving dimension elements from a given cube 332
tPaloOutput 336
tPaloOutput Properties 336
Related scenario 337
tPaloOutputMulti 338
tPaloOutputMulti Properties 338
Scenario 1: Writing data into a given cube 339
Scenario 2: Rejecting inflow data when the elements to be written do not exist in a given cube 342
tPaloRule 346
tPaloRule Properties 346
Scenario: Creating a rule in a given cube 347
tPaloRuleList 350
tPaloRuleList Properties 350
Discovering the read-only output schema of tPaloRuleList 351
Scenario: Retrieving detailed rule information from a given cube 351
tParAccelSCD 354
tParAccelSCD Properties 354
Related scenario 355
tPostgresPlusSCD 356
tPostgresPlusSCD Properties 356
Related scenario 357
tPostgresPlusSCDELT 358
tPostgresPlusSCDELT Properties 358
Related Scenario 359
tPostgresqlSCD 360
tPostgresqlSCD Properties 360
Related scenario 361
tPostgresqlSCDELT 362
tPostgresqlSCDELT Properties 362
Related Scenario 363
tSPSSInput 364
tSPSSInput properties 364
Scenario: Displaying the content of an SPSS .sav file 364
tSPSSOutput 367
tSPSSOutput properties 367
Scenario: Writing data in an .sav file 367
tSPSSProperties 370
tSPSSProperties properties 370
Related scenarios 370
tSPSSStructure 371
tSPSSStructure properties 371
Related scenarios 371
tSybaseSCD 372
tSybaseSCD properties 372
Related scenarios 373
tSybaseSCDELT 374
tSybaseSCDELT Properties 374
Related Scenario 375

Cloud components 377
tAmazonMysqlClose 378
tAmazonMysqlClose properties 378
Related scenario 378
tAmazonMysqlCommit 379
tAmazonMysqlCommit Properties 379
Related scenario 379
tAmazonMysqlConnection 380
tAmazonMysqlConnection Properties 380
Scenario: Inserting data in mother/daughter tables 380
tAmazonMysqlInput 384
tAmazonMysqlInput properties 384
Scenario1: Writing columns from a MySQL database to an output file 385
tAmazonMysqlOutput 389
tAmazonMysqlOutput properties 389
Scenario 1: Adding a new column and altering data in a DB table 392
Scenario 2: Updating data in a database table 396
Scenario 3: Retrieve data in error with a Reject link 398
tAmazonMysqlRollback 404
tAmazonMysqlRollback properties 404
Scenario: Rollback from inserting data in mother/daughter tables 404
tAmazonMysqlRow 406
tAmazonMysqlRow properties 406
Scenario 1: Removing and regenerating a MySQL table index 407
Scenario 2: Using PreparedStatement objects to query data 409
Related scenarios 413
tAmazonOracleClose 414
tAmazonOracleClose properties 414
Related scenario 414
tAmazonOracleCommit 415
tAmazonOracleCommit Properties 415
Related scenario 415
tAmazonOracleConnection 416
tAmazonOracleConnection Properties 416
Related scenario 417
tAmazonOracleInput 418
tAmazonOracleInput properties 418
Related scenarios 419
tAmazonOracleOutput 420
tAmazonOracleOutput properties 420
Related scenarios 422
tAmazonOracleRollback 423
tAmazonOracleRollback properties 423
Related scenario 423
tAmazonOracleRow 424
tAmazonOracleRow properties 424
Related scenarios 425
tMarketoInput 426
tMarketoListOperation 427
tMarketoOutput 428
tSalesforceBulkExec 429
tSalesforceConnection 430
tSalesforceGetDeleted 431
tSalesforceGetServerTimestamp 432
tSalesforceGetUpdated 433
tSalesforceInput 434
tSalesforceOutput 435
tSalesforceOutputBulk 436
tSalesforceOutputBulkExec 437
tSugarCRMInput 438
tSugarCRMOutput 439
Custom Code components 441
tGroovy 442
tGroovy properties 442
Related Scenarios 442
tGroovyFile 443
tGroovyFile properties 443
Scenario: Calling a file which contains Groovy code 443
tJava 445
tJava properties 445
Scenario: Printing out a variable content 445
tJavaFlex 449
tJavaFlex properties 449
Scenario 1: Generating data flow 449
Scenario 2: Processing rows of data with tJavaFlex 452
tJavaRow 456
tJavaRow properties 456
Scenario: Transforming data line by line using tJavaRow 456
tLibraryLoad 460
tLibraryLoad properties 460
Scenario: Checking the format of an e-mail addressl 460
tSetGlobalVar 463
tSetGlobalVar properties 463
Scenario: Printing out the content of a global variable 463
Data Quality components 465
tAddCRCRow 466
tAddCRCRow properties 466
Scenario: Adding a surrogate key to a file 466
tChangeFileEncoding 469
tExtractRegexFields 470
tFuzzyMatch 471
tFuzzyMatch properties 471
Scenario 1: Levenshtein distance of 0 in first names 472
Scenario 2: Levenshtein distance of 1 or 2 in first names 473
Scenario 3: Metaphonic distance in first name 474
tIntervalMatch 476
tIntervalMatch properties 476
Scenario: Identifying Ip country 476
tReplaceList 479
tReplaceList Properties 479
Scenario: Replacement from a reference file 479
tSchemaComplianceCheck 483
tSchemaComplianceCheck Properties 483
Scenario: Validating data against schema 484
tUniqRow 488
tUniqRow Properties 488
Scenario 1: Deduplicating entries 489
tUniservBTGeneric 492
tUniservBTGeneric properties 492
Scenario: Execution of a Job in the *Data Quality Service Hub Studio* 493
tUniservRTConvertName 497
tUniservRTConvertName properties ... 497
Scenario: Analysis of a name line and assignment of the salutation 498
tUniservRTMailBulk 501
tUniservRTMailBulk properties 501
Scenario: Creating an index pool 501
tUniservRTMailOutput 505
tUniservRTMailOutput properties 505
Related scenarios 506

tUniservRTMailSearch 507
tUniservRTMailSearch properties 507
Scenario: Adding contacts to the mailRetrieval index pool 507
tUniservRTPost 512
tUniservRTPost properties 512
Scenario 1: Checking and correcting the postal code, city and street 513
Scenario 2: Checking and correcting the postal code, city and street, as well as rejecting the unfeasible 516

Databases - traditional components 519

tAccessBulkExec 520
tAccessBulkExec properties 520
Related scenarios 521
tAccessCommit 522
tAccessCommit Properties 522
Related scenario 522
tAccessConnection 523
tAccessConnection Properties 523
Scenario: Inserting data in parent/child tables 523
tAccessInput 527
tAccessInput properties 527
Related scenarios 528
tAccessOutput 529
tAccessOutput properties 529
Related scenarios 531
tAccessOutputBulk 532
tAccessOutputBulk properties 532
Related scenarios 532
tAccessOutputBulkExec 533
tAccessOutputBulkExec properties 533
Related scenarios 534
tAccessRollback 535
tAccessRollback properties 535
Related scenarios 535
tAccessRow 536
tAccessRow properties 536
Related scenarios 537
tAS400Close 538
tAS400Close properties 538
Related scenario 538
tAS400Commit 539
tAS400Commit Properties 539
Related scenario 539
tAS400Connection 540
tAS400Connection Properties 540
Related scenario 540
tAS400Input 542
tAS400Input properties 542
Related scenarios 543
tAS400LastInsertId 544
tAS400LastInsertId properties 544
Related scenario 544
tAS400Output 545
tAS400Output properties 545
Related scenarios 547
tAS400Rollback 548
tAS400Rollback properties 548
Related scenarios 548
tAS400Row 549
tAS400Row properties 549
Related scenarios 550
tDB2BulkExec 551
tDB2BulkExec properties 551
Related scenarios 552
tDB2Close 553
tDB2Close properties 553
Related scenario 553
tDB2Commit 554
tDB2Commit Properties 554
Related scenario 554
tDB2Connection 555
tDB2Connection properties 555
Related scenarios 555
tDB2Input 557
tDB2Input properties 557
Related scenarios 558
tDB2Output 559
tDB2Output properties 559
Related scenarios 561
tDB2Rollback 562
tDB2Rollback properties 562
Related scenarios 562
tDB2Row 563
tDB2Row properties 563
Related scenarios 564
tDB2SCD 565
tDB2SCDELT 566
tDB2SP 567
tDB2SP properties 567
Related scenarios 568
tInformixBulkExec 569
tInformixBulkExec Properties 569
Related scenario 571
tInformixClose 572
tInformixClose properties 572
Related scenario 572
tInformixCommit 573
tInformixCommit properties 573
Related Scenario 573
tInformixConnection 574
tInformixConnection properties 574
Related scenario 575
tInformixInput 576
tInformixInput properties 576
Related scenarios 576
tInformixOutput 577
tInformixOutput properties 577
Related scenarios 578
tInformixOutputBulk 580
tInformixOutputBulk properties 580
Related scenario 581
tInformixOutputBulkExec 582
tInformixOutputBulkExec properties ... 582
Related scenario 583
tInformixRollback 585
tInformixRollback properties 585
Related Scenario 585
tInformixRow 586
tInformixRow properties 586
Related scenarios 587
tInformixSCD 588
tInformixSP 589
tInformixSP properties 589
Related scenario 590
tMSSqlBulkExec 591
tMSSqlBulkExec properties 591
Related scenarios 592
tMSSqlColumnList 594
tMSSqlColumnList Properties 594
Related scenario 594
tMSSqlClose 595
tMSSqlClose properties 595
Related scenario 595
tMSSqlCommit 596
tMSSqlCommit properties 596
Related scenarios 596
tMSSqlConnection 597
tMSSqlConnection properties 597
Related scenarios 597
tMSSqlInput 599
tMSSqlInput properties 599

Related scenarios 600
tMSSqlLastInsertId 601
tMSSqlLastInsertId properties 601
Related scenario 601
tMSSqlOutput 602
tMSSqlOutput properties 602
Related scenarios 604
tMSSqlOutputBulk 606
tMSSqlOutputBulk properties 606
Related scenarios 606
tMSSqlOutputBulkExec 607
tMSSqlOutputBulkExec properties 607
Related scenarios 608
tMSSqlRollback 609
tMSSqlRollback properties 609
Related scenario 609
tMSSqlRow 610
tMSSqlRow properties 610
Related scenarios 611
tMSSqlSCD 612
tMSSqlSP 613
tMSSqlSP Properties 613
Related scenario 614
tMSSqlTableList 615
tMSSqlTableList Properties 615
Related scenario 615
tMysqlBulkExec 616
tMysqlBulkExec properties 616
Related scenarios 617
tMysqlClose 618
tMysqlClose properties 618
Related scenario 618
tMysqlColumnList 619
tMysqlColumnList Properties 619
Scenario: Iterating on a DB table and listing its column names 619
tMysqlCommit 622
tMysqlCommit Properties 622
Related scenario 622
tMysqlConnection 623
tMysqlConnection Properties 623
Scenario: Inserting data in mother/daughter tables 623
tMysqlInput 627
tMysqlInput properties 627
Scenario 1: Writing columns from a MySQL database to an output file 628
Scenario 2: Using context parameters when reading a table from a MySQL database 631
tMysqlLastInsertId 634
tMysqlLastInsertId properties 634
Scenario: Get the ID for the last inserted record 634
tMysqlOutput 639
tMysqlOutput properties 639
Scenario 1: Adding a new column and altering data in a DB table 642
Scenario 2: Updating data in a database table 646
Scenario 3: Retrieve data in error with a Reject link 649
tMysqlOutputBulk 654
tMysqlOutputBulk properties 654
Scenario: Inserting transformed data in MySQL database 655
tMysqlOutputBulkExec 658
tMysqlOutputBulkExec properties 658
Scenario: Inserting data in MySQL database 659
tMysqlRollback 661
tMysqlRollback properties 661
Scenario: Rollback from inserting data in mother/daughter tables 661
tMysqlRow 662
tMysqlRow properties 662
Scenario 1: Removing and regenerating a MySQL table index 663
Scenario 2: Using PreparedStatement objects to query data 665
Scenario 3: Combining two flows for selective output 669
tMysqlSCD 675
tMysqlSCDELT 676
tMysqlSP 677
tMysqlSP Properties 677
Scenario: Finding a State Label using a stored procedure 678
tMysqlTableList 681
tMysqlTableList Properties 681
Related scenario 681
tOracleBulkExec 682
tOracleBulkExec properties 682
Scenario: Truncating and inserting file data into Oracle DB 684
tOracleClose 687
tOracleClose properties 687
Related scenario 687
tOracleCommit 688
tOracleCommit Properties 688
Related scenario 688
tOracleConnection 689
tOracleConnection Properties 689
Related scenario 690
tOracleInput 691
tOracleInput properties 691
Scenario 1: Using context parameters when reading a table from an Oracle database 692
Related scenarios 695
tOracleOutput 696
tOracleOutput properties 696
Related scenarios 698
tOracleOutputBulk 700
tOracleOutputBulk properties 700
Related scenarios 700
tOracleOutputBulkExec 702
tOracleOutputBulkExec properties 702
Related scenarios 704
tOracleRollback 705
tOracleRollback properties 705
Related scenario 705
tOracleRow 706
tOracleRow properties 706
Related scenarios 707
tOracleSCD 709
tOracleSCDELT 710
tOracleSP 711
tOracleSP Properties 711
Scenario: Checking number format using a stored procedure 713
tOracleTableList 716
tOracleTableList properties 716
Related scenarios 716
tPostgresqlBulkExec 717
tPostgresqlBulkExec properties 717
Related scenarios 718
tPostgresqlCommit 719
tPostgresqlCommit Properties 719
Related scenario 719
tPostgresqlClose 720
tPostgresqlClose properties 720
Related scenario 720
tPostgresqlConnection 721

tPostgresqlConnection Properties 721
Related scenario 721
tPostgresqlInput 722
tPostgresqlInput properties 722
Related scenarios 723
tPostgresqlOutput 724
tPostgresqlOutput properties 724
Related scenarios 726
tPostgresqlOutputBulk 727
tPostgresqlOutputBulk properties 727
Related scenarios 727
tPostgresqlOutputBulkExec 728
tPostgresqlOutputBulkExec properties 728
Related scenarios 729
tPostgresqlRollback 730
tPostgresqlRollback properties 730
Related scenario 730
tPostgresqlRow 731
tPostgresqlRow properties 731
Related scenarios 732
tPostgresqlSCD 733
tPostgresqlSCDELT 734
tSybaseBulkExec 735
tSybaseBulkExec Properties 735
Related scenarios 736
tSybaseClose 738
tSybaseClose properties 738
Related scenario 738
tSybaseCommit 739
tSybaseCommit Properties 739
Related scenario 739
tSybaseConnection 740
tSybaseConnection Properties 740
Related scenarios 740
tSybaseInput 741
tSybaseInput Properties 741
Related scenarios 742
tSybaseIQBulkExec 743
tSybaseIQBulkExec Properties 743
Related scenarios 745
tSybaseIQOutputBulkExec 746
tSybaseIQOutputBulkExec properties 746
Scenario: Bulk-loading data to a Sybase IQ 12 database 748
Related scenarios 751
tSybaseOutput 752
tSybaseOutput Properties 752
Related scenarios 754
tSybaseOutputBulk 755
tSybaseOutputBulk properties 755
Related scenarios 755
tSybaseOutputBulkExec 757
tSybaseOutputBulkExec properties 757
Related scenarios 759
tSybaseRollback 760
tSybaseRollback properties 760
Related scenarios 760
tSybaseRow 761
tSybaseRow Properties 761
Related scenarios 762
tSybaseSCD 763
tSybaseSCDELT 764
tSybaseSP 765
tSybaseSP properties 765
Related scenarios 766
Databases - appliance/datawarehouse components 767
tGreenplumBulkExec 768
tGreenplumBulkExec Properties 768
Related scenarios 769
tGreenplumClose 770
tGreenplumClose properties 770
Related scenario 770
tGreenplumCommit 771
tGreenplumCommit Properties 771
Related scenarios 771
tGreenplumConnection 772
tGreenplumConnection properties 772
Related scenarios 772
tGreenplumGPLoad 774
tGreenplumGPLoad properties 774
Related scenario 777
tGreenplumInput 778
tGreenplumInput properties 778
Related scenarios 778
tGreenplumOutput 779
tGreenplumOutput Properties 779
Related scenarios 780
tGreenplumOutputBulk 782
tGreenplumOutputBulk properties 782
Related scenarios 782
tGreenplumOutputBulkExec 783
tGreenplumOutputBulkExec properties 783
Related scenarios 784
tGreenplumRollback 785
tGreenplumRollback properties 785
Related scenarios 785
tGreenplumRow 786
tGreenplumRow Properties 786
Related scenarios 787
tGreenplumSCD 788
tIngresBulkExec 789
tIngresBulkExec properties 789
Related scenarios 790
tIngresClose 791
tIngresClose properties 791
Related scenario 791
tIngresCommit 792
tIngresCommit Properties 792
Related scenario 792
tIngresConnection 793
tIngresConnection Properties 793
Related scenarios 793
tIngresInput 794
tIngresInput properties 794
Related scenarios 794
tIngresOutput 795
tIngresOutput properties 795
Related scenarios 796
tIngresOutputBulk 797
tIngresOutputBulk properties 797
Related scenarios 797
tIngresOutputBulkExec 798
tIngresOutputBulkExec properties 798
Scenario: Loading data to a table in the Ingres DBMS 799
Related scenarios 802
tIngresRollback 803
tIngresRollback properties 803
Related scenarios 803
tIngresRow 804
tIngresRow properties 804
Related scenarios 805
tIngresSCD 806
tNetezzaBulkExec 807
tNetezzaBulkExec properties 807
Related scenarios 808
tNetezzaClose 809
tNetezzaClose properties 809
Related scenario 809
tNetezzaCommit 810

tNetezzaCommit Properties 810
Related scenario 810
tNetezzaConnection 811
tNetezzaConnection Properties 811
Related scenarios 811
tNetezzaInput 812
tNetezzaInput properties 812
Related scenarios 813
tNetezzaNzLoad 814
tNetezzaNzLoad properties 814
Related scenario 818
tNetezzaOutput 819
tNetezzaOutput properties 819
Related scenarios 821
tNetezzaRollback 822
tNetezzaRollback properties 822
Related scenarios 822
tNetezzaRow 823
tNetezzaRow properties 823
Related scenarios 824
tParAccelBulkExec 825
tParAccelBulkExec Properties 825
Related scenarios 826
tParAccelClose 827
tParAccelClose properties 827
Related scenario 827
tParAccelCommit 828
tParAccelCommit Properties 828
Related scenario 828
tParAccelConnection 829
tParAccelConnection Properties 829
Related scenario 829
tParAccelInput 831
tParAccelInput properties 831
Related scenarios 832
tParAccelOutput 833
tParAccelOutput Properties 833
Related scenarios 835
tParAccelOutputBulk 836
tParAccelOutputBulk properties 836
Related scenarios 836
tParAccelOutputBulkExec 837
tParAccelOutputBulkExec Properties 837
Related scenarios 838
tParAccelRollback 839
tParAccelRollback properties 839
Related scenario 839
tParAccelRow 840
tParAccelRow Properties 840
Related scenarios 841
tParAccelSCD 842
tTeradataClose 843
tTeradataClose properties 843
Related scenario 843
tTeradataCommit 844
tTeradataCommit Properties 844
Related scenario 844
tTeradataConnection 845
tTeradataConnection Properties 845
Related scenario 846
tTeradataFastExport 847
tTeradataFastExport Properties 847
Related scenario 847
tTeradataFastLoad 848
tTeradataFastLoad Properties 848
Related scenario 848
tTeradataFastLoadUtility 849
tTeradataFastLoadUtility Properties 849
Related scenario 850
tTeradataInput 851
tTeradataInput Properties 851
Related scenarios 852
tTeradataMultiLoad 853
tTeradataMultiLoad Properties 853
Related scenario 854
tTeradataOutput 855
tTeradataOutput Properties 855
Related scenarios 857
tTeradataRollback 858
tTeradataRollback Properties 858
Related scenario 858
tTeradataRow 859
tTeradataRow Properties 859
Related scenarios 860
tTeradataTPTUtility 861
tTeradataTPTUtility Properties 861
Related scenario 862
tTeradataTPump 863
tTeradataTPump Properties 863
Scenario: Inserting data into a Teradata database table 864
tVectorWiseCommit 868
tVectorWiseCommit Properties 868
Related scenario 868
tVectorWiseConnection 869
tVectorWiseConnection Properties 869
Related scenario 869
tVectorWiseInput 871
tVectorWiseInput Properties 871
Related scenario 872
tVectorWiseOutput 873
tVectorWiseOutput Properties 873
Related scenario 875
tVectorWiseRollback 876
tVectorWiseRollback Properties 876
Related scenario 876
tVectorWiseRow 877
tVectorWiseRow Properties 877
Related scenario 878
tVerticaBulkExec 879
tVerticaBulkExec Properties 879
Related scenarios 880
tVerticaClose 881
tVerticaClose properties 881
Related scenario 881
tVerticaCommit 882
tVerticaCommit Properties 882
Related scenario 882
tVerticaConnection 883
tVerticaConnection Properties 883
Related scenario 883
tVerticaInput 884
tVerticaInput Properties 884
Related scenarios 885
tVerticaOutput 886
tVerticaOutput Properties 886
Related scenarios 888
tVerticaOutputBulk 889
tVerticaOutputBulk Properties 889
Related scenarios 889
tVerticaOutputBulkExec 890
tVerticaOutputBulkExec Properties 890
Related scenarios 891
tVerticaRollback 892
tVerticaRollback Properties 892
Related scenario 892
tVerticaRow 893
tVerticaRow Properties 893
Related scenario 894
Databases - other components 895
tCreateTable 896
tCreateTable Properties 896

Scenario: Creating new table in a Mysql Database 898
tDBInput 900
tDBInput properties 900
Scenario 1: Displaying selected data from DB table 900
Scenario 2: Using StoreSQLQuery variable 902
tDBOutput 904
tDBOutput properties 904
Scenario: Writing a row to a table in the MySql database via an ODBC connection 905
tDBSQLRow 908
tDBSQLRow properties 908
Scenario: Resetting a DB auto-increment 909
tEXAInput 910
tEXAInput properties 910
Related scenarios 910
tEXAOutput 912
tEXAOutput properties 912
Related scenario 913
tEXARow 914
tEXARow properties 914
Related scenarios 915
tEXistConnection 916
tEXistConnection properties 916
Related scenarios 916
tEXistDelete 917
tEXistDelete properties 917
Related scenario 918
tEXistGet 919
tEXistGet properties 919
Scenario: Retrieve resources from a remote eXist DB server 920
tEXistList 922
tEXistList properties 922
Related scenario 923
tEXistPut 924
tEXistPut properties 924
Related scenario 925
tEXistXQuery 926
tEXistXQuery properties 926
Related scenario 927
tEXistXUpdate 928
tEXistXUpdate properties 928
Related scenario 929
tFirebirdClose 930
tFirebirdClose properties 930
Related scenario 930
tFirebirdCommit 931
tFirebirdCommit Properties 931
Related scenario 931
tFirebirdConnection 932
tFirebirdConnection properties 932
Related scenarios 932
tFirebirdInput 933
tFirebirdInput properties 933
Related scenarios 933
tFirebirdOutput 934
tFirebirdOutput properties 934
Related scenarios 935
tFirebirdRollback 936
tFirebirdRollback properties 936
Related scenario 936
tFirebirdRow 937
tFirebirdRow properties 937
Related scenarios 938
tHBaseClose 939
tHBaseConnection 940
tHBaseInput 941
tHBaseOutput 942
tHiveClose 943
tHiveConnection 944
tHiveInput 945
tHiveRow 946
tHSQLDbInput 947
tHSQLDbInput properties 947
Related scenarios 948
tHSQLDbOutput 949
tHSQLDbOutput properties 949
Related scenarios 951
tHSQLDbRow 952
tHSQLDbRow properties 952
Related scenarios 953
tInterbaseClose 954
tInterbaseClose properties 954
Related scenario 954
tInterbaseCommit 955
tInterbaseCommit Properties 955
Related scenario 955
tInterbaseConnection 956
tInterbaseConnection properties 956
Related scenarios 956
tInterbaseInput 957
tInterbaseInput properties 957
Related scenarios 957
tInterbaseOutput 958
tInterbaseOutput properties 958
Related scenarios 959
tInterbaseRollback 960
tInterbaseRollback properties 960
Related scenarios 960
tInterbaseRow 961
tInterbaseRow properties 961
Related scenarios 962
tJavaDBInput 963
tJavaDBInput properties 963
Related scenarios 963
tJavaDBOutput 964
tJavaDBOutput properties 964
Related scenarios 965
tJavaDBRow 966
tJavaDBRow properties 966
Related scenarios 967
tJDBCColumnList 968
tJDBCColumnList Properties 968
Related scenario 968
tJDBCClose 969
tJDBCClose properties 969
Related scenario 969
tJDBCCommit 970
tJDBCCommit Properties 970
Related scenario 970
tJDBCConnection 971
tJDBCConnection Properties 971
Related scenario 972
tJDBCInput 973
tJDBCInput properties 973
Related scenarios 974
tJDBCOutput 975
tJDBCOutput properties 975
Related scenarios 977
tJDBCRollback 978
tJDBCRollback properties 978
Related scenario 978
tJDBCRow 979
tJDBCRow properties 979
Related scenarios 980
tJDBCSP 981
tJDBCSP Properties 981
Related scenario 982
tJDBCTableList 983

tJDBCTableList Properties 983
Related scenario 983
tLDAPAttributesInput 984
tLDAPAttributesInput Properties 984
Related scenario 985
tLDAPConnection 986
tLDAPConnection Properties 986
Related scenarios 987
tLDAPInput 988
tLDAPInput Properties 988
Scenario: Displaying LDAP directory's filtered content 989
tLDAPOutput 992
tLDAPOutput Properties 992
Scenario: Editing data in a LDAP directory 993
tLDAPRenameEntry 996
tLDAPRenameEntry properties 996
Related scenarios 997
tMaxDBInput 998
tMaxDBInput properties 998
Related scenario 998
tMaxDBOutput 999
tMaxDBOutput properties 999
Related scenario 1000
tMaxDBRow 1001
tMaxDBRow properties 1001
Related scenario 1002
tMongoDBClose 1003
tMongoDBConnection 1004
tMongoDBInput 1005
tMongoDBOutput 1006
tMongoDBRow 1007
tParseRecordSet 1008
tParseRecordSet properties 1008
Related Scenario 1008
tPostgresPlusBulkExec 1009
tPostgresPlusBulkExec properties 1009
Related scenarios 1010
tPostgresPlusClose 1011
tPostgresPlusClose properties 1011
Related scenario 1011
tPostgresPlusCommit 1012
tPostgresPlusCommit Properties 1012
Related scenario 1012
tPostgresPlusConnection 1013
tPostgresPlusConnection Properties ... 1013
Related scenario 1013
tPostgresPlusInput 1014
tPostgresPlusInput properties 1014
Related scenarios 1015
tPostgresPlusOutput 1016
tPostgresPlusOutput properties 1016
Related scenarios 1018
tPostgresPlusOutputBulk 1019
tPostgresPlusOutputBulk properties ... 1019
Related scenarios 1019
tPostgresPlusOutputBulkExec 1020
tPostgresPlusOutputBulkExec properties 1020
Related scenarios 1021
tPostgresPlusRollback 1022
tPostgresPlusRollback properties 1022
Related scenarios 1022
tPostgresPlusRow 1023
tPostgresPlusRow properties 1023
Related scenarios 1024
tPostgresPlusSCD 1025
tPostgresPlusSCDELT 1026
tSasInput 1027
tSasInput properties 1027
Related scenarios 1027
tSasOutput 1029
tSasOutput properties 1029
Related scenarios 1030
tSQLiteClose 1032
tSQLiteClose properties 1032
Related scenario 1032
tSQLiteCommit 1033
tSQLiteCommit Properties 1033
Related scenario 1033
tSQLiteConnection 1034
SQLiteConnection properties 1034
Related scenarios 1034
tSQLiteInput 1035
tSQLiteInput Properties 1035
Scenario: Filtering SQlite data 1036
tSQLiteOutput 1038
tSQLiteOutput Properties 1038
Related Scenario 1039
tSQLiteRollback 1040
tSQLiteRollback properties 1040
Related scenarios 1040
tSQLiteRow 1041
tSQLiteRow Properties 1041
Scenario: Updating SQLite rows 1042
Related scenarios 1044
DotNET components 1045
tDotNETInstantiate 1046
tDotNETInstantiate properties 1046
Related scenario 1046
tDotNETRow 1047
tDotNETRow properties 1047
Scenario: Utilizing .NET in Talend... 1048
ELT components 1053
tCombinedSQLAggregate 1054
tCombinedSQLAggregate properties .. 1054
Scenario: Filtering and aggregating table columns directly on the DBMS.. 1055
tCombinedSQLFilter 1059
tCombinedSQLFilter Properties 1059
Related Scenario 1059
tCombinedSQLInput 1060
tCombinedSQLInput properties 1060
Related scenario 1060
tCombinedSQLOutput 1061
tCombinedSQLOutput properties 1061
Related scenario 1061
tELTGreenplumInput 1062
tELTGreenplumInput properties 1062
Related scenarios 1062
tELTGreenplumMap 1063
tELTGreenplumMap properties 1063
Scenario: Mapping data using a simple implicit join 1064
Related scenario: 1070
tELTGreenplumOutput 1071
tELTGreenplumOutput properties 1071
Related scenarios 1071
tELTHiveInput 1073
tELTHiveInput properties 1073
Related scenario 1073
tELTHiveMap 1074
tELTHiveMap properties 1074
Scenario: Joining table columns and writing them into Hive 1077
tELTHiveOutput 1084
tELTHiveOutput properties 1084
Related scenario 1084
tELTJDBCInput 1085
tELTJDBCInput properties 1085
Related scenarios 1085
tELTJDBCMap 1086
tELTJDBCMap properties 1086

Related scenario: 1087
tELTJDBCOutput 1088
tELTJDBCOutput properties 1088
Related scenarios 1088
tELTMSSqlInput 1090
tELTMSSqlInput properties 1090
Related scenarios 1090
tELTMSSqlMap 1091
tELTMSSqlMap properties 1091
Related scenario: 1092
tELTMSSqlOutput 1093
tELTMSSqlOutput properties 1093
Related scenarios 1093
tELTMysqlInput 1095
tELTMysqlInput properties 1095
Related scenarios 1095
tELTMysqlMap 1096
tELTMysqlMap properties 1096
Scenario 1: Aggregating table columns and filtering 1098
Scenario 2: ELT using an Alias table.. 1102
tELTMysqlOutput 1106
tELTMysqlOutput properties 1106
Related scenarios 1106
tELTNetezzaInput 1108
tELTNetezzaInput properties 1108
Related scenarios 1108
tELTNetezzaMap 1109
tELTNetezzaMap properties 1109
Related scenarios 1110
tELTNetezzaOutput 1111
tELTNetezzaOutput properties 1111
Related scenarios 1111
tELTOracleInput 1113
tELTOracleInput properties 1113
Related scenarios 1113
tELTOracleMap 1114
tELTOracleMap properties 1114
Scenario: Updating Oracle DB entries 1116
tELTOracleOutput 1119
tELTOracleOutput properties 1119
Scenario: Using the Oracle MERGE function to update and add data simultaneously 1120
tELTPostgresqlInput 1125
tELTPostgresqlInput properties 1125
Related scenarios 1125
tELTPostgresqlMap 1126
tELTPostgresqlMap properties 1126
Related scenario: 1127
tELTPostgresqlOutput 1128
tELTPostgresqlOutput properties 1128
Related scenarios 1128
tELTSybaseInput 1130
tELTSybaseInput properties 1130
Related scenarios 1130
tELTSybaseMap 1131
tELTSybaseMap properties 1131
Related scenarios 1132
tELTSybaseOutput 1133
tELTSybaseOutput properties 1133
Related scenarios 1134
tELTTeradataInput 1135
tELTTeradataInput properties 1135
Related scenarios 1135
tELTTeradataMap 1136
tELTTeradataMap properties 1136
Related scenarios 1137
tELTTeradataOutput 1138
tELTTeradataOutput properties 1138
Related scenarios 1138
tSQLTemplateAggregate 1140
tSQLTemplateAggregate properties ... 1140
Scenario: Filtering and aggregating table columns directly on the DBMS.. 1141
tSQLTemplateCommit 1145
tSQLTemplateCommit properties 1145
Related scenario 1146
tSQLTemplateFilterColumns 1147
tSQLTemplateFilterColumns Properties 1147
Related Scenario 1148
tSQLTemplateFilterRows 1149
tSQLTemplateFilterRows Properties .. 1149
Related Scenario 1150
tSQLTemplateMerge 1151
tSQLTemplateMerge properties 1151
Scenario: Merging data directly on the DBMS 1152
tSQLTemplateRollback 1158
tSQLTemplateRollback properties 1158
Related scenarios 1158
ESB components 1159
tESBConsumer 1160
tESBConsumer properties 1160
Scenario: Returning valid email 1161
tRESTClient 1168
tRESTClient properties 1168
Scenario: Sending and retrieving data by interacting with a RESTful service 1170
tRouteFault 1175
tRouteFault properties 1175
Scenario: Getting messages from tRouteFault 1175
tRouteInput 1180
tRouteInput properties 1180
Scenario: Getting messages from a Route 1180
tRouteOutput 1184
tRouteOutput properties 1184
Scenario: Getting messages from tRouteOutput 1184
File components 1189
tAdvancedFileOutputXML 1190
tApacheLogInput 1191
tApacheLogInput properties 1191
Scenario: Reading an Apache access-log file 1191
tCreateTemporaryFile 1193
tCreateTemporaryFile properties 1193
Scenario: Creating a temporary file and writing data in it 1193
tChangeFileEncoding 1197
tChangeFileEncoding Properties 1197
Scenario: Transforming the character encoding of a file 1197
tFileArchive 1199
tFileArchive properties 1199
Scenario: Zip files using a tFileArchive 1200
tFileCompare 1202
tFileCompare properties 1202
Scenario: Comparing unzipped files... 1203
tFileCopy 1205
tFileCopy Properties 1205
Scenario: Restoring files from bin 1206
tFileDelete 1207
tFileDelete Properties 1207
Scenario: Deleting files 1208
tFileExist 1210
tFileExist Properties 1210

Scenario: Checking for the presence of a file and creating it if it does not exist ... 1210
tFileInputARFF ... 1214
tFileInputARFF properties ... 1214
Scenario: Display the content of a ARFF file ... 1214
tFileInputDelimited ... 1218
tFileInputDelimited properties ... 1218
Scenario: Delimited file content display ... 1219
Scenario 2: Reading data from a remote file in streaming mode ... 1221
tFileInputExcel ... 1224
tFileInputExcel properties ... 1224
Related scenarios ... 1226
tFileInputFullRow ... 1227
tFileInputFull Row properties ... 1227
Scenario: Reading full rows in a delimited file ... 1227
tFileInputJSON ... 1230
tFileInputJSON properties ... 1230
Scenario: Extracting data from the fields of a JSON format file ... 1230
tFileInputLDIF ... 1233
tFileInputLDIF Properties ... 1233
Related scenario ... 1234
tFileInputMail ... 1235
tFileInputMail properties ... 1235
Scenario: Extracting key fields from an email ... 1235
tFileInputMSDelimited ... 1238
tFileInputMSDelimited properties ... 1238
Scenario: Reading a multi structure delimited file ... 1239
tFileInputMSPositional ... 1245
tFileInputMSPositional properties ... 1245
Scenario: Reading data from a positional file ... 1246
tFileInputMSXML ... 1250
tFileInputMSXML Properties ... 1250
Scenario: Reading a multi structure XML file ... 1251
tFileInputPositional ... 1254
tFileInputPositional properties ... 1254
Scenario 1: From Positional to XML file ... 1255
Scenario 2: Handling a positional file based on a dynamic schema ... 1259
tFileInputProperties ... 1265
tFileInputProperties properties ... 1265
Scenario: Reading and matching the keys and the values of different .properties files and outputting the results in a glossary ... 1265
tFileInputRegex ... 1269
tFileInputRegex properties ... 1269
Scenario: Regex to Positional file ... 1270
tFileInputXML ... 1272
tFileList ... 1273
tFileList properties ... 1273
Scenario: Iterating on a file directory ... 1274
tFileOutputARFF ... 1277
tFileOutputARFF properties ... 1277
Related scenarios ... 1278
tFileOutputDelimited ... 1279
tFileOutputDelimited properties ... 1279
Scenario 1: Writing data in a delimited file ... 1280
Scenario 2: Utilizing Output Stream to save filtered data to a local file ... 1283
tFileOutputExcel ... 1286
tFileOutputExcel Properties ... 1286
Related scenario ... 1287
tFileOutputJSON ... 1288
tFileOutputJSON properties ... 1288
Scenario: Writing a JSON structured file ... 1288
tFileOutputLDIF ... 1291
tFileOutputLDIF Properties ... 1291
Scenario: Writing DB data into an LDIF-type file ... 1291
tFileOutputMSDelimited ... 1295
tFileOutputMSDelimited properties ... 1295
Related scenarios ... 1295
tFileOutputMSPositional ... 1296
tFileOutputMSPositional properties ... 1296
Related scenario ... 1296
tFileOutputMSXML ... 1297
tFileOutputMSXML Properties ... 1297
Related scenario ... 1302
tFileOutputPositional ... 1303
tFileOutputPositional Properties ... 1303
Related scenario ... 1304
tFileOutputProperties ... 1305
tFileOutputProperties properties ... 1305
Related scenarios ... 1305
tFileOutputXML ... 1306
tFileProperties ... 1307
tFileProperties Properties ... 1307
Scenario: Displaying the properties of a processed file ... 1307
tFileRowCount ... 1309
tFileRowCount properties ... 1309
Related scenario ... 1309
tFileTouch ... 1310
tFileTouch properties ... 1310
Related scenario ... 1310
tFileUnarchive ... 1311
tFileUnarchive Properties ... 1311
Related scenario ... 1312
tGPGDecrypt ... 1313
tGPGDecrypt Properties ... 1313
Scenario: Decrypt a GnuPG-encrypted file and display its content ... 1313
tHDFSCompare ... 1316
tHDFSConnection ... 1317
tHDFSCopy ... 1318
tHDFSDelete ... 1319
tHDFSExist ... 1320
tHDFSGet ... 1321
tHDFSList ... 1322
tHDFSInput ... 1323
tHDFSOutput ... 1324
tHDFSProperties ... 1325
tHDFSPut ... 1326
tHDFSRename ... 1327
tHDFSRowCount ... 1328
tNamedPipeClose ... 1329
tNamedPipeClose properties ... 1329
Related scenario ... 1329
tNamedPipeOpen ... 1330
tNamedPipeOpen properties ... 1330
Related scenario ... 1330
tNamedPipeOutput ... 1331
tNamedPipeOutput properties ... 1331
Scenario: Writing and loading data through a named-pipe ... 1332
tPivotToColumnsDelimited ... 1337
tPivotToColumnsDelimited Properties ... 1337
Scenario: Using a pivot column to aggregate data ... 1337
tSqoopExport ... 1340

tSqoopImport **1341**
tSqoopImportAllTables **1342**
Internet components **1343**
tFileFetch **1344**
tFileFetch properties 1344
Scenario 1: Fetching data through HTTP 1345
Scenario 2: Reusing stored cookie to fetch files through HTTP 1346
Related scenario 1349
tFileInputJSON **1350**
tFTPConnection **1351**
tFTPConnection properties 1351
Related scenarios 1351
tFTPDelete **1352**
tFTPDelete properties 1352
Related scenario 1352
tFTPFileExist **1353**
tFTPFileExist properties 1353
Related scenario 1354
tFTPFileList **1355**
tFTPFileList properties 1355
Scenario: Iterating on a remote directory 1356
tFTPFileProperties **1359**
tFTPFileProperties Properties 1359
Related scenario 1360
tFTPGet **1361**
tFTPGet properties 1361
Related scenario 1362
tFTPPut **1363**
tFTPPut properties 1363
Scenario: Putting files on a remote FTP server 1364
tFTPRename **1367**
tFTPRename Properties 1367
Related scenario 1368
tFTPTruncate **1369**
tFTPTruncate properties 1369
Related scenario 1370
tHttpRequest **1371**
tHttpRequest properties 1371
Scenario: Sending a HTTP request to the server and saving the response information to a local file 1372
tJMSInput **1374**
tJMSInput properties 1374
Related scenarios 1375
tJMSOutput **1376**
tJMSOutput properties 1376
Related scenarios 1377
tMicrosoftMQInput **1378**
tMicrosoftMQInput Properties 1378
Scenario: Writing and fetching queuing messages from Microsoft message queue 1378
tMicrosoftMQOutput **1383**
tMicrosoftMQOutput Properties 1383
Related scenario 1383
tPOP **1384**
tPOP properties 1384
Scenario: Retrieving a selection of email messages from an email server.. 1385
tREST **1387**
tREST properties 1387
Scenario: Creating and retrieving data by invoking REST Web service.. 1388
tRSSInput **1390**
tRSSInput Properties 1390
Scenario: Fetching frequently updated blog entries. 1390
tRSSOutput **1392**
tRSSOutput Properties 1392
Scenario 1: Creating an RSS flow and storing files on an FTP server 1393
Scenario 2: Creating an RSS flow that contains metadata 1397
Scenario 3: Creating an ATOM feed XML file 1399
tSCPClose **1403**
tSCPClose Properties 1403
Related scenario 1403
tSCPConnection **1404**
tSCPConnection properties 1404
Related scenarios 1404
tSCPDelete **1405**
tSCPDelete properties 1405
Related scenario 1405
tSCPFileExists **1406**
tSCPFileExists properties 1406
Related scenario 1406
tSCPFileList **1407**
tSCPFileList properties 1407
Related scenario 1407
tSCPGet **1408**
tSCPGet properties 1408
Scenario: Getting files from a remote SCP server 1408
tSCPPut **1410**
tSCPPut properties 1410
Related scenario 1410
tSCPRename **1411**
tSCPRename properties 1411
Related scenario 1411
tSCPTruncate **1412**
tSCPRename properties 1412
Related scenario 1412
tSendMail **1413**
tSendMail Properties 1413
Scenario: Email on error 1414
tSetKerberosConfiguration **1416**
tSetKerberosConfiguration properties 1416
Related scenarios 1416
tSetKeystore **1417**
tSetKeystore properties 1417
Scenario: Extracting customer information from a private WSDL file 1417
tSetProxy **1422**
tSetProxy properties 1422
Related scenarios 1422
tSocketInput **1423**
tSocketInput properties 1423
Scenario: Passing on data to the listening port 1423
tSocketOutput **1427**
tSocketOutput properties 1427
Related Scenario 1427
tSOAP **1428**
tSOAP properties 1428
Scenario 1: Extracting the weather information using a Web service 1429
Scenario 2: Using a SOAP message from an XML file to get weather information and saving the information to an XML file 1431
tWebService **1435**
tWebService properties 1435
Scenario: Extracting a name list using a Web service 1436
tWebServiceInput **1442**
tWebServiceInput Properties 1442

Scenario 1: Extracting images through a Web service 1443
Scenario 2: Reading the data published on a Web service using the tWebServiceInput advanced features .. 1444
tXMLRPCInput 1449
tXMLRPCInput Properties 1449
Scenario: Guessing the State name from an XMLRPC 1449
Logs & Errors components 1451
tAssert 1452
tAssert Properties 1452
Scenario: Setting up the assertive condition for a Job execution 1452
tAssertCatcher 1458
tAssertCatcher Properties 1458
Related scenarios 1458
tChronometerStart 1460
tChronometerStart Properties 1460
Related scenario 1460
tChronometerStop 1461
tChronometerStop Properties 1461
Scenario: Measuring the processing time of a subjob and part of a subjob.. 1461
tDie 1465
tDie properties 1465
Related scenarios 1465
tFlowMeter 1466
tFlowMeter Properties 1466
Related scenario 1466
tFlowMeterCatcher 1467
tFlowMeterCatcher Properties 1467
Scenario: Catching flow metrics from a Job 1467
tLogCatcher 1471
tLogCatcher properties 1471
Scenario 1: warning & log on entries.. 1471
Scenario 2: Log & kill a Job 1472
tLogRow 1475
tLogRow properties 1475
Scenario: Delimited file content display 1476
tStatCatcher 1477
tStatCatcher Properties 1477
Scenario: Displaying job stats log 1477
tWarn 1480
tWarn Properties 1480
Related scenarios 1480
Misc group components 1481
tAddLocationFromIP 1482
tAddLocationFromIP Properties 1482
Scenario: Identifying a real-world geographic location of an IP 1482
tBufferInput 1485
tBufferInput properties 1485
Scenario: Retrieving bufferized data.. 1485
tBufferOutput 1488
tBufferOutput properties 1488
Scenario 1: Buffering data (Java) 1488
Scenario 2: Buffering output data on the webapp server 1490
Scenario 3: Calling a Job with context variables from a browser 1493
Scenario 4: Calling a Job exported as Webservice in another Job 1495
tContextDump 1497
tContextDump properties 1497
Related Scenario 1497
tContextLoad 1498
tContextLoad properties 1498
Scenario: Dynamic context use in MySQL DB insert 1499
tFixedFlowInput 1501
tFixedFlowInput properties 1501
Related scenarios 1501
tMemorizeRows 1502
tMemorizeRows properties 1502
Scenario: Counting the occurrences of different ages 1503
tMsgBox 1508
tMsgBox properties 1508
Scenario: ‘Hello world!’ type test 1508
tRowGenerator 1510
tRowGenerator properties 1510
Scenario: Generating random java data 1511
Orchestration components 1515
tFileList 1516
tFlowToIterate 1517
tFlowToIterate Properties 1517
Scenario: Transforming data flow to a list 1517
tForeach 1521
tForeach Properties 1521
Scenario: Iterating on a list and retrieving the values 1521
tInfiniteLoop 1523
tInfiniteLoop Properties 1523
Related scenario 1523
tIterateToFlow 1524
tIterateToFlow Properties 1524
Scenario: Transforming a list of files as data flow 1524
tLoop 1527
tLoop Properties 1527
Scenario: Job execution in a loop 1528
tPostjob 1530
tPostjob Properties 1530
Related scenario 1530
tPrejob 1531
tPrejob Properties 1531
Related scenario 1531
tReplicate 1532
tReplicate Properties 1532
Related scenario 1532
tRunJob 1533
tSleep 1534
tSleep Properties 1534
Related scenarios 1534
tUnite 1535
tUnite Properties 1535
Scenario: Iterate on files and merge the content 1535
tWaitForFile 1539
tWaitForFile properties 1539
Scenario: Waiting for a file to be removed 1540
tWaitForSocket 1543
tWaitForSocket properties 1543
Related scenario 1543
tWaitForSqlData 1544
tWaitForSqlData properties 1544
Scenario: Waiting for insertion of rows in a table 1545
Processing components 1549
tAggregateRow 1550
tAggregateRow properties 1550
Scenario 1: Aggregating values and sorting data 1551
tAggregateSortedRow 1554
tAggregateSortedRow properties 1554
Related scenario 1555
tConvertType 1556
tConvertType properties 1556

Scenario: Converting java types 1556
tDenormalize 1561
tDenormalize Properties 1561
Scenario 1: Denormalizing on one column 1561
Scenario 2: Denormalizing on multiple columns 1563
tDenormalizeSortedRow 1565
tDenormalizeSortedRow properties ... 1565
Scenario: Regrouping sorted rows 1565
tExternalSortRow 1569
tExternalSortRow properties 1569
Related scenario 1570
tExtractDelimitedFields 1571
tExtractDelimitedFields properties 1571
Scenario: Extracting fields from a comma-delimited file 1572
tExtractEBCDICFields 1575
tExtractEBCDICFields properties 1575
Related scenario 1575
tExtractPositionalFields 1576
tExtractPositionalFields properties 1576
Related scenario 1577
tExtractRegexFields 1578
tExtractRegexFields properties 1578
Scenario: Extracting name, domain and TLD from e-mail addresses 1578
tExtractXMLField 1581
tFilterColumns 1582
tFilterColumns Properties 1582
Related Scenario 1582
tFilterRow 1583
tFilterRow Properties 1583
Scenario: Filtering and searching a list of names 1583
tJoin 1586
tJoin properties 1586
Scenario 1: Doing an exact match on two columns and outputting the main and rejected data 1586
tMap 1591
tMap properties 1591
Scenario 1: Mapping data using a filter and a simple explicit join 1591
Scenario 2: Mapping data using inner join rejections 1595
Scenario 3: Cascading join mapping.. 1599
Scenario 4: Advanced mapping using filters, explicit joins and rejections.... 1600
Scenario 5: Advanced mapping with filters and different rejections 1604
Scenario 6: Advanced mapping with lookup reload at each row 1607
Scenario 7: Mapping with join output tables 1613
tNormalize 1617
tNormalize Properties 1617
Scenario: Normalizing data 1617
tPigAggregate 1621
tPigCode 1622
tPigCross 1623
tPigDistinct 1624
tPigFilterColumns 1625
tPigFilterRow 1626
tPigJoin 1627
tPigLoad 1628
tPigSort 1629
tPigStoreResult 1630
tReplace 1631
tReplace Properties 1631
Scenario: multiple replacements and column filtering 1631
tSampleRow 1635
tSampleRow properties 1635
Scenario: Filtering rows and groups of rows 1635
tSortRow 1638
tSortRow properties 1638
Scenario 1: Sorting entries 1638
tSplitRow 1641
tSplitRow properties 1641
Scenario 1: Splitting one row into two rows 1641
tWriteJSONField 1645
tWriteJSONField properties 1645
Related Scenarios 1645
tXMLMap 1646
tXMLMap properties 1646
Scenario 1: Mapping and transforming XML data 1646
Scenario 2: Launching a lookup in a second XML flow to join complementary data 1651
Scenario 3: Mapping data using a filter 1656
Scenario 4: Catching the data rejected by lookup and filter 1658
Scenario 5: Mapping data using a group element 1661
Scenario 6: classing the output data with aggregate element 1665
Scenario 7: Restructuring products data using multiple loop elements 1668
System components 1677
tRunJob 1678
tRunJob Properties 1678
Scenario: Executing a child Job 1679
tSetEnv 1684
tSetEnv Properties 1684
Scenario: Modifying a variable during a Job execution 1684
tSSH 1687
tSSH Properties 1687
Scenario: Remote system information display via SSH 1688
tSystem 1690
tSystem Properties 1690
Scenario: Echo ‘Hello World!’ 1691
Talend MDM components 1693
tMDMBulkLoad 1694
tMDMBulkLoad properties 1694
Scenario: Loading records into a business entity 1697
tMDMClose 1702
tMDMClose properties 1702
Related scenario 1702
tMDMConnection 1703
tMDMConnection properties 1703
Related scenario 1703
tMDMDelete 1704
tMDMDelete properties 1704
Scenario: Deleting master data from an MDM Hub 1705
tMDMInput 1711
tMDMInput properties 1711
Scenario: Reading master data in an MDM hub 1712
tMDMOutput 1715
tMDMOutput properties 1715
Scenario: Writing master data in an MDM hub 1717
tMDMReceive 1723
tMDMReceive properties 1723
Related scenario 1724

tMDMRouteRecord 1725
tMDMRouteRecord properties 1725
Scenario: Routing a record to Event Manager 1726
tMDMSP 1734
tMDMSP Properties 1734
Scenario: Executing a stored procedure in the MDM Hub 1735
tMDMTriggerInput 1740
tMDMTriggerInput properties 1740
Scenario: Exchanging the event information about an MDM record.... 1741
tMDMTriggerOutput 1753
tMDMTriggerOutput properties 1753
Related scenario 1754
tMDMViewSearch 1755
tMDMViewSearch properties 1755
Scenario: Retrieving records from an MDM hub via an existing view 1756
Technical components 1761
tHashInput 1762
tHashInput Properties 1762
Scenario 1: Reading data from the cache memory for high-speed data access 1762
Scenario 2: Clearing the memory before loading data to it in case an iterator exists in the same subjob 1766
tHashOutput 1771
tHashOutput Properties 1771
Related scenarios 1771
XML components 1773
tAdvancedFileOutputXML 1774
tAdvancedFileOutputXML properties 1774
Scenario: Creating an XML file using a loop 1780
tDTDValidator 1785
tDTDValidator Properties 1785
Scenario: Validating XML files 1785
tExtractXMLField 1788
tExtractXMLField properties 1788
Scenario 1: Extracting XML data from a field in a database table 1789
Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file 1791
tFileInputXML 1795
tFileInputXML Properties 1795
Scenario 1: Reading and extracting data from an XML structure 1797
Scenario 2: Extracting erroneous XML data via a reject flow 1798
tFileOutputXML 1802
tFileOutputXML properties 1802
Related scenarios 1803
tWriteXMLField 1804
tWriteXMLField properties 1804
Scenario: Extracting the structure of an XML file and inserting it into the fields of a database table 1805
tXMLMap 1809
tXSDValidator 1810
tXSDValidator Properties 1810
Scenario: Validating data flows against an XSD file 1810
tXSLT 1814
tXSLT Properties 1814
Scenario: Transforming XML to html using an XSL stylesheet 1814

Preface

General information

Purpose

This Reference Guide provides use cases and details about how to set parameters for the major components found in the **Palette** of the **Integration** perspective of *Talend Open Studio for Big Data*.

Information presented in this document applies to release **5.2.1**.

Audience

This guide is for users and administrators of *Talend Open Studio for Big Data*.

The layout of GUI screens provided in this document may vary slightly from your actual GUI.

Typographical conventions

This guide uses the following typographical conventions:

- text in **bold:** window and dialog box buttons and fields, keyboard keys, menus, and menu options,
- text in **[bold]:** window, wizard, and dialog box titles,
- text in `courier`: system parameters typed in by the user,
- text in *italics*: file, schema, column, row, and variable names referred to in all use cases, and also names of the fields in the Basic and Advanced setting views referred to in the property table for each component,
- The icon indicates an item that provides additional information about an important point. It is also used to add comments related to a table or a figure,
- The icon indicates a message that gives information about the execution requirements or recommendation type. It is also used to refer to situations or information the end-user needs to be aware of or pay special attention to.

Feedback and Support

Your feedback is valuable. Do not hesitate to give your input, make suggestions or requests regarding this documentation or product and find support from the **Talend** team, on **Talend**'s Forum website at:

http://talendforge.org/forum

Big Data components

This chapter details the main components that you can find in **Big Data** family of the **Palette**.

Using those components, you can connect, in the unified development environment provided by the Studio, to the modules of the Hadoop distribution you are using and perform operations natively on the big data clusters.

The **Big Data** components create connections to various third-party tools used for transferring, storing or analyzing big data, such as Sqoop, MongoDB and BigQuery and help you quickly load, extract, transform and process large and diverse data sets.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

Also, if you have any questions, concerns or general comments please take part in our product forums which can be found at: http://www.talendforge.org/forum/index.php

tBigQueryBulkExec

The **tBigQueryOutputBulk** and **tBigQueryBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used to feed a dataset. These two steps are fused together in the **tBigQueryOutput** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the dataset.

tBigQueryBulkExec Properties

Component family	Big Data / Google BigQuery	
Function	This component transfers a given file from Google Cloud Storage to Google BigQuery, or uploads a given file into Google Cloud Storage and then transfers it to Google BigQuery.	
Purpose	This component transfers given data to Google BigQuery.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Connection	*Client ID* and *Client secret*	Paste the client ID and the client secret created and viewable on the API Access tab view of the project hosting the BigQuery service and the Cloud Storage service you need to use.
	Project ID	Paste the ID of the project hosting the BigQuery service you need to use. The default ID of this project can be found in the URL of the Google API Console, or by hovering your mouse pointer over the name of the project in the BigQuery Browser Tool.
	Authorization code	Paste the authorization code provided by Google for the access you are building. To obtain the authorization code, you need to execute the Job using this component and when this Job pauses execution to print out an URL address, you navigate to this address to copy the authorization code displayed.
	Dataset	Enter the name of the dataset you need to transfer data to.
	Table	Enter the name of the table you need to transfer data to. If this table does not exist, select the **Create the table if it doesn't exist** check box.
	Action on table	Select the action to be performed from the drop-down list when transferring data to the target table. The action may be: • **Truncate**: it empties the contents of the table and repopulates it with the transferred data. • **Append**: it adds rows to the existing data in the table. • **Empty**: it populates the empty table.
Google storage configuration	*Access key* and *Access secret*	Paste the authentication information obtained from Google for making requests to Google Cloud Storage.

		These keys can be consulted on the Interoperable Access tab view under the Google Cloud Storage tab of the project.
	File to upload	When the data to be transferred to BigQuery is not stored on Google Cloud Storage, browse to, or enter the path to it.
	Bucket	Enter the name of the bucket, the Google Cloud Storage container, that holds the data to be transferred to Google BigQuery.
	File	Enter the directory of the data stored on Google Cloud Storage and to be transferred to BigQuery . If the data is not on Google Cloud Storage, this directory is used as the intermediate destination before the data is transferred to BigQuery.
	Header	Set values to ignore the header of the transferred data. For example, enter *0* to ignore no rows for the data without header.
	Bulk file already exists in Google storage	Select this check box to reuse the authentication information for Google Cloud Storage connection, then, complete the **File** and the **Header** fields.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Field Separator*	Enter character, string or regular expression to separate fields for the transfered data.
	Create directory if not exists	Select this check box to create the directory you defined in the **File** field for Google Cloud Storage, if it does not exist.
	Custom the flush buffer size	Customize the amount of memory used to temporarily store output data. In the **Row number** field, enter the number of rows after which the memory is to be freed again.
	Check disk space	Select the this check box to throw an exception during execution if the disk is full.
	Encoding	Select the encoding from the list or select **Custom** and define it manually.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This is a standalone component.	
Limitation	N/A	

Related Scenario

For related topic, see section *Scenario: Writing data in BigQuery*

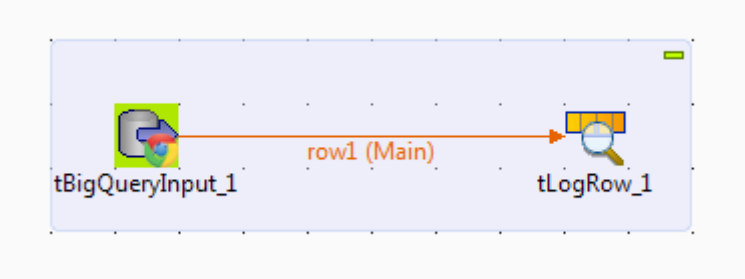
tBigQueryInput

tBigQueryInput Properties

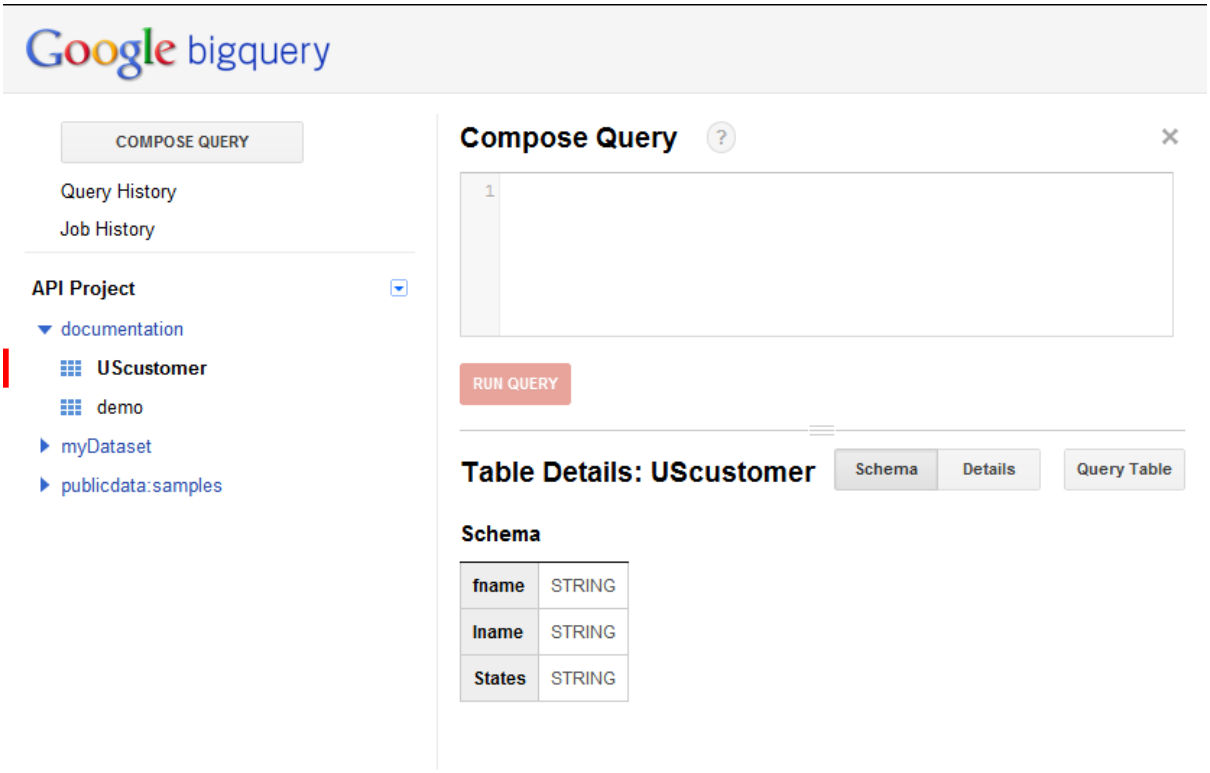
Component family	Big Data / Google BigQuery	
Function	This component connects to Google BigQuery and performs queries in it.	
Purpose	This component performed the queries supported by Google BigQuery.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Connection	*Client ID* and *Client secret*	Paste the client ID and the client secret created and viewable on the API Access tab view of the project hosting the BigQuery service and the Cloud Storage service you need to use.
	Project ID	Paste the ID of the project hosting the BigQuery service you need to use. The default ID of this project can be found in the URL of the Google API Console, or by hovering your mouse pointer over the name of the project in the BigQuery Browser Tool.
	Authorization code	Paste the authorization code provided by Google for the access you are building. To obtain the authorization code, you need to execute the Job using this component and when this Job pauses execution to print out an URL address, you navigate to this address to copy the authorization code displayed.
	Query	Enter the query you need to use.
Advanced settings	*Advanced Separator (for number)*	Select this check box to change the separator used for the numbers.
	Encoding	Select the encoding from the list or select **Custom** and define it manually.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This is an input component. It sends the extracted data to the component that follows it.	
Limitation	N/A	

Scenario: Performing a query in BigQuery

This scenario uses two components to perform the SELECT query in BigQuery and present the result in the Studio.



The following figure shows the schema of the table, *UScustomer*, we use as example to perform the SELECT query in.



We will select the State records and count the occurrence of each State among those records.

Linking the components

1. In the **Integration** perspective of the Studio, create an empty Job, named *BigQueryInput* for example, from the **Job Designs** node in the **Repository** tree view.

   For further information about how to create a Job, see the user guide of the Studio for integration.

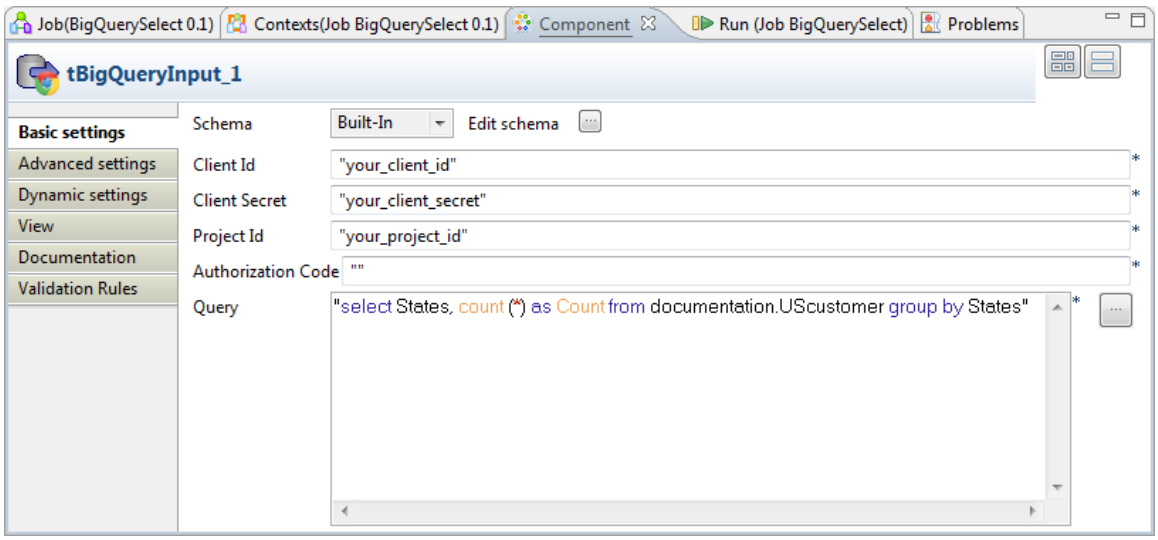
2. Drop **tBigQueryInput** and **tLogRow** onto the workspace.

3. Connect them using the **Row** > **Main** link.

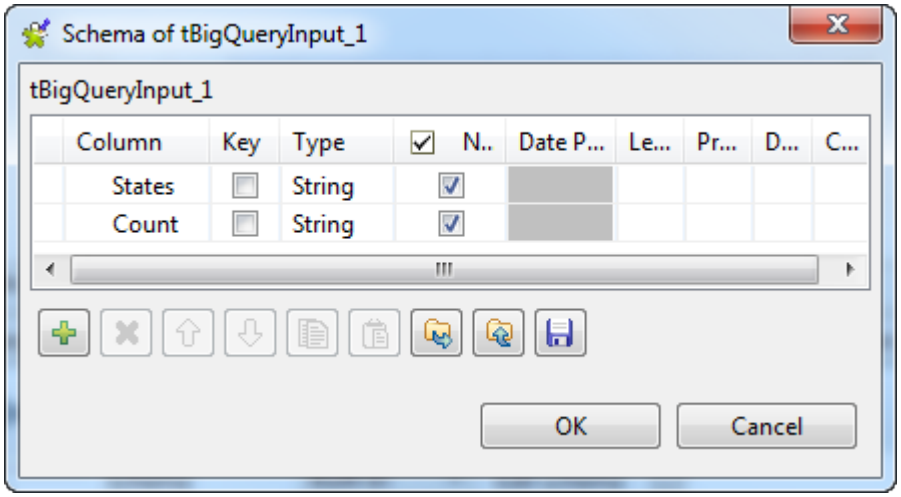
Creating the query

Building access to BigQuery

1. Double-click **tBigQueryInput** to open its **Component** view.



2. Click **Edit schema** to open the editor



3. Click the button twice to add two rows and enter the names of your choice for each of them in the **Column** column. In this scenario, they are: *States* and *Count*.

4. Click **OK** to validate these changes and accept the propagation prompted by the pop-up dialog box.

5. Navigate to the Google APIs Console in your web browser to access the Google project hosting the BigQuery and the Cloud Storage services you need to use.

6. Click the API Access tab to open its view.

7. In the **Component** view of the Studio, paste Client ID, Client secret and Project ID from the API Access tab view to the corresponding fields, respectively.

Getting Authorization code

1. In the **Run** view of the Studio, click **Run** to execute this Job. The execution will pause at a given moment to print out in the console the URL address used to get the authorization code.

2. Navigate to this address in your web browser and copy the authorization code displayed.

3. In the **Component** view of **tBigQueryInput**, paste the authorization code in the **Authorization Code** field.

Writing the query

- In the **Query** field, enter

```
select States, count (*) as Count from documentation.UScustomer group by States
```

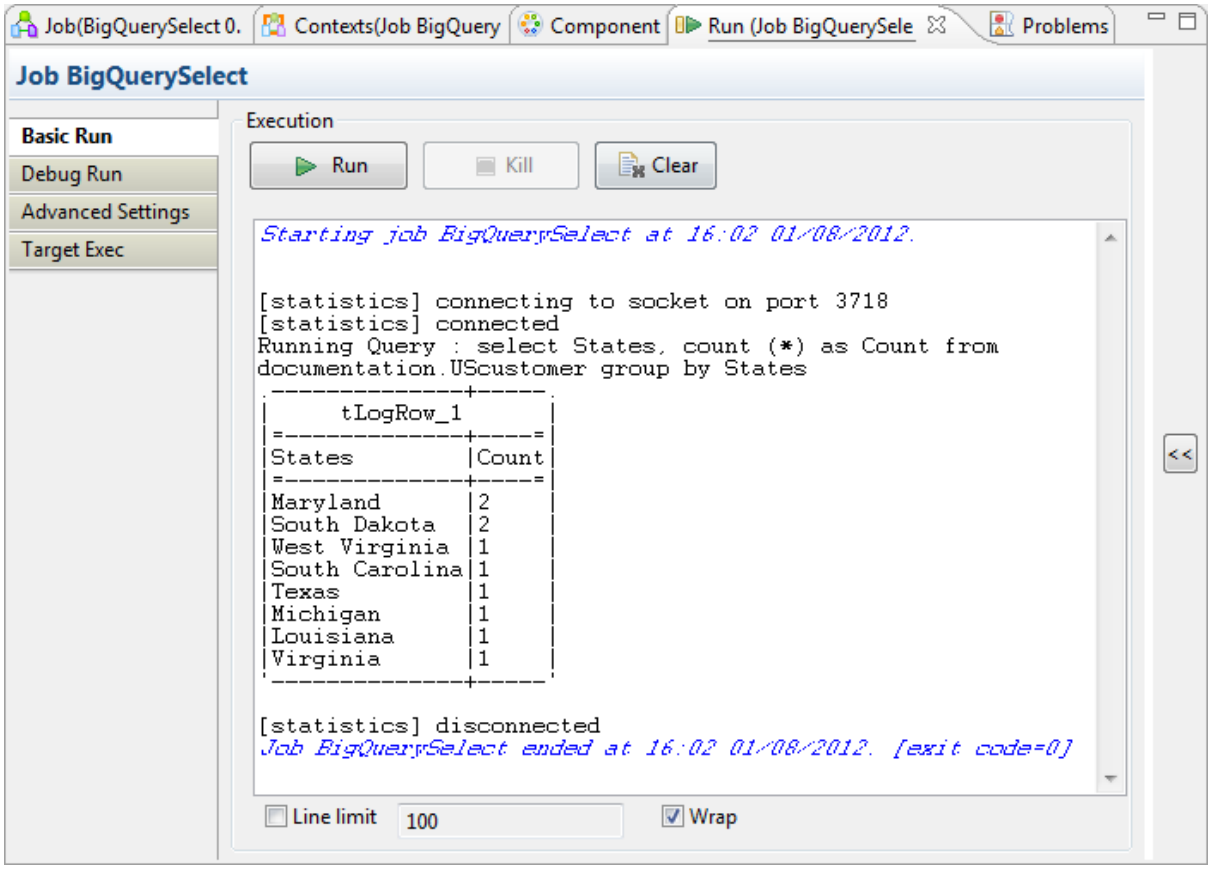
Executing the Job

The **tLogRow** component presents the execution result of the Job. You can configure the presentation mode on its **Component** view.

To do this, double-click **tLogRow** to open the **Component** view and in the **Mode** area, select the **Table (print values in cells of a table)** option.

- To execute this Job, press **F6**.

Once done, the **Run** view is opened automatically, where you can check the execution result.



tBigQueryOutput

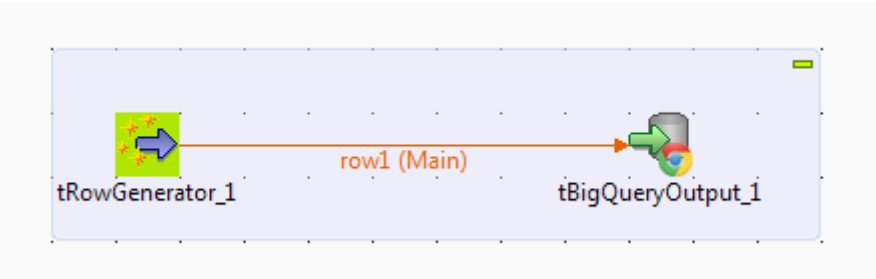
tBigQueryOutput Properties

Component family	Big Data / Google BigQuery	
Function	This component writes the data it receives in a user-specified directory and transfers the data to Google BigQuery via Google Cloud Storage.	
Purpose	This component transfers the data provided by its preceding component to Google BigQuery.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Property type	**Built-in**: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Local filename	Browse to, or enter the path to the file you want to write the received data in.
	Append	Select this check box to add rows to the existing data in the file specified in **Local filename**.
Connection	*Client ID* and *Client secret*	Paste the client ID and the client secret created and viewable on the API Access tab view of the project hosting the BigQuery service and the Cloud Storage service you need to use.
	Project ID	Paste the ID of the project you need to use. The default ID of this project can be found in the URL of the Google API Console, or by hovering your mouse pointer over the name of the project in the BigQuery Browser Tool.
	Authorization code	Paste the authorization code provided by Google for the access you are building. To obtain the authorization code, you need to execute the Job using this component and when this Job pauses execution to print out an URL address, you navigate to this address to copy the authorization code displayed.
	Dataset	Enter the name of the dataset you need to transfer data to.
	Table	Enter the name of the table you need to transfer data to. If this table does not exist, select the **Create the table if it doesn't exist** check box.
	Action on table	Select the action to be performed from the drop-down list when transferring data to the target table. The action may be: • **Truncate**: it empties the contents of the table and repopulates it with the transferred data. • **Append**: it adds rows to the existing data in the table. • **Empty**: it populates the empty table.
Google storage configuration	*Access key* and *Access secret*	Paste the authentication information obtained from Google for making requests to Google Cloud Storage. These keys can be consulted on the Interoperable Access tab view under the Google Cloud Storage tab of the project.

	Bucket	Enter the name of the bucket, the Google Cloud Storage container, that holds the data to be transferred to Google BigQuery.
	File	Enter the directory of the data stored on Google Cloud Storage and to be transferred to BigQuery . If the data is not on Google Cloud Storage, this directory is used as the intermediate destination before the data is transferred to BigQuery.
	Header	Set values to ignore the header of the transferred data. For example, enter *0* to ignore no rows for the data without header and set *1* for the data with header at the first row.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Field Separator*	Enter character, string or regular expression to separate fields for the transfered data.
	Create directory if not exists	Select this check box to create the directory you defined in the **File** field for Google Cloud Storage, if it does not exist.
	Custom the flush buffer size	Customize the amount of memory used to temporarily store output data. In the **Row number** field, enter the number of rows after which the memory is to be freed again.
	Check disk space	Select the this check box to throw an exception during execution if the disk is full.
	Encoding	Select the encoding from the list or select **Custom** and define it manually.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This is an output component used at the end of a Job. It receives data from its preceding component such as **tFileInputDelimited**, **tMap** or **tMysqlInput**.	
Limitation	N/A	

Scenario: Writing data in BigQuery

This scenario uses two components to write data in Google BigQuery.



Linking the components

1. In the Integration perspective of the Studio, create an empty Job, named WriteBigQuery for example, from the **Job Designs** node in the **Repository** tree view.

   For further information about how to create a Job, see the user guide of the Studio for integration.

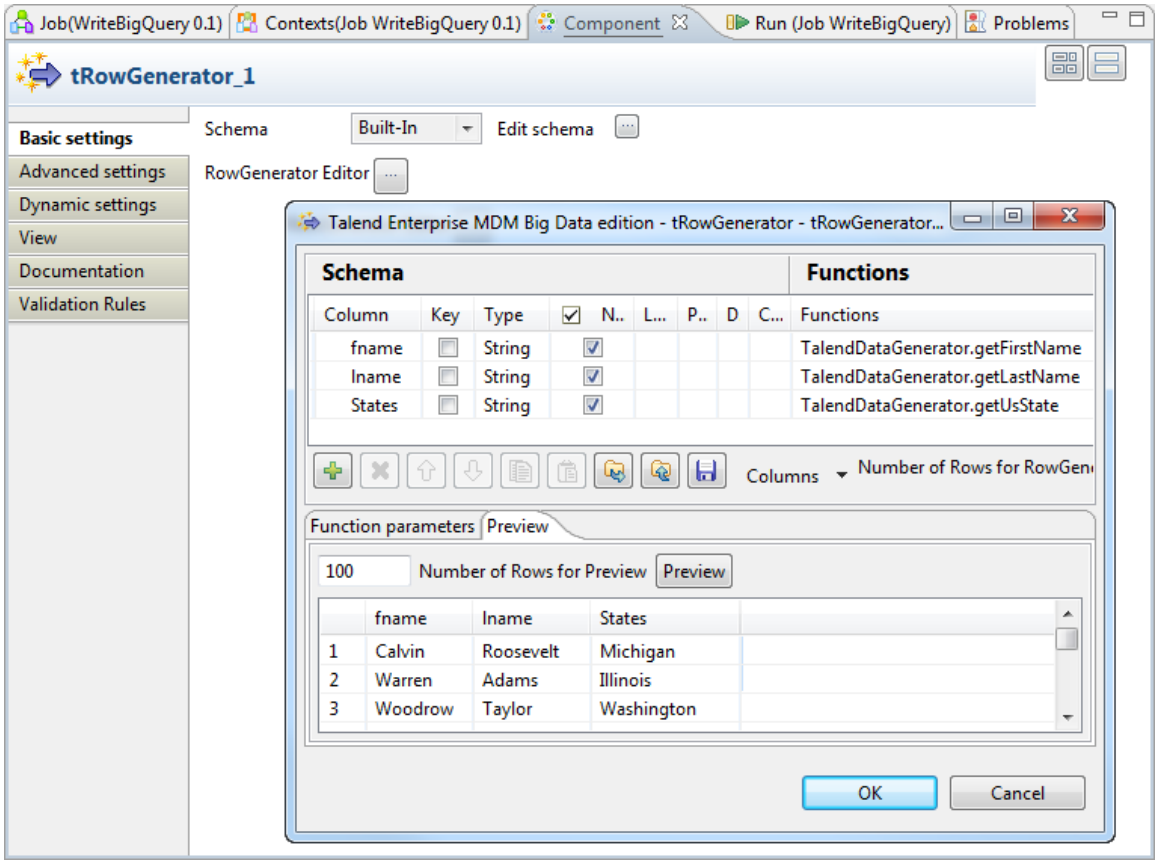
2. Drop **tRowGenerator** and **tBigQueryOutput** onto the workspace.

   The **tRowGenerator** component generates the data to be transferred to Google BigQuery in this scenario. In the real-world case, you can use other components such as **tMysqlInput** or **tMap** in the place of **tRowGenerator** to design a sophisticated process to prepare your data to be transferred.

3. Connect them using the **Row > Main** link.

Preparing the data to be transferred

1. Double-click **tRowGenerator** to open its **Component** view.



2. Click **RowGenerator Editor** to open the editor.

3. Click three times to add three rows in the **Schema** table.

4. In the **Column** column, enter the name of your choice for each of the new rows. For example, *fname*, *lname* and *States*.

5. In the **Functions** column, select *TalendDataGenerator.getFirstName* for the *fname* row, *TalendDataGenerator.getLastName* for the *lname* row and *TalendDataGenerator.getUsState* for the *States* row.

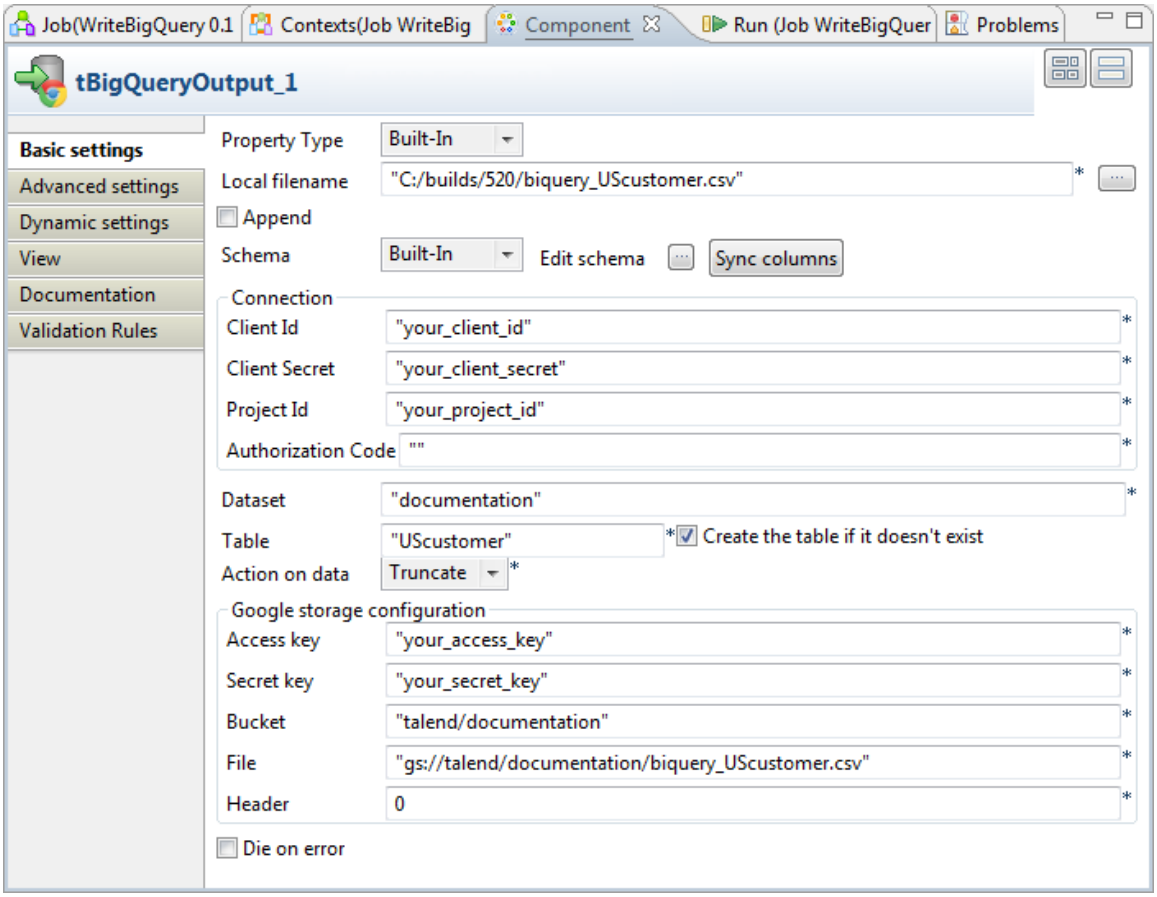
6. In the **Number of Rows for RowGenerator** field, enter, for example, *100* to define the number of rows to be generated.

7. Click **OK** to validate these changes.

Configuring the access to BigQuery and Cloud Storage

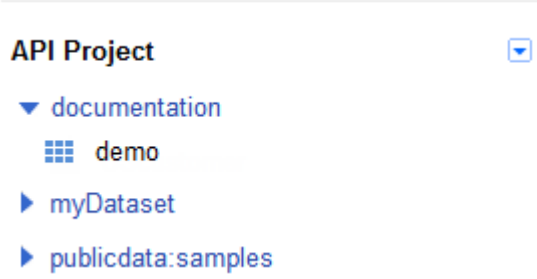
Building access to BigQuery

1. Double-click **tBigQueryOutput** to open its **Component** view.



2. Click **Sync columns** to retrieve the schema from its preceding component.
3. In the **Local filename** field, enter the directory where you need to create the file to be transferred to BigQuery.
4. Navigate to the Google APIs Console in your web browser to access the Google project hosting the BigQuery and the Cloud Storage services you need to use.
5. Click the API Access tab to open its view
6. In the **Component** view of the Studio, paste Client ID, Client secret and Project ID from the API Access tab view to the corresponding fields, respectively.
7. In the **Dataset** field, enter the dataset you need to transfer data in. In this scenario, it is *documentation*.

   This dataset must exist in BigQuery. The following figure shows the dataset used by this scenario.



8. In the **Table** field, enter the name of the table you need to write data in, for example, *UScustomer*.

   If this table does not exist in BigQuery you are using, select **Create the table if it doesn't exist**.
9. In the **Action on data** field, select the action. In this example, select **Truncate** to empty the contents, if there are any, of target table and to repopulate it with the transferred data.

Building access to Cloud Storage

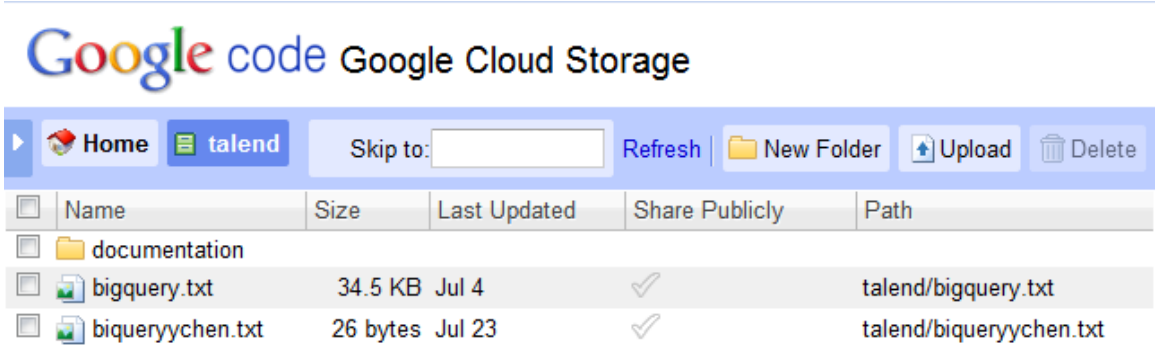
1. Navigate to the Google APIs Console in your web browser to access the Google project hosting the BigQuery and the Cloud Storage services you need to use.

2. Click Google Cloud Storage > Interoperable Access to open its view

3. In the **Component** view of the Studio, paste Access key, Access secret from the Interoperable Access tab view to the corresponding fields, respectively.

4. In the **Bucket** field, enter the path to the bucket you want to store the transferred data in. In this example, it is *talend/documentation*

   This bucket must exist in the directory in Cloud Storage



5. In the **File** field, enter the directory where in Google Clould Storage you receive and create the file to be transferred to BigQuery. In this example, it is *gs://talend/documentation/biquery_UScustomer.csv*. The file name must be the same as the one you defined in the **Local filename** field.

   Troubleshooting: if you encounter issues such as *Unable to read source URI* of the file stored in Google Cloud Storage, check whether you put the same file name in these two fields.

6. Enter *0* in the **Header** field to ignore no rows in the transferred data.

Getting Authorization code

1. In the **Run** view of the Studio, click **Run** to execute this Job. The execution will pause at a given moment to print out in the console the URL address used to get the authorization code.

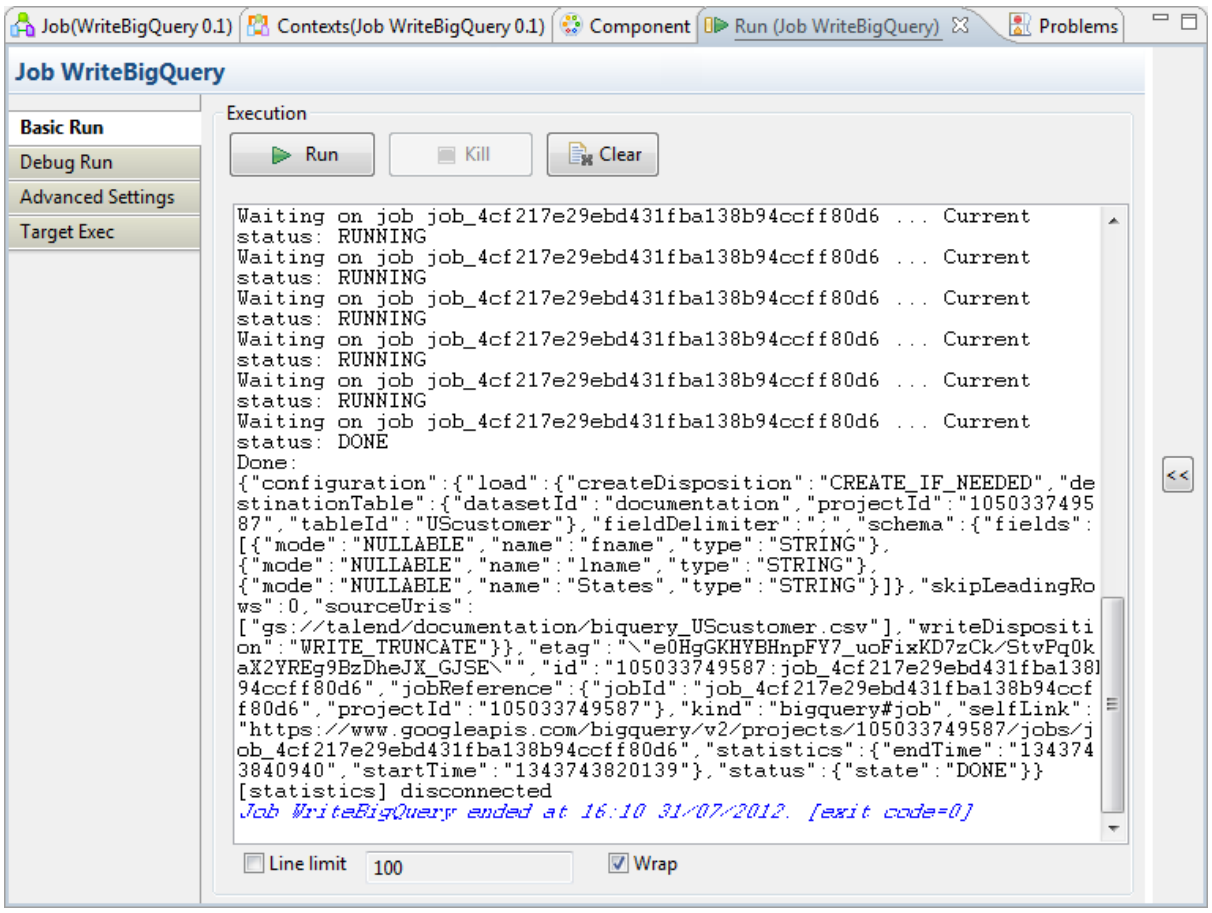
2. Navigate to this address in your web browser and copy the authorization code displayed.

3. In the **Component** view of **tBigQueryOutput**, paste the authorization code in the **Authorization Code** field.

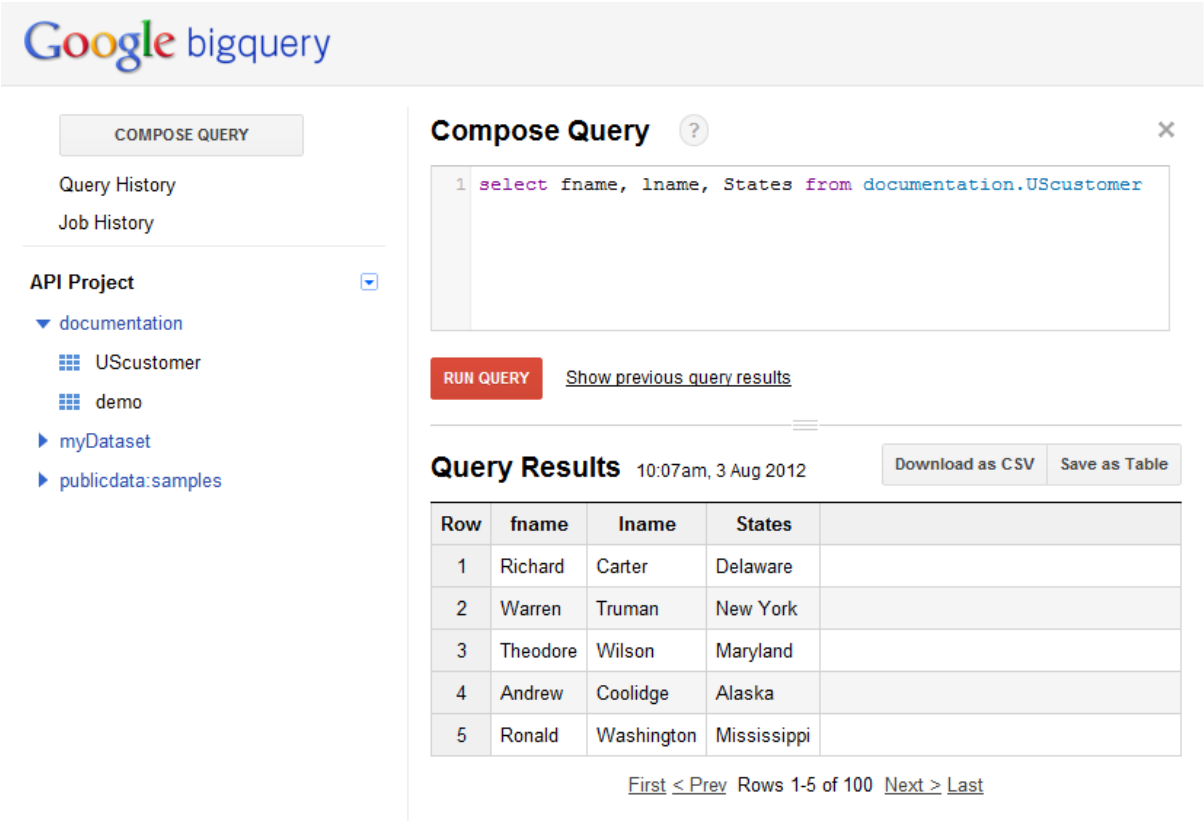
Executing the Job

- Press **F6**.

Once done, the **Run** view is opened automatically, where you can check the execution result.



The data is transferred to Google BigQuery.



tBigQueryOutputBulk

The **tBigQueryOutputBulk** and **tBigQueryBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used to feed a dataset. These two steps are fused together in the **tBigQueryOutput** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the dataset.

tBigQueryOutputBulk Properties

Component family	Big Data / Google BigQuery	
Function	This component writes given data into a *.txt* or *.csv* file, ready to be transferred to Google BigQuery.	
Purpose	This component creates a *.txt* or *.csv* file for the data of large size so that you can process it according to your needs before transferring it to Google BigQuery.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Property type	**Built-in**: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File name	Browse, or enter the path to the *.txt* or *.csv* file you need to generate.
	Append	Select the check box to write new data at the end of the existing data. Otherwise, the existing data will be overwritten.
Advanced settings	*Field Separator*	Enter character, string or regular expression to separate fields for the transfered data.
	Create directory if not exists	Select this check box to create the directory you defined in the File name field, if it does not exist.
	Custom the flush buffer size	Customize the amount of memory used to temporarily store output data. In the **Row number** field, enter the number of rows after which the memory is to be freed again.
	Check disk space	Select the this check box to throw an exception during execution if the disk is full.
	Encoding	Select the encoding from the list or select Custom and define it manually.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level/
Usage	This is an output component which needs the data provided by its preceding component.	
Limitation	N/A	

Related Scenario

For related topic, see section *Scenario: Writing data in BigQuery*

tCassandraClose

tCassandraClose properties

Component family	Big Data / Cassandra	
Function	**tCassandraClose** closes an active connection to the Cassandra server.	
Purpose	**tCassandraClose** is used to close an active connection to the Cassandra server.	
Basic settings	*Component List*	Select an active Cassandra connection to be closed.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.

Related Scenario

For a scenario in which **tCassandraClose** is used, see section *Scenario: Handling data with Cassandra*.

tCassandraConnection

tCassandraConnection properties

Component Family	Big Data / Cassandra	
Function	**tCassandraConnection** opens a connection to a Cassandra server.	
Purpose	**tCassandraConnection** allows you to define a connection to a Cassandra server to be reused by other Cassandra components.	
Basic settings	*Property type*	**Built-in**: No property data stored centrally. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Server	Type in the IP address or hostname of the Cassandra server.
	Port	Type in the listening port number of the Cassandra server.
Required Authentication	*Username*	Fill in this field with the username for the Cassandra authentication.
	Password	Fill in this field with the password for the Cassandra authentication.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is generally used with other Cassandra components, particularly **tCassandraClose**.	
Limitation	n/a	

Related scenario

For a scenario in which **tCassandraConnection** is used, see section *Scenario: Handling data with Cassandra*.

tCassandraInput

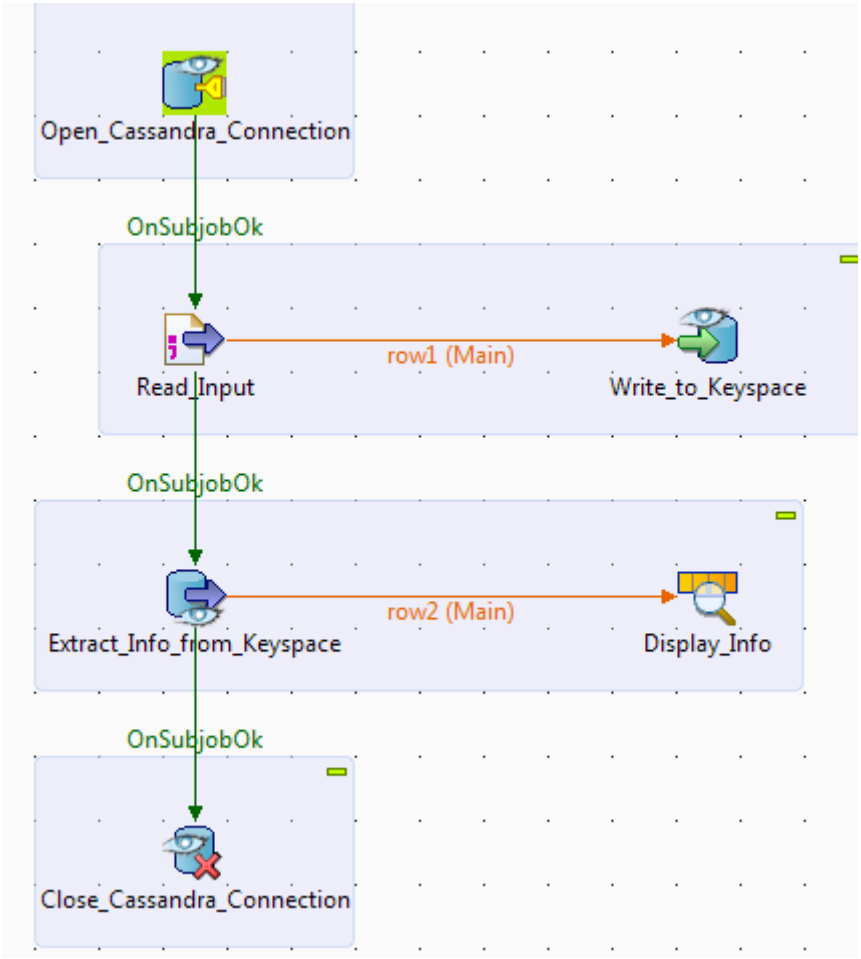
tCassandraInput properties

Component family	Big Data / Cassandra	
Function	**tCassandraInput** allows you to read data from a Cassandra keyspace and send data in the Talend flow.	
Purpose	**tCassandraInput** reads data from the specified Cassandra keyspace and sends data to the next component.	
Basic settings	*Use existing connection*	Select this check box and select the appropriate **tCassandraConnection** component from the **Component List** if you want to re-use the connection parameters that you have already defined.
	Host	Hostname or IP address of the Cassandra server.
	Port	Listening port number of the Cassandra server.
	Required authentication	Select this check box to provide credentials for the Cassandra authentication. This check box appears only if you do not select the **Use existing connection** check box.
	Username	Fill in this field with the username for the Cassandra authentication.
	Password	Fill in this field with the password for the Cassandra authentication.
Keyspace configuration	*Keyspace*	Type in the name of the keyspace from which you want to read data.
Column family configuration	*Column family*	Type in the name of the column family from which you want to read data.
	Column family type	**Standard**: Column family is of standard type. **Super**: Column family is of super type.
	Include row key in output columns	Select this check box to include the row key of the column family in output columns.
	Row key type	Select the appropriate data type of the row key from the drop-down list. This drop-down list appears only if you do not select the **Include row key in output columns** check box.
	Row key column	Select the row key column from the list.
	Include super columns in output columns	Select this check box to include the super columns of the column family in output columns. This check box appears only if you select **Super** from the **Column family type** drop-down list.
	Super column type	Select the appropriate data type of the super column from the drop-down list. This drop-down list appears only if you do not select the **Include super columns in output columns** check box.
	Super columns	Select the super column from the list.
Query configuration	*Specify row keys*	Select this check box to specify the row keys of the column family directly.
	Row Keys	Type in the specific row keys of the column family in the correct format depending on the row key type.

		This field appears only if you select the **Specify row keys** check box.
	Key start	Type in the start row key of the correct data type.
	Key end	Type in the end row key of the correct data type.
	Key limit	Type in the number of rows to be read between the start row key and the end row key.
	Specify columns	Select this check box to specify the column names of the column family directly.
	Columns	Type in the specific column names of the column family in the correct format depending on the column type. This field appears only if you select the **Specify columns** check box.
	Columns range start	Type in the start column name of the correct data type.
	Columns range end	Type in the end column name of the correct data type.
	Columns range limit	Type in the number of columns to be read between the start column and the end column.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository** If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.

Scenario: Handling data with Cassandra

This scenario describes a simple Job that reads the employee data from a CSV file, writes the data to a Cassandra keyspace, then extracts the personal information of some employees and displays the information on the console.



This scenario requires six components, which are:

- **tCassandraConnection**: opens a connection to the Cassandra server.
- **tFileInputDelimited**: reads the input file, defines the data structure and sends it to the next component.
- **tCassandraOutput**: writes the data it receives from the preceding component into a Cassandra keyspace.
- **tCassandraInput**: reads the data from the Cassandra keyspace.
- **tLogRow**: displays the data it receives from the preceding component on the console.
- **tCassandraClose**: closes the connection to the Cassandra server.

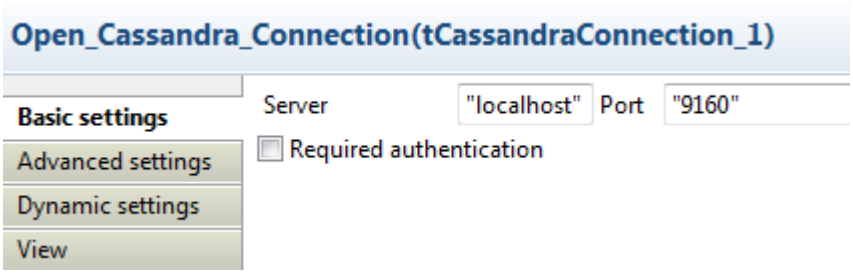
Dropping and linking the components

1. Drop the following components from the **Palette** onto the design workspace: **tCassandraConnection**, **tFileInputDelimited**, **tCassandraOutput**, **tCassandraInput**, **tLogRow** and **tCassandraClose**.
2. Connect **tFileInputDelimited** to **tCassandraOutput** using a **Row** > **Main** link.
3. Do the same to connect **tCassandraInput** to **tLogRow**.
4. Connect **tCassandraConnection** to **tFileInputDelimited** using a **Trigger** > **OnSubjobOk** link.
5. Do the same to connect **tFileInputDelimited** to **tCassandraInput** and **tCassandraInput** to **tCassandraClose**.
6. Label the components to better identify their functions.

Configuring the components

Opening a Cassandra connection

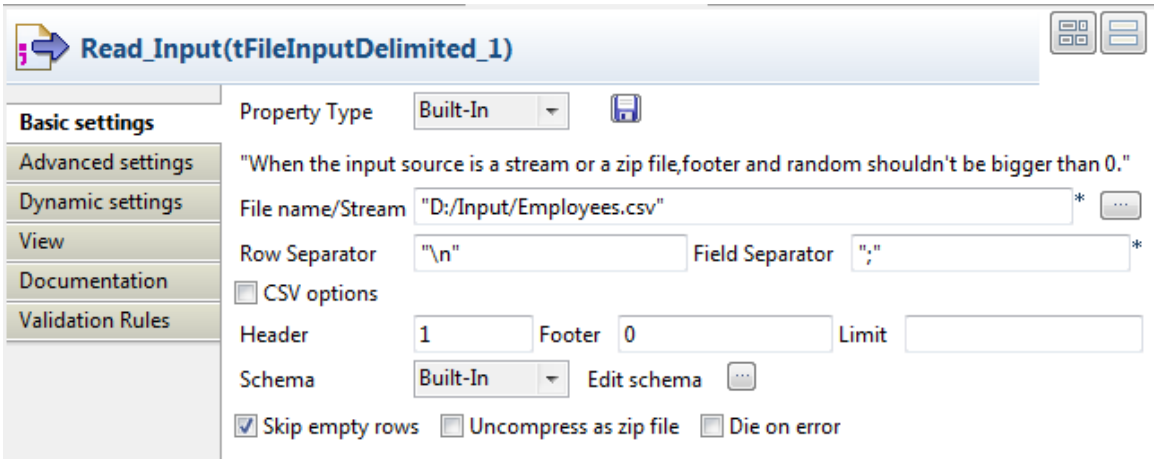
1. Double-click the **tCassandraConnection** component to open its **Basic settings** view in the**Component** tab.



2. In the **Server** field, type in the hostname or IP address of the Cassandra server. In this example, it is *localhost*.
3. In the **Port** field, type in the listening port number of the Cassandra server.
4. In the **Cluster** field, type in the identifying string of the cluster instance.
5. If required, type in the authentication information for the Cassandra connection: **Username** and **Password**.

Reading the input data

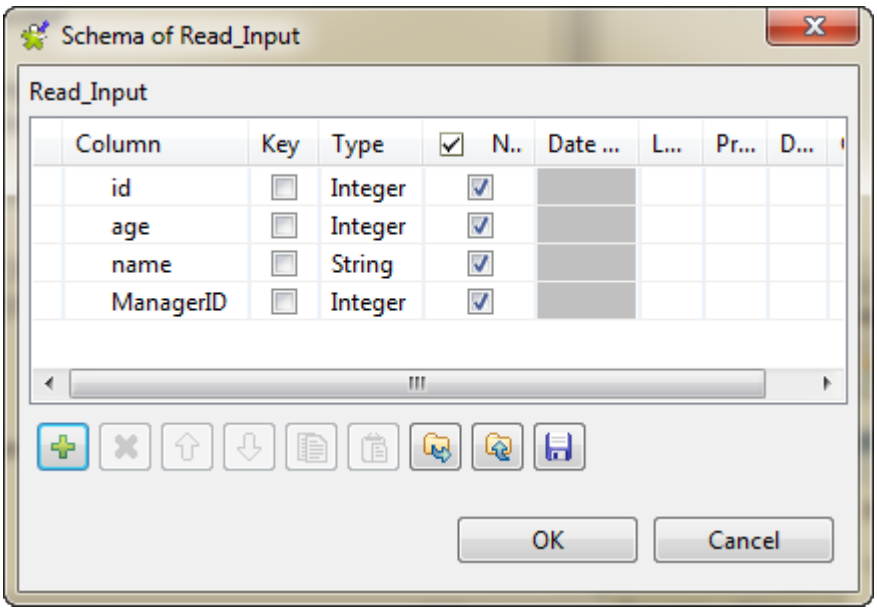
1. Double-click the **tFileInputDelimited** component to open its **Component** view.



2. Click the **[...]** button next to the **File Name/Stream** field to browse to the file that you want to read data from. In this scenario, the directory is *D:/Input/Employees.csv*. The CSV file contains four columns: *id*, *age*, *name* and *ManagerID*.

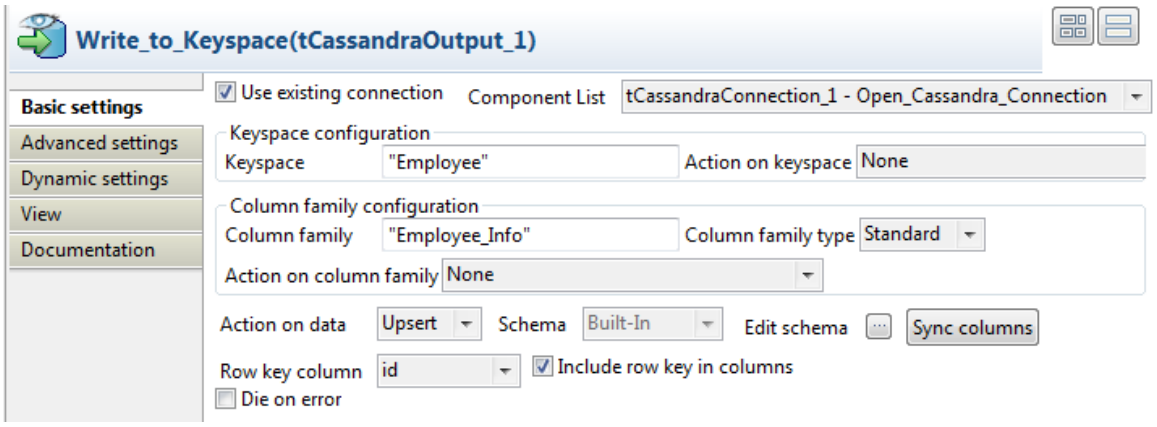
```
id;age;name;ManagerID
1;20;Alex;1
2;40;Peter;1
3;25;Mark;1
4;26;Michael;1
5;30;Christophe;2
6;26;Stephane;3
7;37;Cedric;3
8;52;Bill;4
9;43;Jack;2
10;28;Andrews;4
```

3. Click **Edit schema** to define the data to pass on to the **tCassandraOutput** component.



Writing data to a Cassandra keyspace

1. Double-click the **tCassandraOutput** component to open its **Basic settings** view in the **Component** tab.



2. Type in required information for the connection or use the existing connection you have configured before. In this scenario, the **Use existing connection** check box is selected.

3. In the **Keyspace configuration** area, type in the name of the keyspace: *Employee* in this example.

4. In the **Column family configuration** area, type in the name of the column family: *Employee_Info* in this example.

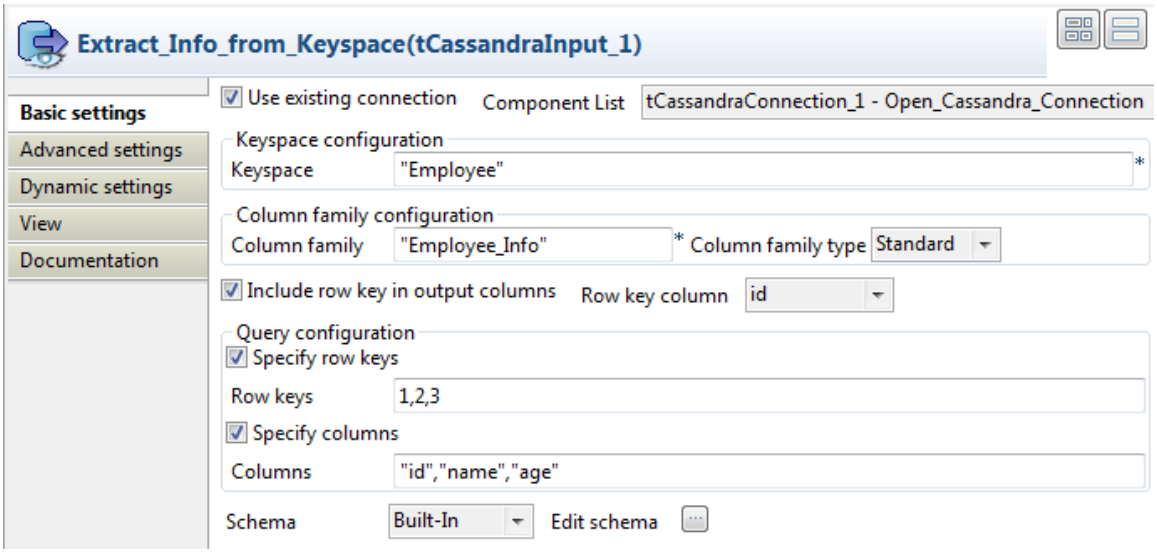
5. In the **Action on data** list, select the action you want to carry on.

6. Click **Sync columns** to retrieve the schema from the preceding component.

7. Select the key column of the column family from the **Key column** list. If needed, select the **Include key in columns** check box.

Reading data from the Cassandra keyspace

1. Double-click the **tCassandraInput** component to open its **Component** view.



2. Type in required information for the connection or use the existing connection you have configured before. In this scenario, the **Use existing connection** check box is selected.

3. In the **Keyspace configuration** area, type in the name of the keyspace: *Employee* in this example.

4. In the **Column family configuration** area, type in the name of the column family: *Employee_Info* in this example.

5. If needed, select the **Include key in output columns** check box, and then select the key column of the column family you want to include from the **Key column** list.

6. In the **Query configuration** area, select the **Specify row keys** check box and specify the row keys directly. In this example, three rows will be read. Next, select the **Specify columns** check box and specify the column names of the column family directly. This scenario will read three columns from the keyspace: *id*, *name* and *age*.

7. If needed, the **Key start** and the **Key end** fields allow you to define the range of rows, and the **Key limit** field allows you to specify the number of rows within the range of rows to be read. Similarly, the **Columns range start** and the **Columns range end** fields allow you to define the range of columns of the column family, and the **Columns range limit** field allows you to specify the number of columns within the range of columns to be read.

8. Select **Edit schema** to define the data structure to be read from the Cassandra keyspace.

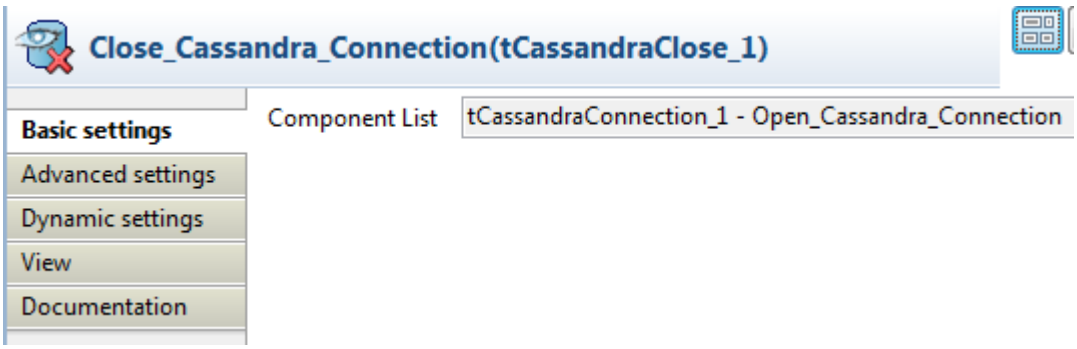
Displaying the information of interest

1. Double-click the **tLogRow** component to open its **Component** view.

2. In the **Mode** area, select **Table (print values in cells of a table)**.

Closing the Cassandra connection

1. Double-click the **tCassandraClose** component to open its **Component** view.

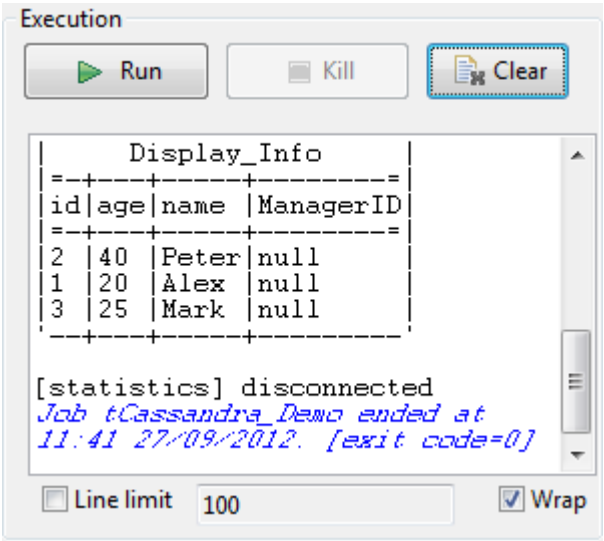


2. Select the connection to be closed from the **Component List**.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Execute the Job by pressing **F6** or clicking **Run** on the **Run** tab.

   The personal information of three employees is displayed on the console.



tCassandraOutput

tCassandraOutput properties

Component family	Big Data / Cassandra	
Function	**tCassandraOutput** receives data from the preceding component, and writes data into Cassandra.	
Purpose	**tCassandraOutput** is used to output data into Cassandra.	
Basic settings	*Use existing connection*	Select this check box and select the appropriate **tCassandraConnection** component from the **Component list** if you want to re-use the connection parameters that you have already defined.
	Host	Hostname or IP address of the Cassandra server.
	Port	Listening port number of the Cassandra server.
Keyspace configuration	*Keyspace*	Type in the name of the keyspace into which you want to write data.
	Action on keyspace	Select the operation you want to perform on the keyspace of interest: • **None**: No operation is carried out. • **Drop and create keyspace**: The keyspace is removed and created again. • **Create keyspace**: The keyspace does not exist and gets created. • **Create keyspace if not exists**: A keyspace gets created if it does not exist. • **Drop keyspace if exists and create**: The keyspace is removed if it already exists and created again.
	Column family configuration	Type in the name of the column family into which you want to write data.
	Column family type	**Standard**: Column family is of standard type. **Super**: Column family is of super type.
	Action on column family	Select the operation you want to perform on the column family of interest: • **None**: No operation is carried out. • **Drop and create column family**: The column family is removed and created again. • **Create column family**: The column family does not exist and gets created. • **Create column family if not exists**: A column family gets created if it does not exist. • **Drop column family if exists and create**: The column family is removed if it already exists and created again.

	Action on data	On the data of the table defined, you can perform: • **Upsert**: Insert the columns if they do not exist or update the existing columns. • **Delete**: Remove columns corresponding to the input flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository** If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Sync columns	Click this button to retrieve schema from the previous component connected in the Job.
	Row key column	Select the row key column from the list.
	Include row key in columns	Select this check box to include row key in columns.
	Super columns	Select the super column from the list. This drop-down list appears only if you select **Super** from the **Column family type** drop-down list.
	Include super columns in standard columns	Select this check box to include the super columns in standard columns.
	Delete row	Select this check box to delete the row. This check box appears only if you select **Delete** from the **Action on data** drop-down list.
	Delete columns	Customize the columns you want to delete.
	Delete super columns	Select this check box to delete super columns. This check box appears only if you select the **Delete Row** check box.
Advanced settings	*Batch Size*	Number of lines in each processed batch.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.

Related Scenario

For a scenario in which **tCassandraOutput** is used, see section *Scenario: Handling data with Cassandra*.

tCassandraRow

tCassandraRow properties

Component Family	Big Data / Cassandra	
Function	**tCassandraRow** is the specific component for this database query. It executes the Cassandra Query Language (CQL) query stated in the specified database. The row suffix means the component implements a flow in the Job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tCassandraRow** acts on the actual DB structure or on the data (although without handling data).	
Basic settings	*Use existing connection*	Select this check box and select the appropriate **tCassandraConnection** component from the **Component List** if you want to re-use the connection parameters that you have already defined.
	Host	Type in the IP address or hostname of the Cassandra server.
	Port	Type in the listening port number of the Cassandra server.
	Required Authentication	Select this check box to provide credentials for the Cassandra authentication. This check box appears only if you do not select the **Use existing connection** check box.
	Username	Fill in this field with the username for the Cassandra authentication.
	Password	Fill in this field with the password for the Cassandra authentication.
Keyspace configuration	*Keyspace*	Type in the name of the keyspace on which you want to execute the CQL commands.
Column family configuration	*Column family*	Name of the column family.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Query	Type in the CQL command to be executed.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Limitation	Currently, the *select* commands are not supported.	

Related scenario

For related topics, see

- section *Scenario 1: Removing and regenerating a MySQL table index*.
- section *Scenario 2: Using PreparedStatement objects to query data*.

tHBaseClose

tHBaseClose properties

Component family	Big Data / HBase	
Function	**tHBaseClose** closes the transaction committed in the connected DB.	
Purpose	This component closes a transaction.	
Basic settings	*Component list*	Select the **tHBaseConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with HBase components, especially with **tHBaseConnection**.	
Limitation	n/a	

Related scenario

For a scenario in which **tHBaseClose** is used, see section *Scenario: Exchanging customer data with HBase* .

tHBaseConnection

tHBaseConnection properties

Component Family	Big Data / HBase	
Function	**tHBaseConnection** opens a connection to a database in order to make a transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	- **Built-in** : no property data stored centrally. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HBase version	Select the version of the Hadoop distribution you are using.
	Zookeeper quorum	Type in the name or the URL of the Zookeeper service you use to coordinate the transaction between **Talend** and HBase.
	Zookeeper client port	Type in the number of the client listening port of the Zookeeper service you are using.
Advanced settings	*Properties*	If you need to use custom configuration for your HBase, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for your HBase. For example, you need to define the value of the *dfs.replication* property as *1* for the HBase configuration. Then you need to add one row to this table using the plus button and type in the name and the value of this property in this row.
	tStatCatcher Statistics	Select this check box to collect the log data at a component level.
Usage	This component is generally used with other HBase components, particularly **tHBaseClose**.	
Limitation	n/a	

Related scenario

For a scenario in which **tHBaseConnection** is used, see section *Scenario: Exchanging customer data with HBase* .

tHBaseInput

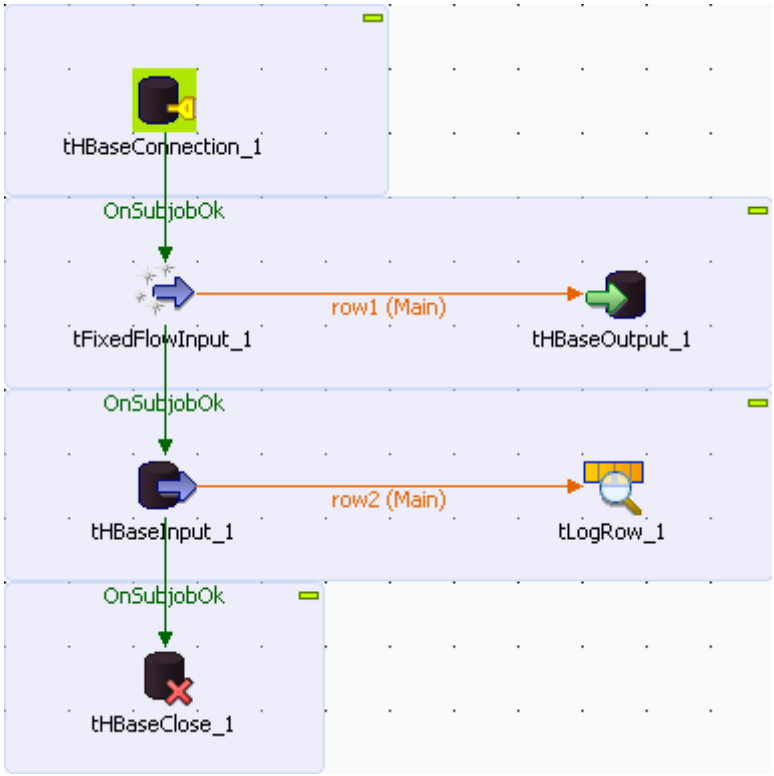
tHBaseInput properties

Component family	Big Data / HBase	
Function	**tHBaseInput** reads the corresponding database and extracts fields of selection.	
Purpose	**tHBaseInput** extracts columns corresponding to schema definition. Then it passes these columns to the next component via a Main row link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. - **Built-in** : No property data stored centrally. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Use an existing connection	Select this check box and select the appropriate **tHBaseConnection** component from the **Component list** if you want to re-use the connection parameters that you have already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HBase version	Select the version of the Hadoop distribution you are using.
	Zookeeper quorum	Type in the name or the URL of the Zookeeper service you use to coordinate the transaction between **Talend** and HBase.
	Zookeeper client port	Type in the number of the client listening port of the Zookeeper service you are using.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Type in the name of the HBase table from which you need to extract columns.
	Mapping	Complete this table to specify the column or columns to be extracted and the corresponding column family or families. The **Column** column of this table is automatically filled once you have defined the schema.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
	Properties	If you need to use custom configuration for your HBase, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for your HBase. For example, you need to define the value of the *dfs.replication* property as *1* for the HBase configuration. Then you need to add one row to this table using the plus button and type in the name and the value of this property in this row.

		This table is not available when you are using an existing connection by selecting the **Using an existing connection** check box in the **Basic settings** view.
Usage	This component is a start component of a Job and always needs an output link.	

Scenario: Exchanging customer data with HBase

In this scenario, a six-component Job is used to exchange customer data with a given HBase.



The six components are:

- **tHBaseConnection**: creates a connection to your HBase database.
- **tFixedFlowInput**: creates the data to be written into your HBase. In the real use case, this component could be replaced by the other input components like **tFileInputDelimited**.
- **tHBaseOutput**: writes the data it receives from the preceding component into your HBase.
- **tHBaseInput**: extracts the columns of interest from your HBase.
- **tLogRow**: presents the execution result.
- **tHBaseClose**: closes the transaction.

To replicate this scenario, proceed as the following sections illustrate.

Before starting the replication, your Hbase and Zookeeper service should have been correctly installed and well configured. This scenario explains only how to use Talend solution to make data transaction with a given HBase.

Dropping and linking the components

To do this, proceed as follows:

1. Drop **tHBaseConnection**, **tFixedFlowInput**, **tHBaseOutput**, **tHBaseInput**, **tLogRow** and **tHBaseClose** from Palette onto the Design workspace.

2. Right-click **tHBaseConnection** to open its contextual menu and select the **Trigger** > **On Subjob Ok** link from this menu to connect this component to **tFixedFlowInput**.

3. Do the same to create the **OnSubjobOk** link from **tFixedFlowInput** to **tHBaseInput** and then to **tHBaseClose**.

4. Right-click **tFixedFlowInput** and select the **Row** > **Main** link to connect this component to **tHBaseOutput**.

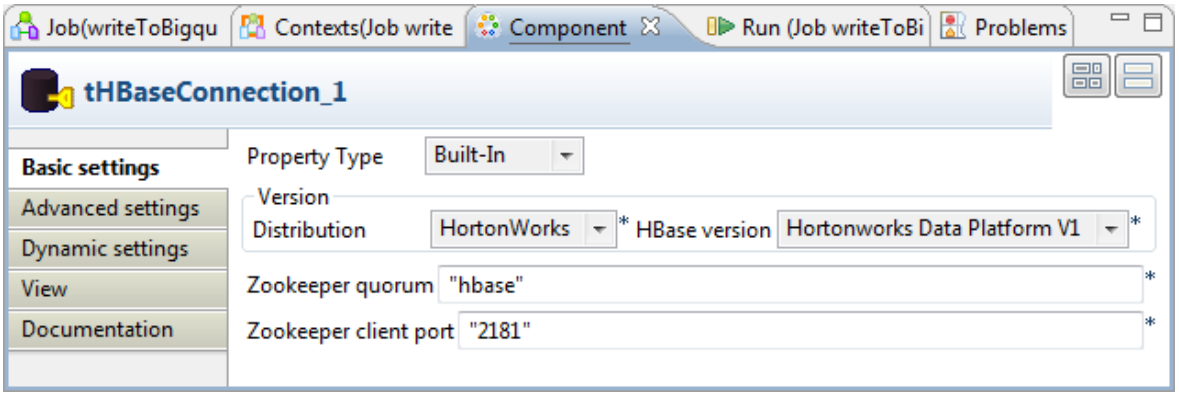
5. Do the same to create the **Main** link from **tHBaseInput** to **tLogrow**.

The components to be used in this scenario are all placed and linked. Then you need continue to configure them sucessively.

Configuring the connection

To configure the connection to your Zookeeper service and thus to the HBase of interest, proceed as follows:

1. On the Design workspace of your Studio, double-click the **tHBaseConnection** component to open its **Component** view.



2. Select **Hortonworks Data Platform 1.0** from the **HBase version** list.

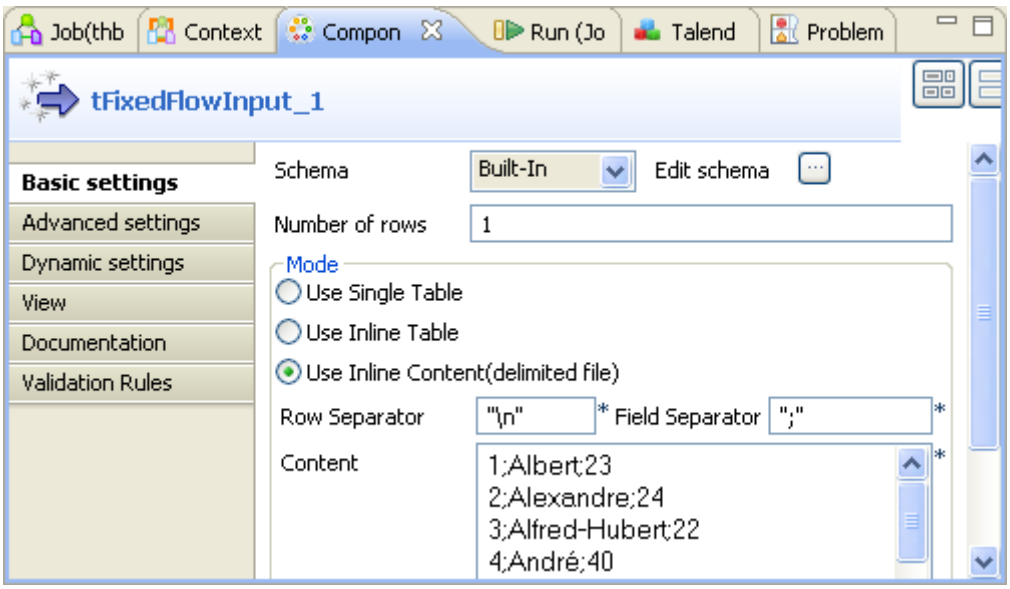
3. In the **Zookeeper quorum** field, type in the name or the URL of the Zookeeper service you are using. In this example, the name of the service in use is *hbase*.

4. In the **Zookeeper client port** field, type in the number of client listening port. In this example, it is *2181*.

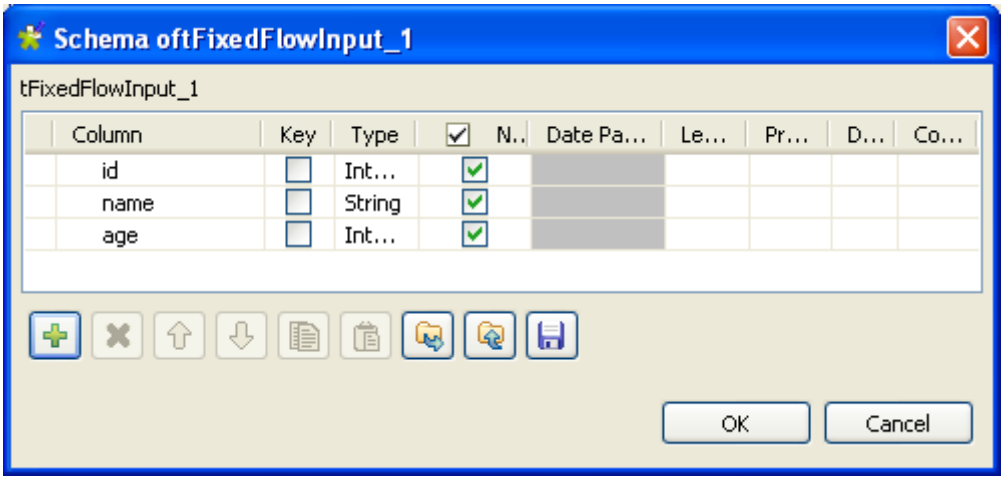
Configuring the process of writing data into the HBase

To do this, proceed as follows:

1. On the Design workspace, double-click the **tFixedFlowInput** component to open its **Component** view.



2. In this view, click the three-dot button next to **Edit schema** to open the schema editor.



3. Click the plus button three times to add three rows and in the **Column** column, rename the three rows respectively as: *id*, *name* and *age*.

4. In the **Type** column, click each of these rows and from the drop-down list, select the data type of every row. In this scenario, they are **Integer** for *id* and *age*, **String** for *name*.

5. Click **OK** to validate these changes and accept the propagation prompted by the pop-up dialog box.

6. In the **Mode** area, select the **Use Inline Content (delimited file)** to display the fields for editing.

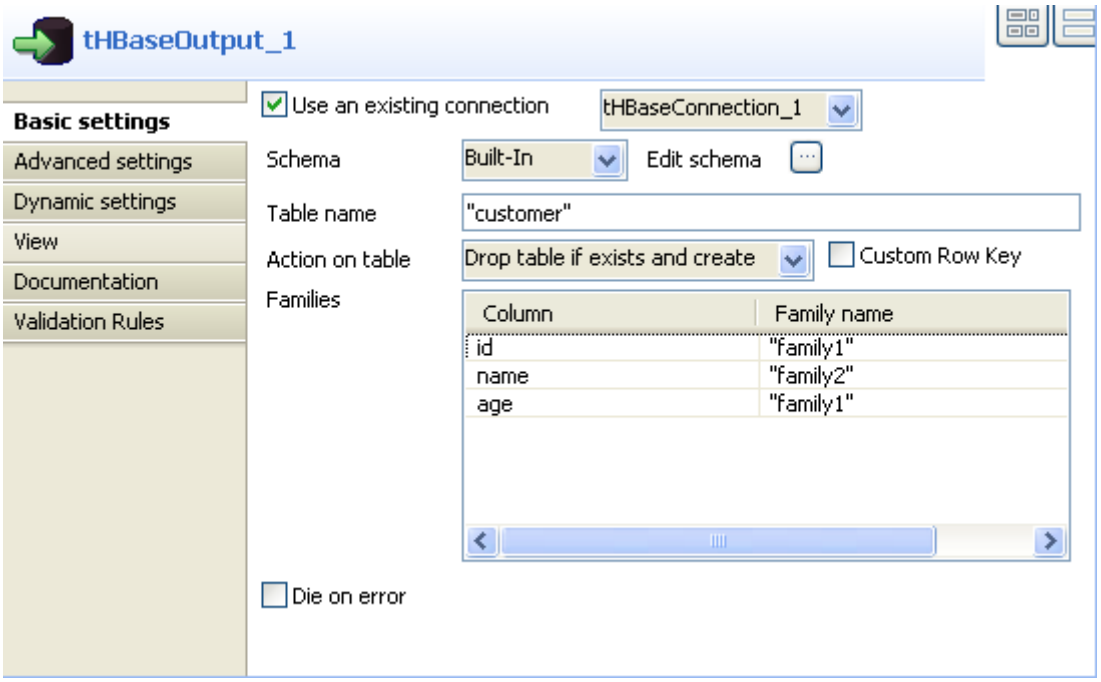
7. In the **Content** field, type in the delimited data to be written into the HBase, separated with the semicolon ";". In this example, they are:

```
1;Albert;23
2;Alexandre;24
3;Alfred-Hubert;22
4;André;40
5;Didier;28
6;Anthony;35
7;Artus;32
8;Benoît;56
9;Catherine;34
10;Charles;21
11;Christophe;36
```

```
12;Christian;67
13;Clément ;64
14;Danniel;54
15;Elisabeth;58
16;Emile;32
17;Gregory;30
```

8. Double-click **tHBaseOutput** to open its **Component** view.

If this component does not have the same schema of the preceding component, a warning icon appears. In this case, click the **Sync columns** button to retrieve the schema from the preceding one and once done, the warning icon disappears.

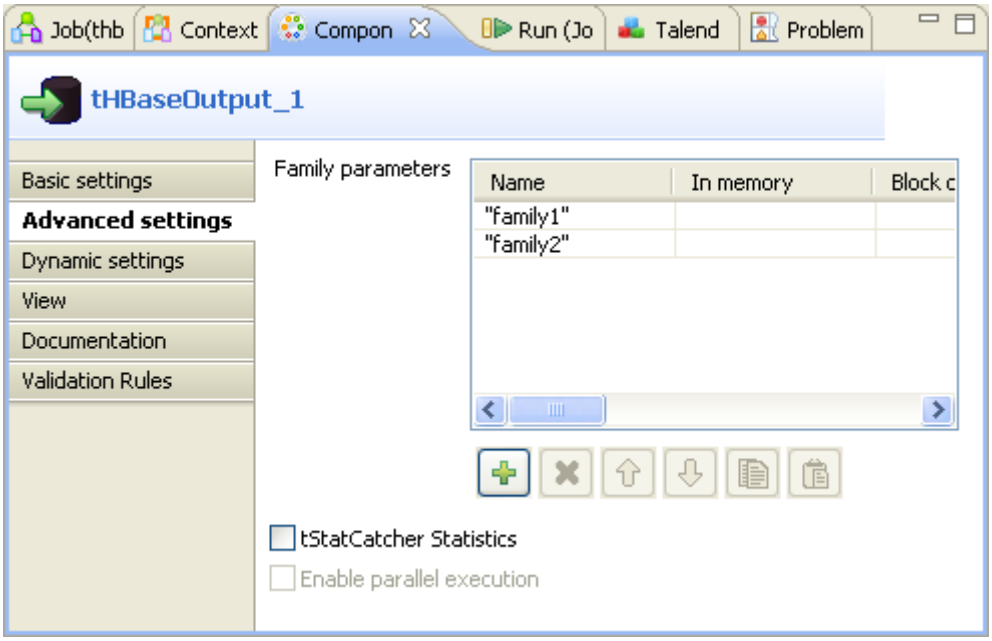


9. Select the **Use an existing connection** check box and then select the connection you have configured earlier. In this example, it is *tHBaseConnection_1*.

10. In the **Table name** field, type in the name of the table to be created in the HBase. In this example, it is *customer*.

11. In the **Action on table** field, select the action of interest from the drop-down list. In this scenario, select **Drop table if exists and create**. This way, if a table named customer exists already in the HBase, it will be disabled and deleted before creating this current table.

12. Click the **Advanced settings** tab to open the corresponding view.



13. In the **Family parameters** table, add two rows by clicking the plus button, rename them as *family1* and *family2* respectively and then leave the other columns empty. These two column families will be created in the HBase using the default family performance options.

The **Family parameters** table is available only when the action you have selected in the **Action on table** field is to create a table in HBase. For further information about this **Family parameters** table, see section *tHBaseOutput*.

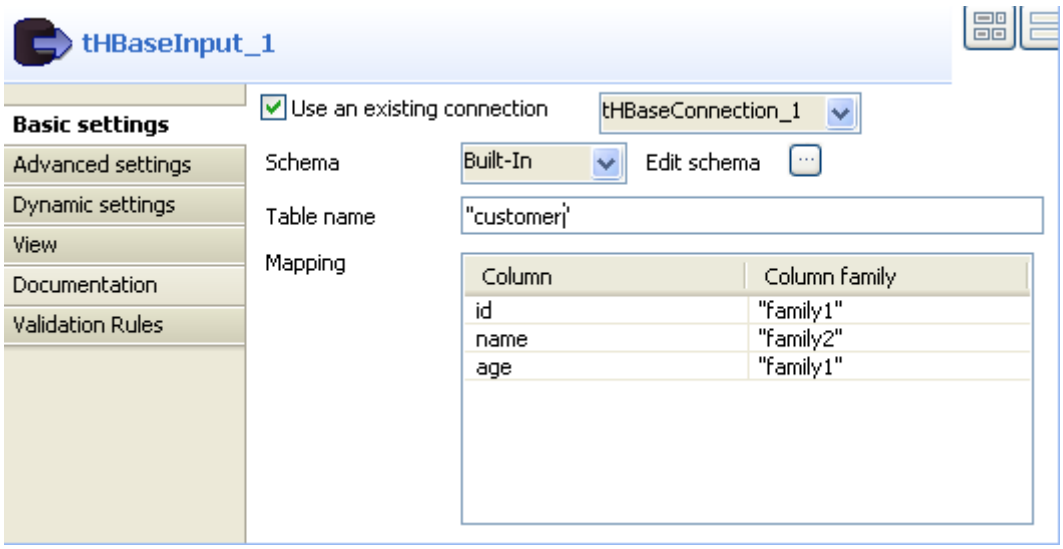
14. In the **Families** table of the **Basic settings** view, enter the family names in the **Family name** column, each corresponding to the column this family contains. In this example, the *id* and the *age* columns belong to *family1* and the *name* column to *family2*.

These column families should already exist in the HBase to be connected to; if not, you need to define them in the **Family parameters** table of the **Advanced settings** view for creating them at runtime.

Configuring the process of extracting data from the HBase

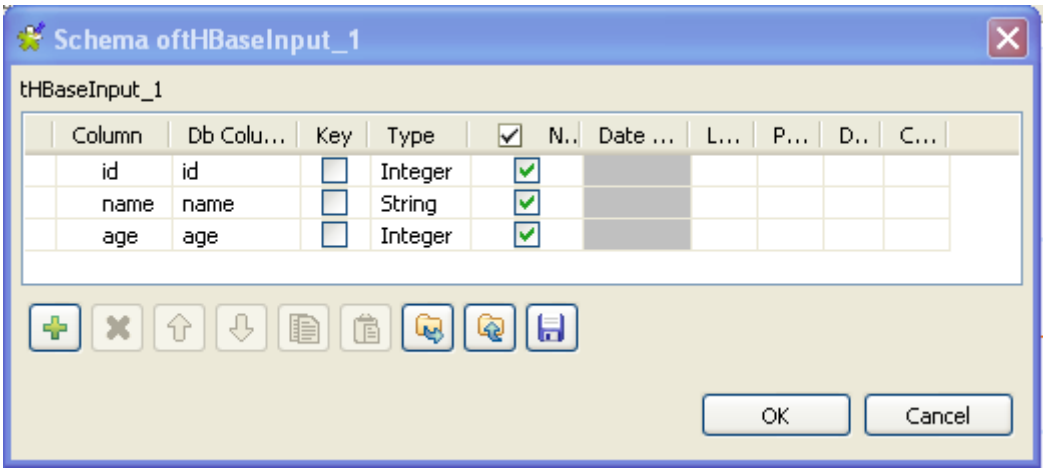
To do this, perform the following operations:

1. Double-click **tHBaseInput** to open its **Component** view.



2. Select the **Use an existing connection** check box and then select the connection you have configured earlier. In this example, it is *tHBaseConnection_1*.

3. Click the three-dot button next to **Edit schema** to open the schema editor.



4. Click the plus button three times to add three rows and rename them as *id*, *name* and *age* respectively in the **Column** column. This means that you extract these three columns from the HBase.

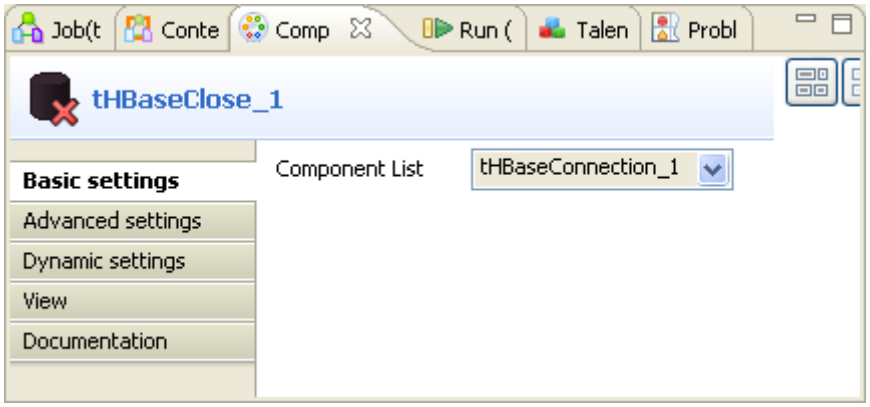
5. Select the types for each of the three columns. In this example, **Integer** for *id* and *age*, **String** for *name*.

6. Click **OK** to validate these changes and accept the propagation prompted by the pop-up dialog box.

7. In the **Table name** field, type in the table from which you extract the columns of interest. In this scenario, the table is *customer*.

8. In the **Mapping** table, the **Column** column has been already filled automatically since the schema was defined, so simply enter the name of every family in the **Column family** column, each corresponding to the column it contains.

9. Double-click **tHBaseClose** to open its **Component** view.

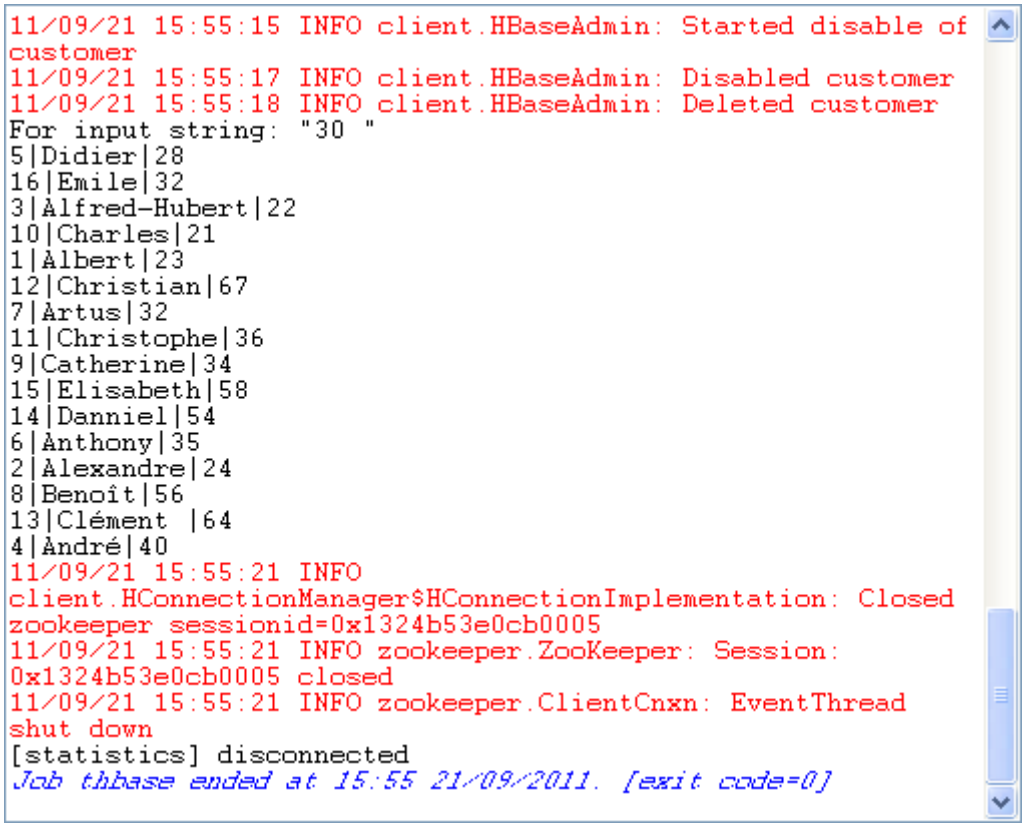


10. In the **Component List** field, select the connection you need to close. In this example, this connection is *tHBaseConnection_1*.

Executing the Job

To execute this Job, press **F6**.

Once done, the **Run** view is opened automatically, where you can check the execution result.



These columns of interest are extracted and you can process them according to your needs.

Login to your HBase database, you can check the *customer* table this Job has created.

tHBaseOutput

tHBaseOutput properties

Component family	Big Data / HBase	
Function	**tHBaseOutput** receives data from its preceding component, creates a table in a given HBase database and writes the received data into this HBase table.	
Purpose	**tHBaseOutput** writes columns of data into a given tHBase database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. - **Built-in** : No property data stored centrally. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Use an existing connection	Select this check box and select the appropriate **tHBaseConnection** component from the **Component list** if you want to re-use the connection parameters that you have already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HBase version	Select the version of the Hadoop distribution you are using.
	Zookeeper quorum	Type in the name or the URL of the Zookeeper service you use to coordinate the transaction between **Talend** and HBase.
	Zookeeper client port	Type in the number of the client listening port of the Zookeeper service you are using.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Type in the name of the HBase table you need create.
	Action on table	Select the action you need to take for creating an HBase table.
	Custom Row Key	Select this check box to use the customized row keys. Once selected, the corresponding field appears. Then type in the user-defined row key to index the rows of the HBase table being created. For example, you can type in `"France"+Numeric.sequence("s1",1,1)` to produce the row key series: *France1*, *France2*, *France3* and so on.
	Families	Complete this table to specify the column or columns to be created and the corresponding column family or families they belong to respectively. The **Column** column of this table is automatically filled once you have defined the schema.
Advanced settings	*Properties*	If you need to use custom configuration for your HBase, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for your HBase.

		For example, you need to define the value of the *dfs.replication* property as *1* for the HBase configuration. Then you need to add one row to this table using the plus button and type in the name and the value of this property in this row. This table is not available when you are using an existing connection by selecting the **Using an existing connection** check box in the **Basic settings** view.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Only available for creating a HBase table	*Family parameters*	Type in the names and, when needs be, the custom performance options of the column families to be created. These options are all attributes defined by the HBase data model, so for further explanation about these options, see your HBase manual.
Usage	This component is normally an end component of a Job and always needs an input link.	

Related scenario

For related scenario to **tHBaseOutput**, see section *Scenario: Exchanging customer data with HBase* .

tHCatalogInput

tHCatalogInput Properties

Component family	Big Data / HCatalog	
Function	This component allows you to read data from HCatalog managed database and send data in the talend flow.	
Purpose	The **tHCatalogInput** component reads data from the specified HCatalog managed database and sends data in the talend flow to the console or to a specified local file by connecting this component to a proper component.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HCatalog version	Select the version of the Hadoop distribution you are using.
	Templeton hostname	Fill this field with the URL of Templeton Webservice. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Templeton port	Fill this field with the port of URL of Templeton Webservice. By default, this value is *50111*. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Database	The database in which the HCatalog managed tables are placed.
	Table	Fill this field to operate on one or multiple tables in the specified database.
	Partition	Fill this field to specify one or more partitions for the partition operation on a specified table. When you specify multiple partitions, use commas to separate every two partitions and use double quotation marks to quote the partition string.

		For further information about **Partition**, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-AddPartitions.
	Username	Fill this field with the username for the DB authentication.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Row separator*	Enter the separator used to identify the end of a row.
	Field separator	Enter the separator used to identify the end of a field.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Retrieve the HCatalog logs	Select this check box to retrieve log files generated during HCatalog operations.
	Standard Output Folder	Fill this field with the path to which log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	Error Output Folder	Fill this field with the path to which error log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as the starting component in a Job.	
Limitation	Knowledge of Hive Data Definition Language and HCatalog Data Definition Language is required. For further information about Hive Data Definition Language, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL. For further information about HCatalog Data Definition Language, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.	

Related scenario

For a related scenario, see section *Scenario: HCatalog table management on Hortonworks Data Platform*.

tHCatalogLoad

tHCatalogLoad Properties

Component family	Big Data / HCatalog	
Function	This component allows you to write data into an established HCatalog managed table from an existing file in HDFS.	
Purpose	The **tHCatalogLoad** component writes data into an established HCatalog managed table from an existing file in HDFS.	
Basic settings	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HCatalog version	Select the version of the Hadoop distribution you are using.
	Templeton hostname	Fill this field with the URL of Templeton Webservice. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see Templeton 0.1.0.
	Templeton port	Fill this field with the port of URL of Templeton Webservice. By default, this value is *50111*. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see Templeton 0.1.0.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Database	Fill this field to specify an existing database in HDFS.
	Table	Fill this field to specify an existing table in HDFS.
	Partition	Fill this field to specify one or more partitions for the partition operation on the specified table. When you specify multiple partitions, use commas to separate every two partitions and use double quotation marks to quote the partition string. For further information about **Partition**, see Operation on Partitions in Hive.
	Username	Fill this field with the username for the DB authentication.
	File location	Fill this field with the HDFS location to which loaded data is stored.
Advanced settings	*Retrieve the HCatalog logs*	Select this check box to retrieve log files generated during HCatalog operations.
	Standard Output Folder	Fill this field with the path to which log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	Error Output Folder	Fill this field with the path to which error log files are stored.

		This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component can be used in a single-component Job or used together with a subjob.	
Limitation	Knowledge of Hive Data Definition Language and HCatalog Data Definition Language is required. For further information about Hive Data Definition Language, see Hive Data Definition Language. For further information about HCatalog Data Definition Language, see HCatalog Data Definition Language.	

Related scenario

For a related scenario, see section *Scenario: HCatalog table management on Hortonworks Data Platform*.

tHCatalogOperation

tHCatalogOperation Properties

Component family	Big Data / HCatalog	
Function	This component allows you to manage the data stored in HCatalog managed database/table/partition in HDFS.	
Purpose	The **tHCatalogOperation** component offers a platform on which you can operate in HCatalog managed database/table/partition in HDFS.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HCatalog version	Select the version of the Hadoop distribution you are using.
	Templeton hostname	Fill this field with the URL of Templeton Webservice. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Templeton port	Fill this field with the port of URL of Templeton Webservice. By default, the value for this field is *50111*. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Operation on	Select an object from the list for the DB operation as follows: **Database**: The HCatalog managed database in HDFS. **Table**: The HCatalog managed table in HDFS. **Partition**: The partition specified by the user.
	Operation	Select an action from the list for the DB operation as follows: **Create**/**Drop**/**Drop if exist**/**Drop and create**/**Drop if exist and create**. For further information about the DB operation in HDFS, see https://cwiki.apache.org/Hive/languagemanual-ddl.html.

	Create the table only it doesn't exist already	Select this check box to avoid creating duplicate table when you create a table. This check box is enabled only when you select **Table** from the **Operation on** list.
	Database	Fill this field with the name of the database in which the HCatalog managed tables are placed.
	Table	Fill this field to operate on one or multiple tables in a database or on a specified HDFS location. This field is enabled only when you select **Table** from the **Operation on** list. For further information about the operation on **Table**, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-Create%252FDropTable.
	Partition	Fill this field to specify one or more partitions for the partition operation on a specified table. When you specify multiple partitions, use comma to separate every two partitions and use double quotation marks to quote the partition string. This field is enabled only when you select **Partition** from the **Operation on** list. For further information about the operation on **Partition**, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-AddPartitions.
	Username	Fill this field with the username for the DB authentication.
	Database location	Fill this field with the location of the database file in HDFS. This field is enabled only when you select **Database** from the **Operation on** list.
	Database description	The description for the database to be created. This field is enabled only when you select **Database** from the **Operation on** list.
	Create an external table	Select this field to create an external table in an alternative path defined in the **Set HDFS location** field in the **Advanced settings** view. For further information about creating external table, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-CreateTable. This check box is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list.
	Format	Select a file format from the list to specify the format of the external table you want to create: **TEXTFILE**: Plain text files. **RCFILE**: Record Columnar files. For further information about RCFILE, see http://hive.apache.org/docs/r0.4.0/api/org/apache/hadoop/hive/ql/io/RCFile.html. **RCFILE** is only available starting with Hive 0.6.0. This list is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list.
	Set partitions	Select this check box to set the partition schema by clicking the **Edit schema** to the right of **Set partitions** check box. The partition schema is either built-in or remote in the Repository. This check box is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list. You must follow the rules of using

		partition schema in HCatalog managed tables. For more information about the rules in using partition schema, see http://incubator.apache.org/hcatalog/docs/r0.2.0/inputoutput.html#Partition+Schema+Semantics. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Set the user group to use	Select this check box to specify the user group. This check box is enabled only when you select **Drop**/**Drop if exist**/**Drop and create**/**Drop if exist and create** from the **Operation** list. By default, the value for this field is *root*. For more information about the user group in the server, contact your system administrator.
	Option	Select a clause when you drop a database. This list is enabled only when you select **Database** from the **Operation on** list and **Drop**/**Drop if exist**/**Drop and create**/**Drop if exist and create** from the **Operation** list. For more information about **Drop** operation on database, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-DropDatabase.
	Set the permissions to use	Select this check box to specify the permissions needed by the operation you select from the **Operation** list. This check box is enabled only when you select **Drop**/**Drop if exist**/**Drop and create**/**Drop if exist and create** from the **Operation** list. By default, the value for this field is *rwxrw-r-x*. For more information on user permissions, contact your system administrator.
	Set File location	Fill this field to specify a path to which partitioned data is stored. This check box is enabled only when you select **Partition** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list. For further information about storing partitioned data in HDFS, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-AddPartitions.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Comment*	Fill this field with the comment for the table you want to create. This field is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list in the **Basic settings view**.
	Set HDFS location	Select this check box to specify an HDFS location to which the table you want to create is saved. Deselect it to save the table you want to create in the warehouse directory defined in the key *hive.metastore.warehouse.dir* in Hive configuration file *hive-site.xml*. This check box is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list in the **Basic settings** view. For further information about saving data in HDFS, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-CreateTable.
	Set row format(terminated by)	Select this check box to use and define the row formats when you want to create a table:

		Field: Select this check box to use **Field** as the row format. The default value for this field is *"\u0001"*. You can also specify a customized char in this field. **Collection Item**: Select this check box to use **Collection Item** as the row format. The default value for this field is *"\u0002"*. You can also specify a customized char in this field. **Map Key**: Select this check box to use **Map Key** as the row format. The default value for this field is *"\u0003"*. You can also specify a customized char in this field. **Line**: Select this check box to use **Line** as the row format. The default value for this field is *"\n"*. You can also specify a customized char in this field. This check box is enabled only when you select **Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list in the **Basic settings** view. For further information about row formats in the HCatalog managed table, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-CreateTable.
	Properties	Click **[+]** to add one or more lines to define table properties. The table properties allow you to tag the table definition with your own metadata key/value pairs. Make sure that values in both **Key** row and **Value** row must be quoted in double quotation marks. This table is enabled only when you select **Database/Table** from the **Operation on** list and **Create/Drop and create/Drop if exist and create** from the **Operation** list in the **Basic settings** view. For further information about table properties, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-CreateTable.
	Retrieve the HCatalog logs	Select this check box to retrieve log files generated during HCatalog operations.
	Standard Output Folder	Browse to, or enter the directory where the log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	Error Output Folder	Browse to, or enter the directory where the error log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used in a single-component Job or used together with a subjob.	
Limitation	Knowledge of Hive Data Definition Language and HCatalog Data Definition Language is required. For further information about Hive Data Definition Language, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL. For further information about HCatalog Data Definition Language, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.	

Scenario: HCatalog table management on Hortonworks Data Platform

This scenario describes a six-component Job that includes the common operations for the HCatalog table management on Hortonworks Data Platform. Sub-sections in this scenario covers DB operations including:

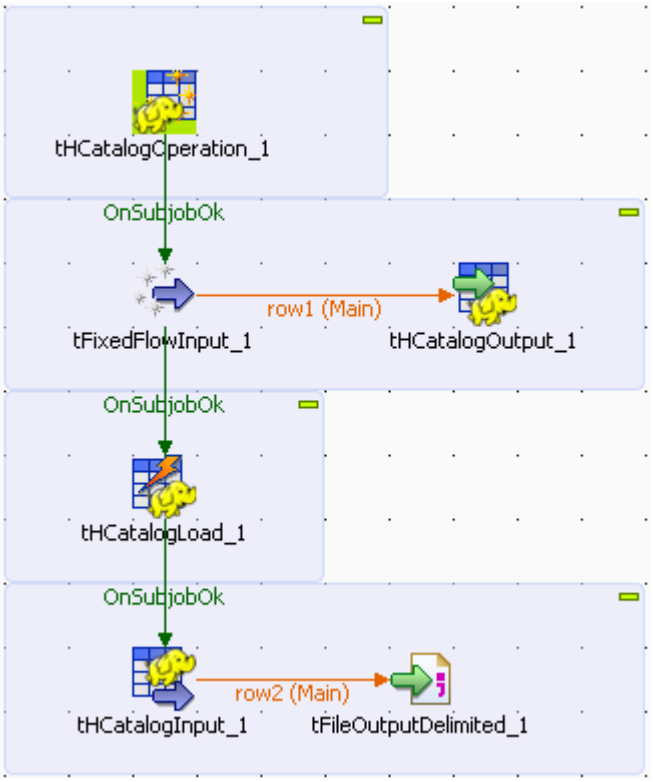
- Creating a table to the database in HDFS;

- Writing data to the HCatalog managed table;
- Writing data to the partitioned table using **tHCatalogLoad**;
- Reading data from the HCatalog managed table;
- Outputting the data read from the table in HDFS.

Knowledge of Hive Data Definition Language and HCatalog Data Definition Language is required. For further information about Hive Data Definition Language, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL. For further information about HCatalog Data Definition Language, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.

Setting up the Job

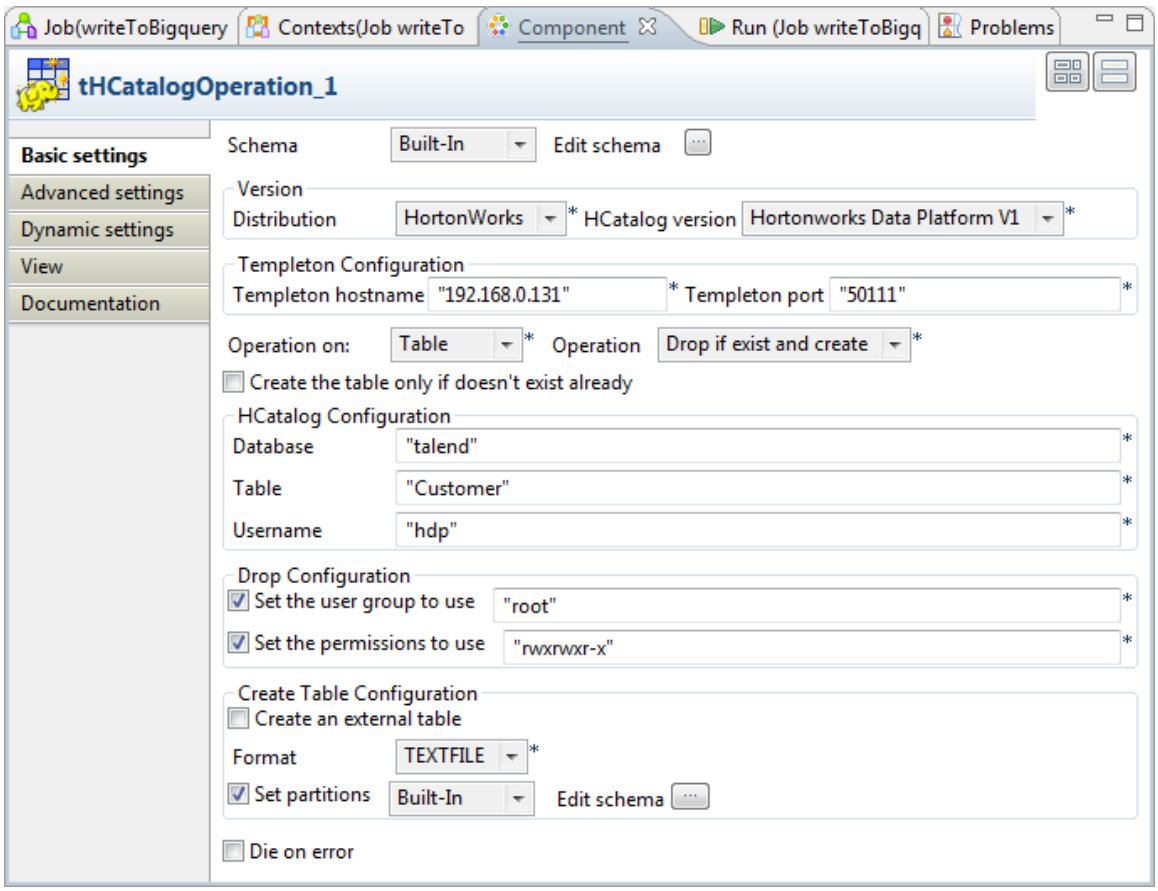
1. Drop the following components from the **Palette** to the design workspace: **tHCatalogOperation**, **tHCatalogLoad**, **tHCatalogInput**, **tHCatalogOutput**, **tFixedFlowInput**, and **tFileOutputDelimited**.



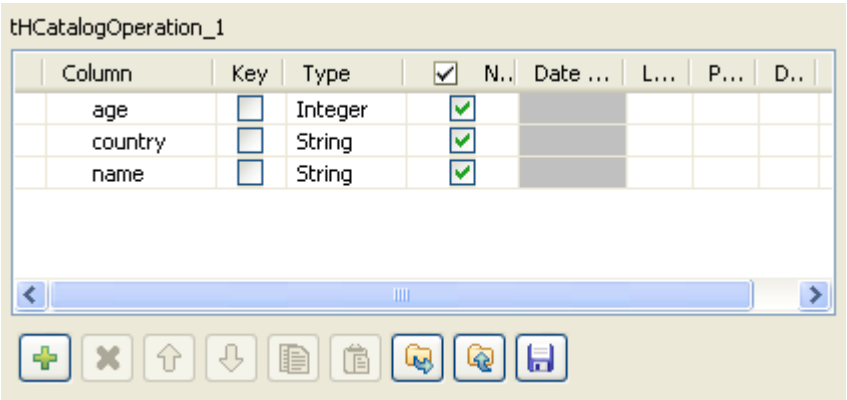
2. Right-click **tHCatalogOperation** to connect it to **tFixedFlowInput** component using a **Trigger>OnSubjobOk** connection.
3. Right-click **tFixedFlowInput** to connect it to **tHCatalogOutput** using a **Row** > **Main** connection.
4. Right-click **tFixedFlowInput** to connect it to **tHCatalogLoad** using a **Trigger** > **OnSubjobOk** connection.
5. Right-click **tHCatalogLoad** to connect it to the **tHCatalogInput** component using a **Trigger** > **OnSubjobOk** connection.
6. Right-click **tHCatalogInput** to connect it to **tFileOutputDelimited** using a **Row** > **Main** connection.

Creating a table in HDFS

1. Double-click **tHCatalogOperation** to open its **Basic settings** view.



2. Click **Edit schema** to define the schema for the table to be created.



3. Click **[+]** to add at least one column to the schema and click **OK** when you finish setting the schema. In this scenario, the columns added to the schema are: *name*, *country* and *age*.

4. Fill the **Templeton hostname** field with URL of the Templeton webservice you are using. In this scenario, fill this field with "*192.168.0.131*".

5. Fill the **Templeton port** field with the port for **Templeton hostname**. By default, the value for this field is "*50111*"

6. Select **Table** from the **Operation on** list and **Drop if exist and create** from the **Operation** list to create a table in HDFS.

7. Fill the **Database** field with an existing database name in HDFS. In this scenario, the database name is "*talend*".

8. Fill the **Table** field with the name of the table to be created. In this scenario, the table name is "*Customer*".

9. Fill the **Username** field with the username for the DB authentication.

10. Select the **Set the user group to use** check box to specify the user group. The default user group is "*root*", you need to specify the value for this field according to real practice.

11. Select the **Set the permissions to use** check box to specify the user permission. The default value for this field is "*rwxrwxr-x*".

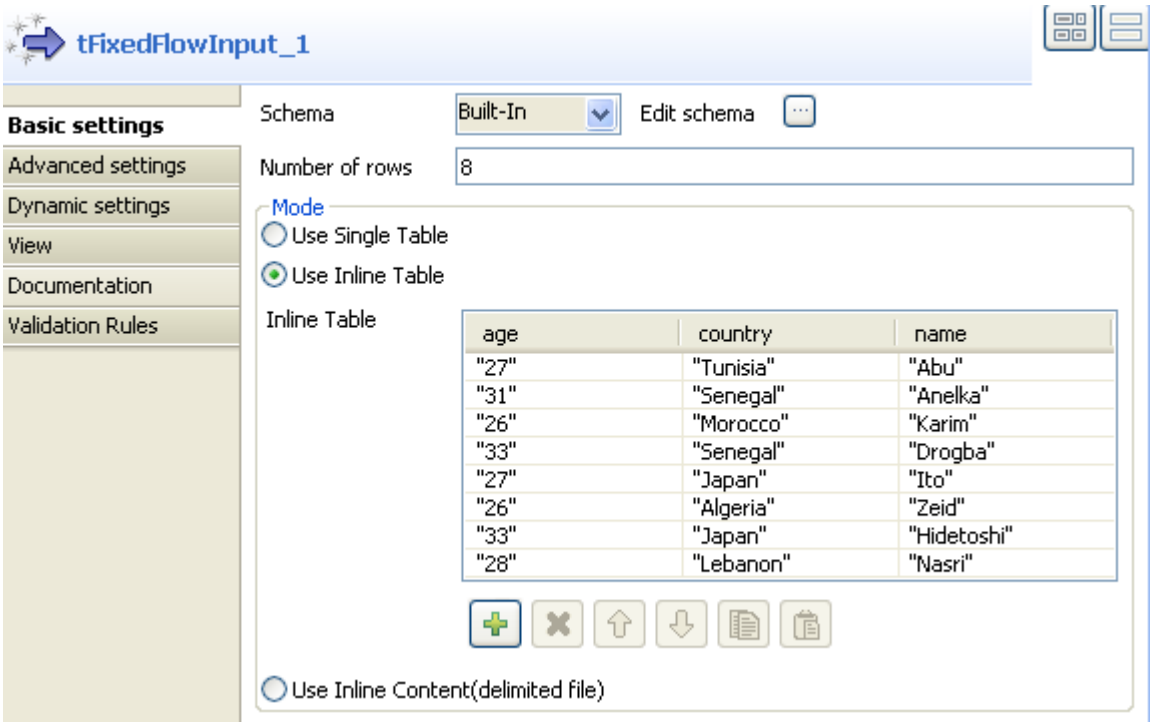
12. Select the **Set partitions** check box to enable the partition schema.

13. Click the **Edit schema** button next to the **Set partitions** check box to define the partition schema.

14. Click **[+]** to add one column to the schema and click **OK** when you finish setting the schema. In this scenario, the column added to the partition schema is: *match_age*.

Writing data to the existing table

1. Double-click **tFixedFlowInput** to open its **Basic settings** view.



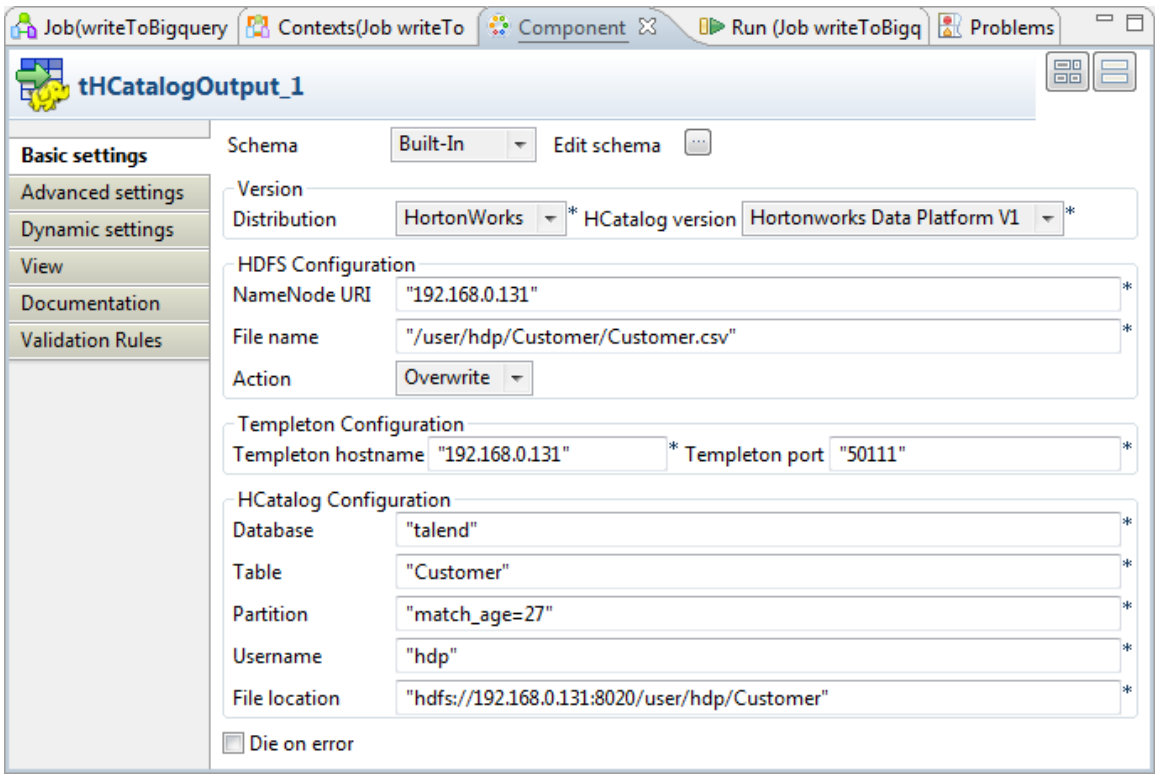
2. Click **Edit schema** to define a same schema as the one you defined in **tHCatalogOperation**.

3. Fill the **Number of rows** field with integer *8*.

4. Select **Use Inline Table** in the **Mode** area.

5. Click **[+]** to add new lines in the inline table.

6. Double-click **tHCatalogOutput** to open its **Basic settings** view.



7. Click **Sync columns** to retrieve the schema defined in the preceding component.

8. Fill the **NameNode URI** field with the URI to the NameNode. In this scenario, this URL is "*192.168.0.131*".

9. Fill the **File name** field with the HDFS location of the file you write data to. In this scenario, the file location is "*/user/hdp/Customer/Customer.csv*".

10. Select **Overwrite** from the **Action** list.

11. Fill the **Templeton hostname** field with URL of the Templeton webservice you are using. In this scenario, fill this field with "*192.168.0.131*".

12. Fill the **Templeton port** field with the port for **Templeton hostname**. By default, the value for this field is "*50111*"

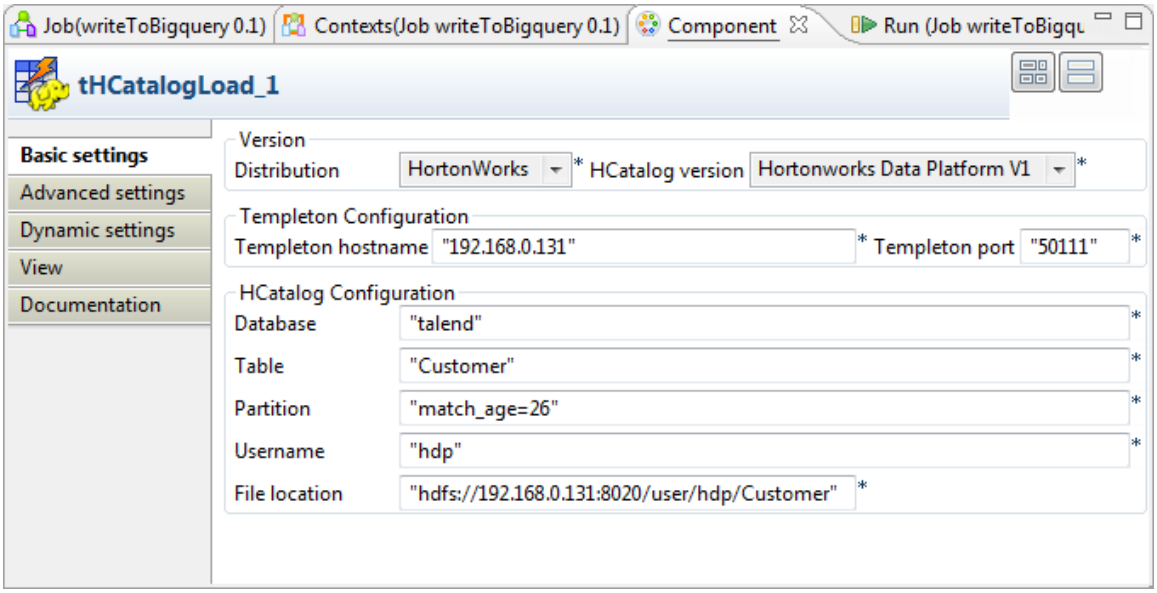
13. Fill the **Database** field, the **Table** field, the **Username** field with the same value you specified in **tHCatalogOperation**.

14. Fill the **Partition** field with "*match_age=27*".

15. Fill the **File location** field with the HDFS location to which the table will be saved. In this example, use "*hdfs://192.168.0.131:8020/user/hdp/Customer*".

Writing data to the partitioned table using tHCatalogLoad

1. Double-click **tHCatalogLoad** to open its **Basic settings** view.

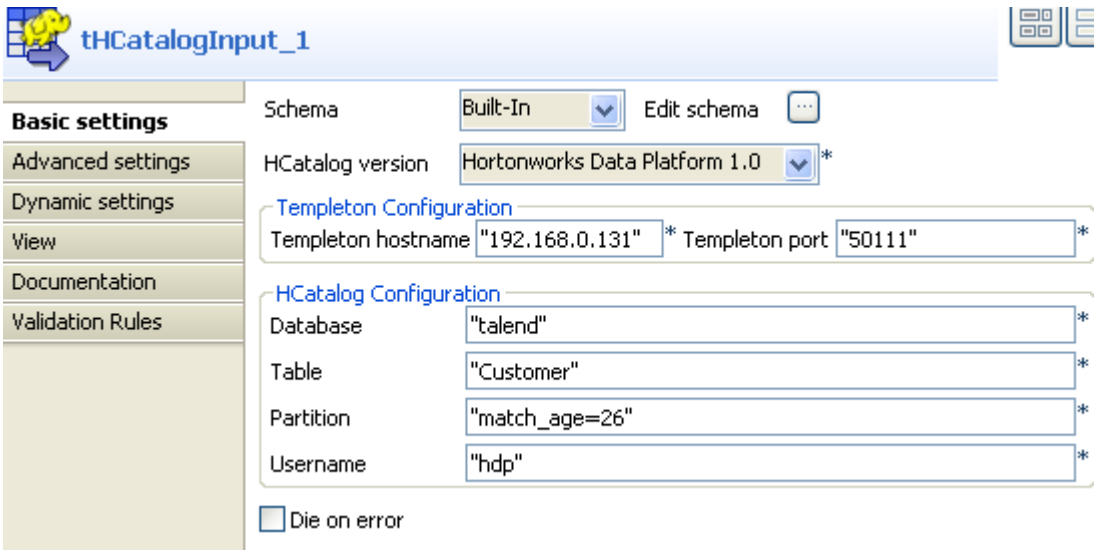


2. Fill the **Partition** field with "*match_age=26*".

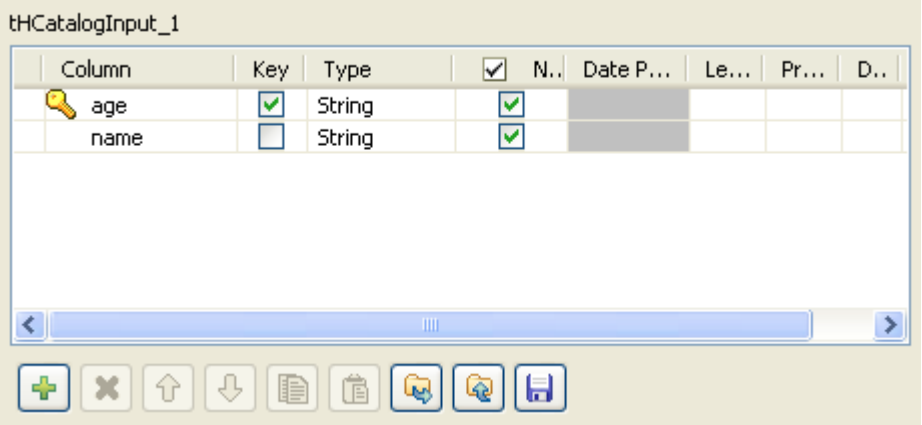
3. Do the rest of the settings in the same way as configuring **tHCatalogOperation**.

Reading data from the table in HDFS

1. Double-click **tHCatalogInput** to open its **Basic settings** view.



2. Click **Edit schema** to define the schema of the table to be read from the database.



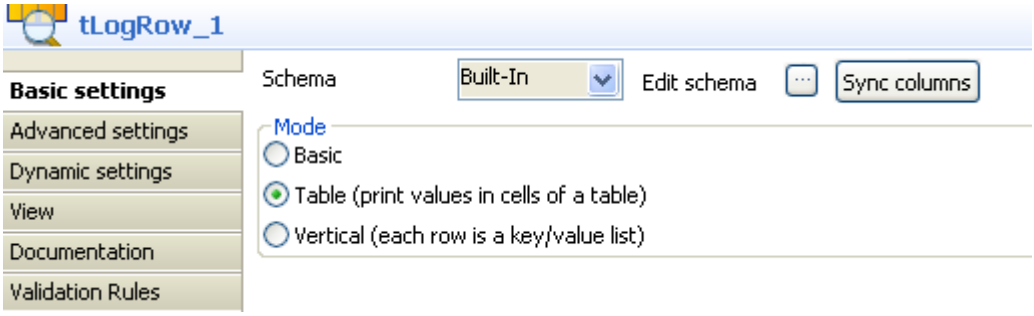
3. Click **[+]** to add at least one column to the schema. In this scenario, the columns added to the schema are *age* and *name*.

4. Fill the **Partition** field with "*match_age=26*".

5. Do the rest of the settings in the same way as configuring **tHCatalogOperation**.

Outputting the data read from the table in HDFS to the console

1. Double-click **tLogRow** to open its **Basic settings** view.



2. Click **Sync columns** to retrieve the schema defined in the preceding component.

3. Select **Table** from the **Mode** area.

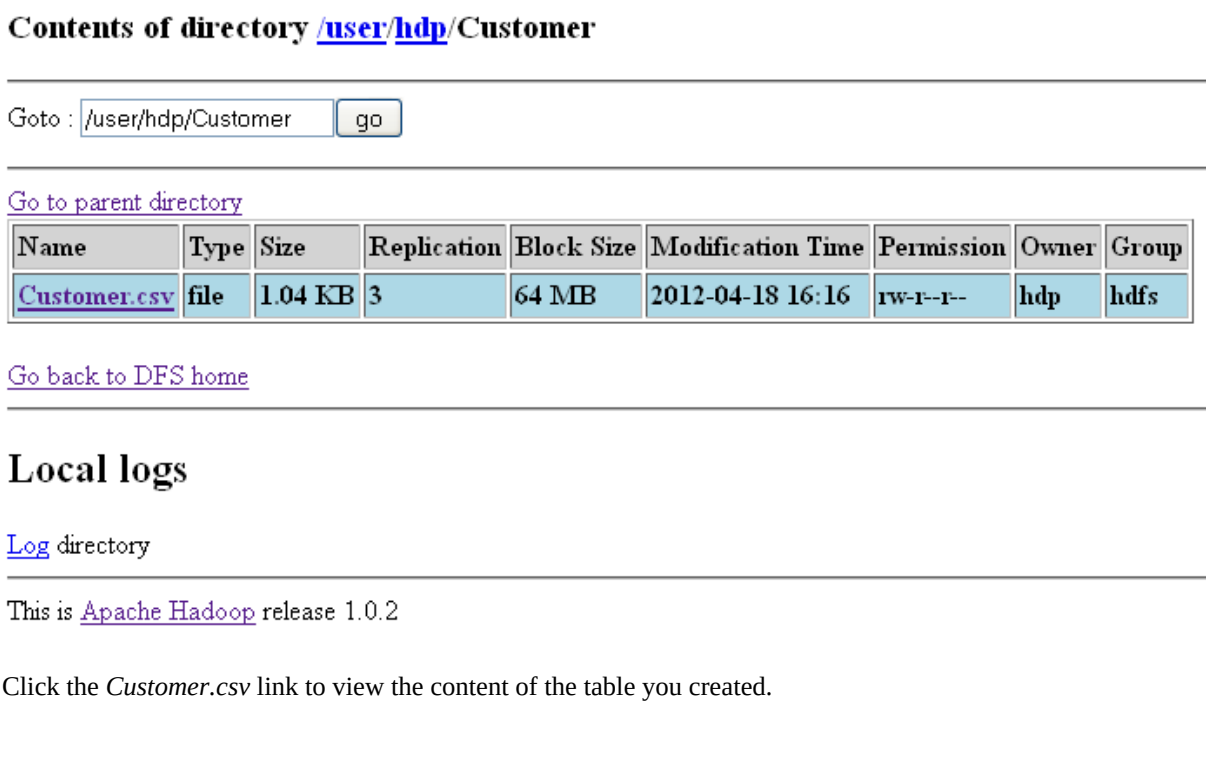
Job execution

Press **CTRL**+**S** to save your Job and **F6** to execute it.

```
+---+-------+
| tLogRow_1 |
|=--+------=|
|age|name   |
|=--+------=|
|27 |Tunisia|
|31 |Senegal|
|26 |Morocco|
|33 |Senegal|
|27 |Japan  |
|26 |Algeria|
|33 |Japan  |
|28 |Lebanon|
|27 |Tunisia|
|31 |Senegal|
|26 |Morocco|
|33 |Senegal|
|27 |Japan  |
|26 |Algeria|
|33 |Japan  |
|28 |Lebanon|
|27 |Tunisia|
|31 |Senegal|
|26 |Morocco|
|33 |Senegal|
|27 |Japan  |
|26 |Algeria|
|33 |Japan  |
|28 |Lebanon|
|27 |Tunisia|
|31 |Senegal|
|26 |Morocco|
|33 |Senegal|
|27 |Japan  |
|26 |Algeria|
|33 |Japan  |
|28 |Lebanon|
|27 |Tunisia|
|31 |Senegal|
|26 |Morocco|
|33 |Senegal|
|27 |Japan  |
|26 |Algeria|
```

The data of the restricted table read from the HDFS is displayed onto the console.

Type in *http://talend-hdp:50075/browseDirectory.jsp?dir=/user/hdp/Customer&namenodeInfoPort=50070* to the address bar of your browser to view the table you created:



Contents of directory /user/hdp/Customer

Goto : /user/hdp/Customer go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
Customer.csv	file	1.04 KB	3	64 MB	2012-04-18 16:16	rw-r--r--	hdp	hdfs

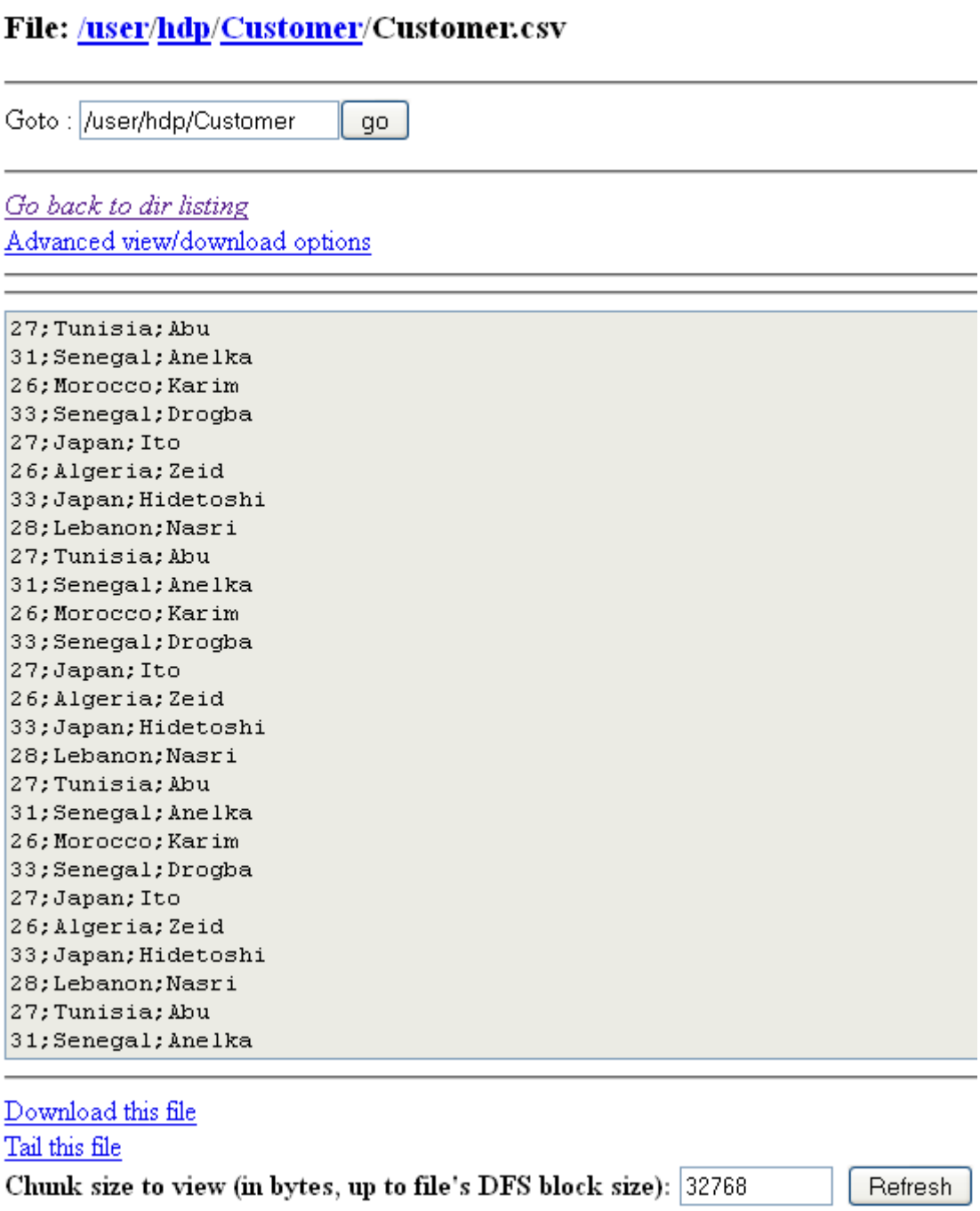
Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.2

Click the *Customer.csv* link to view the content of the table you created.


File: /user/hdp/Customer/Customer.csv
Goto : /user/hdp/Customer
go
Go back to dir listing
Advanced view/download options
27;Tunisia;Abu
31;Senegal;Anelka
26;Morocco;Karim
33;Senegal;Drogba
27;Japan;Ito
26;Algeria;Zeid
33;Japan;Hidetoshi
28;Lebanon;Nasri
27;Tunisia;Abu
31;Senegal;Anelka
26;Morocco;Karim
33;Senegal;Drogba
27;Japan;Ito
26;Algeria;Zeid
33;Japan;Hidetoshi
28;Lebanon;Nasri
27;Tunisia;Abu
31;Senegal;Anelka
26;Morocco;Karim
33;Senegal;Drogba
27;Japan;Ito
26;Algeria;Zeid
33;Japan;Hidetoshi
28;Lebanon;Nasri
27;Tunisia;Abu
31;Senegal;Anelka
Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768
Refresh

tHCatalogOutput

tHCatalogOutput Properties

Component family	Big Data / HCatalog	
Function	This component allows you to write data into an HCatalog managed table using Talend data flow.	
Purpose	The **tHCatalogOutput** component writes data into a HCatalog managed table using Talend data flow.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	HCatalog version	Select the version of the Hadoop distribution you are using.
	NameNode URI	Type in the URI of the NameNode in HDFS.
	File name	Browse to, or enter the location of the file which you write data to. This file is created automatically if it does not exist.
	Action	Select a DB operation in HDFS: **Create**: Creates a file with data using the file name defined in the **File Name** field. **Overwrite**: Overwrites the data in the file specified in the **File Name** field. **Append**: Inserts the data into the file specified in the **File Name** field. The specified file is created automatically if it does not exist.
	Templeton hostname	Fill this field with the URL of Templeton Webservice. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Templeton port	Fill this field with the port of URL of Templeton Webservice. By default, this value is *50111*. Templeton is a webservice API for Hadoop. It allows you to move data directly into/out-of HDFS through WebHDFS. For further information about Templeton, see http://people.apache.org/~thejas/templeton_doc_latest.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed.

		This check box is available depending on the Hadoop distribution you are connecting to.
	Database	Fill this field to specify an existing database in HDFS.
	Table	Fill this field to specify an existing table in HDFS.
	Partition	Fill this field to specify one or more partitions for the partition operation on the specified table. When you specify multiple partitions, use commas to separate every two partitions and use double quotation marks to quote the partition string. For further information about **Partition**, see https://cwiki.apache.org/Hive/languagemanual-ddl.html#LanguageManualDDL-AddPartitions.
	Username	Fill this field with the username for the DB authentication.
	File location	Fill this field with the path to which source data file is stored.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Row separator*	Enter the separator used to identify the end of a row.
	Field separator	Enter the separator used to identify the end of a field.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Retrieve the HCatalog logs	Select this check box to retrieve log files generated during HCatalog operations.
	Standard Output Folder	Browse to, or enter the directory where the log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	Error Output Folder	Browse to, or enter the directory where the error log files are stored. This field is enabled only when you selected **Retrieve the HCatalog logs** check box.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used together with an input component.	
Limitation	Knowledge of Hive Data Definition Language and HCatalog Data Definition Language is required. For further information about Hive Data Definition Language, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL. For further information about HCatalog Data Definition Language, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.	

Related scenario

For a related scenario, see section *Scenario: HCatalog table management on Hortonworks Data Platform*.

tHDFSCompare

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSCompare properties

Component family	Big Data/File	
Function	This component compares two files in HDFS and based on the read-only schema, generates a row flow that presents the comparison information.	
Purpose	This component helps to control the quality of the data processed.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. . If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. The schema of this component is read-only. You can click **Edit schema** to view the schema.
	Comparison mode	Select the mode to be applied on the comparison.
	File to compare	Browse, or enter the path to the file in HDFS you need to check for quality control.
	Reference file	Browse, or enter the path to the file in HDFS the comparison is based on.
	If differences detected, display and *If no differences detected, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the message in the **Run** console.
Advanced settings	*Encoding*	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Hadoop properties	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to

		be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tHDFSCompare** can be standalone component or send the information it generates to its following component.	
Limitation	JRE 1.6+ is required.	

Related scenario

No scenario is available for this component yet.

tHDFSConnection

tHDFSConnection properties

Component family	Big Data / Hadoop	
Function	**tHDFSConnection** provides connection to the Hadoop distributed file system (HDFS) of interest at runtime.	
Purpose	**tHDFSConnection** connects to a given HDFS so that the other Hadoop components can reuse the connection it creates to communicate with this HDFS.	
Basic settings	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
	NameNode URI	Type in the URI of the Hadoop NameNode.
	User name	User authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Hadoop properties	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is generally used with other Hadoop components.	
Limitations	JRE 1.6+ is required.	

Related scenario

No scenario is available for this component yet.

tHDFSCopy

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSCopy properties

Component family	Big Data/File	
Function	**tHDFSCopy** copies a source file or folder into a target directory in HDFS and removes this source if required.	
Purpose	**tHDFSCopy** moves files in HDFS with or without keeping the source.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Source file or directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	Target location	Browse to, or enter the directory in HDFS to which you need to copy the data.
	Rename	To rename the file or folder copied to the target location, select this check box to display the **New name** field, then, enter the new name.
	Remove source	Select this check box to remove the source file or folder once this source is copied to the target location.
	Override target file (This option does not override the directory)	Select this check box to override the file already existing in the target location. This option does not override the folder.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.

Usage	**tHDFSCopy** is a standalone component.
Limitation	JRE 1.6+ is required.

Related scenario

Related topic, see section *Scenario: Restoring files from bin*

Related topic, see section *Scenario: Iterating on a HDFS directory*

tHDFSDelete

tHDFSDelete properties

Component family	Big Data / Hadoop	
Function	**tHDFSDelete** deletes a file located on a given Hadoop distributed file system (HDFS).	
Purpose	**tHDFSDelete** deletes a file on a given HDFS.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	User authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	File or Directory Path	Browse to, or enter the path to the file or folder to be deleted on HDFS.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used to compose a single-component Job or Subjob.	
Limitations	JRE 1.6+ is required.	

Related scenario

No scenario is available for this component yet.

tHDFSExist

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSExist properties

Component family	Big Data/File	
Function	This component checks whether a file exists in a specific directory in HDFS.	
Purpose	This component checks the existence of a specific file in HDFS.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	HDFS directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	File name or relative path	Enter the name of the file you want to check whether this file exists. Or if needs be, browse to the file or enter the path to the file, relative to the directory you entered in **HDFS directory**.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.

	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tHDFSExist** is a standalone component.	
Limitation	JRE 1.6+ is required.	

Scenario: Checking the existence of a file in HDFS

In this scenario, the two-component Job checks whether a specific file exists in HDFS and returns a message to indicate the result of the verification.

In the real-world practice, you can take further action to process the file checked according to the verification result, using the other HDFS components provided with the Studio.

Launch the Hadoop distribution in which you want to check the existence of a particular file. Then, proceed as follows:

Linking the components

1. In the **Integration** perspective of the Studio, create an empty Job, named *hdfsexist_file* for example, from the **Job Designs** node in the **Repository** tree view.

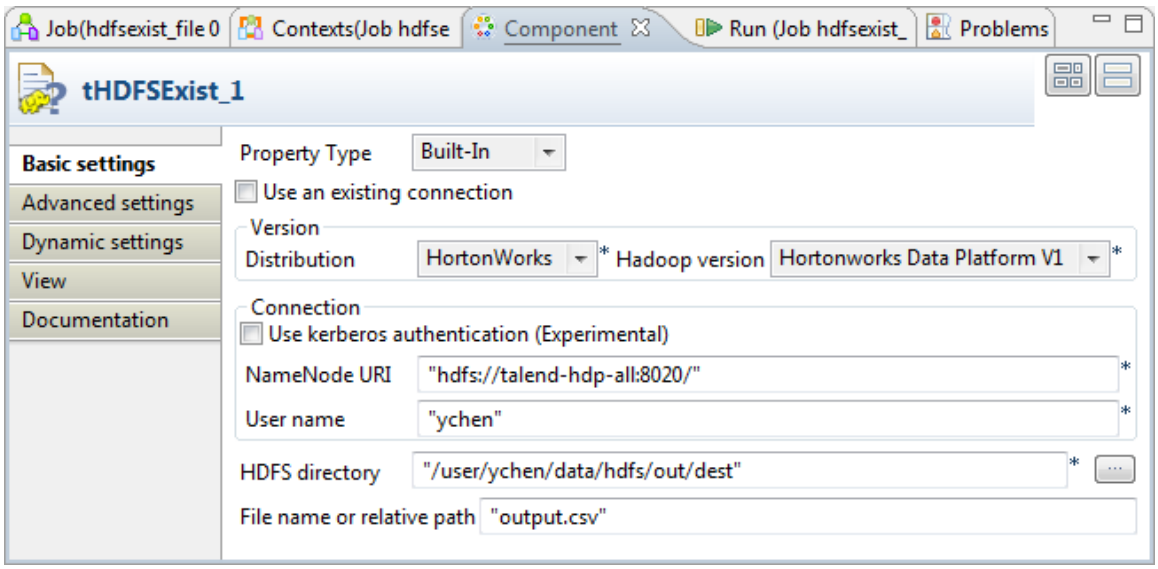
   For further information about how to create a Job, see the user guide of the Studio for integration.

2. Drop **tHDFSExist** and **tMsgBox** onto the workspace.

3. Connect them using the **Trigger** > **Run if** link.

Configuring the connection to HDFS

1. Double-click **tHDFSExist** to open its **Component** view.



2. In the **Version** area, select the Hadoop distribution you are connecting to and its version.

3. In the **Connection** area, enter the values of the parameters required to connect to the HDFS.

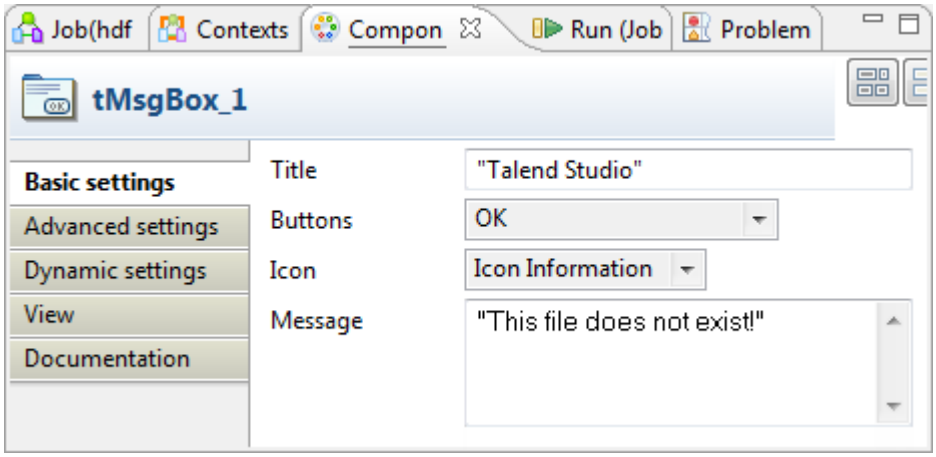
   In the real-world practice, you may use **tHDFSConnection** to create a connection and reuse it from the current component. For further information, see section *tHDFSConnection*.

4. In the **HDFS Directory** field, browse to, or enter the path to the folder where the file to be checked is. In this example, browse to */user/ychen/data/hdfs/out/dest*.

5. In the **File name or relative path** field, enter the name of the file you want to check the existence. For example, **output.csv**.

Defining the message to be returned

1. Double-click **tMsgBox** to open its **Component** view.



2. In the **Title** field, enter the title to be used for the pop-up message box to be created.

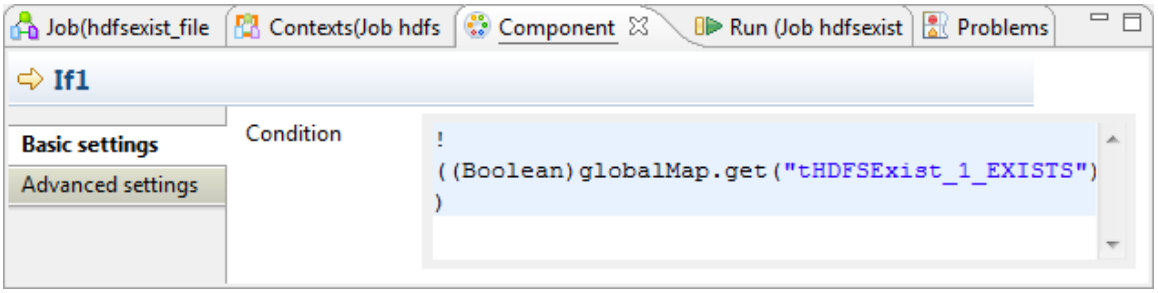
3. In the **Buttons** list, select *OK*. This defines the button to be displayed on the message box.

4. In the **Icon** list, select *Icon information*.

5. In the **Message** field, enter the message you want to displayed once the file checking is done. In this example, enter *"This file does not exist!"*.

Defining the condition

1. Click the **If** link to open the **Basic settings** view, where you are able to define the condition for checking the existence of this file.

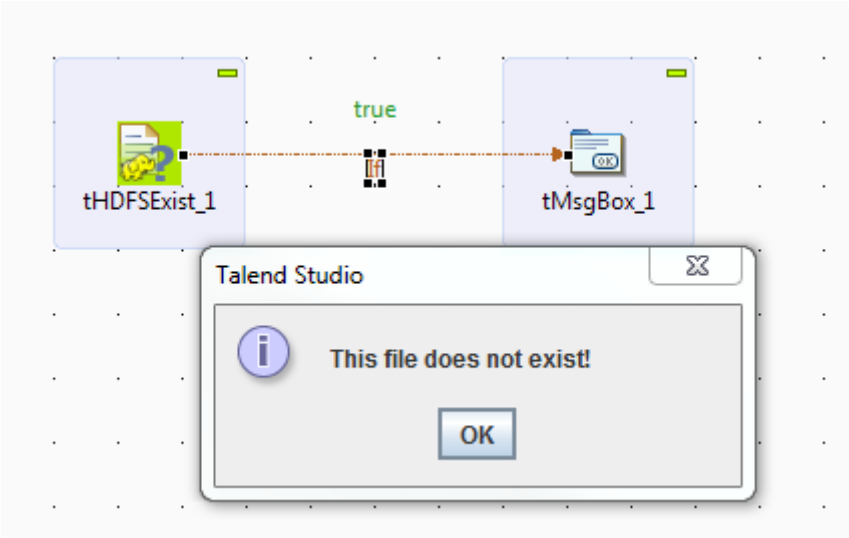


2. In the **Condition** box, press **Ctrl+Space** to access the variable list and select the global variable **EXISTS**. Type an exclamation mark before the variable to negate the meaning of the variable.

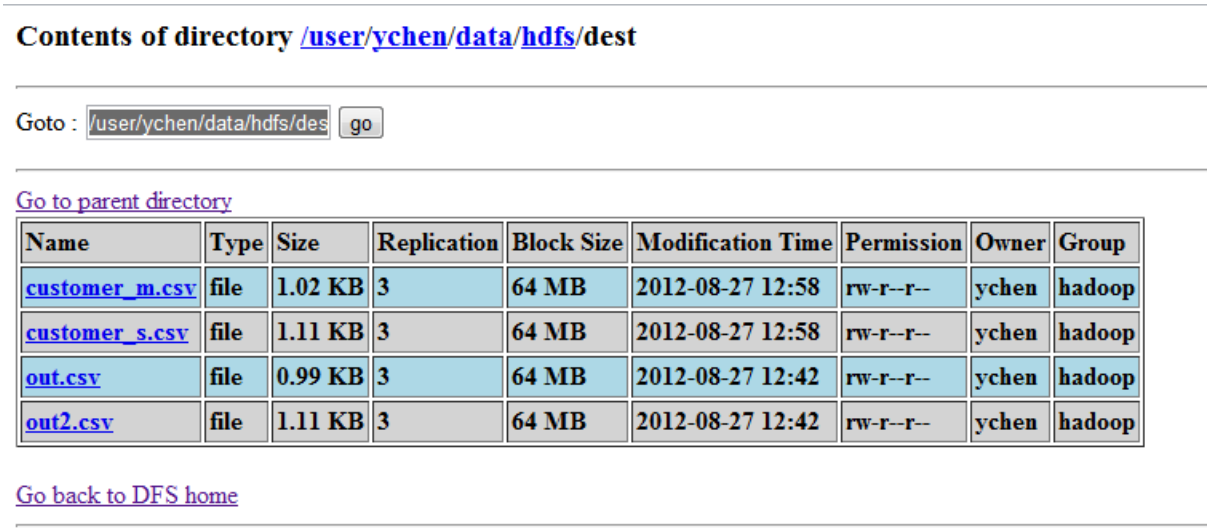
Executing the Job

- Press **F6** to execute this Job.

Once done, a message box pops up to indicate that this file called *output.csv* does not exist in the directory you defined earlier.



In the HDFS we check the existence of the file, browse to this directory specified, you can see that this file does not exist.



Contents of directory /user/ychen/data/hdfs/dest

Goto : /user/ychen/data/hdfs/des go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
customer_m.csv	file	1.02 KB	3	64 MB	2012-08-27 12:58	rw-r--r--	ychen	hadoop
customer_s.csv	file	1.11 KB	3	64 MB	2012-08-27 12:58	rw-r--r--	ychen	hadoop
out.csv	file	0.99 KB	3	64 MB	2012-08-27 12:42	rw-r--r--	ychen	hadoop
out2.csv	file	1.11 KB	3	64 MB	2012-08-27 12:42	rw-r--r--	ychen	hadoop

Go back to DFS home

tHDFSGet

tHDFSGet properties

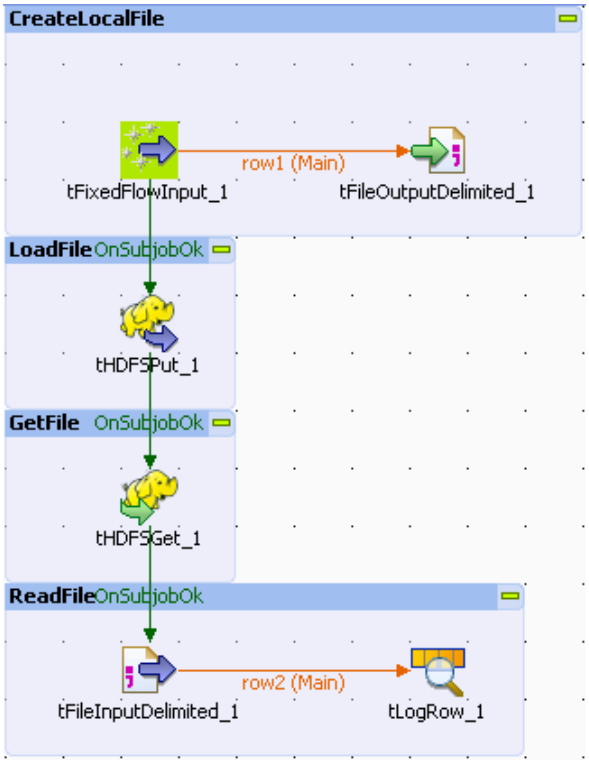
Component family	Big Data / Hadoop	
Function	**tHDFSGet** gets data from Hadoop distributed file system(HDFS).	
Purpose	**tHDFSGet** connects to Hadoop distributed file system, helping to obtain large-scale files with optimized performance.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	HDFS directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	Local directory	Browse to, or enter the local directory to store the files obtained from HDFS.
	Overwrite file	Options to overwrite or not the existing file with the new one.
	Append	Select this check box to add the new rows at the end of the records
	Files	In the **Files** area, the fields to be completed are: - **File mask**: type in the file name to be selected from HDFS. Regular expression is available. - **New name**: give a new name to the obtained file.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component combines HDFS connection and data extraction, thus used as a single-component subjob or to prepare data to feed the input flow of a Job. It is often connected to the Job using **OnSubJobOk** or **OnComponentOk link**, depending on the context.	
Limitations	JRE 1.6+ is required.	

Scenario: Computing data with Hadoop distributed file system

The following scenario describes a simple Job that creates a file in a defined directory, get it into and out of HDFS, subsequently store it to another local directory, and read it at the end of the Job.

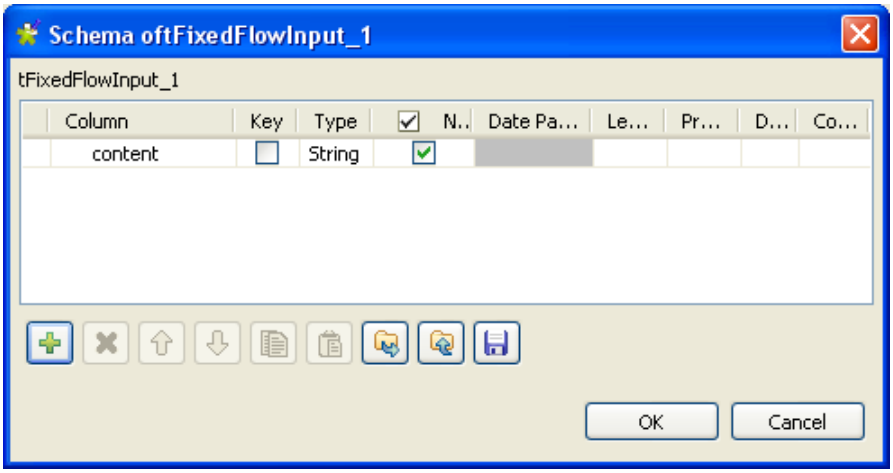
Setting up the Job

1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput**, **tFileOutputDelimited**, **tHDFSPut**, **tHDFSGet**, **tFileInputDelimited** and **tLogRow**.
2. Connect **tFixedFlowInput** to **tFileOutputDelimited** using a **Row** > **Main** connection.
3. Connect **tFileInputDelimited** to **tLogRow** using a **Row** > **Main** connection.
4. Connect **tFixedFlowInput** to **tHDFSPut** using an **OnSubjobOk** connection.
5. Connect **tHDFSPut** to **tHDFSGet** using an **OnSubjobOk** connection.
6. Connect **tHDFSGet** to **tFileInputDelimited**using an **OnSubjobOk** connection.

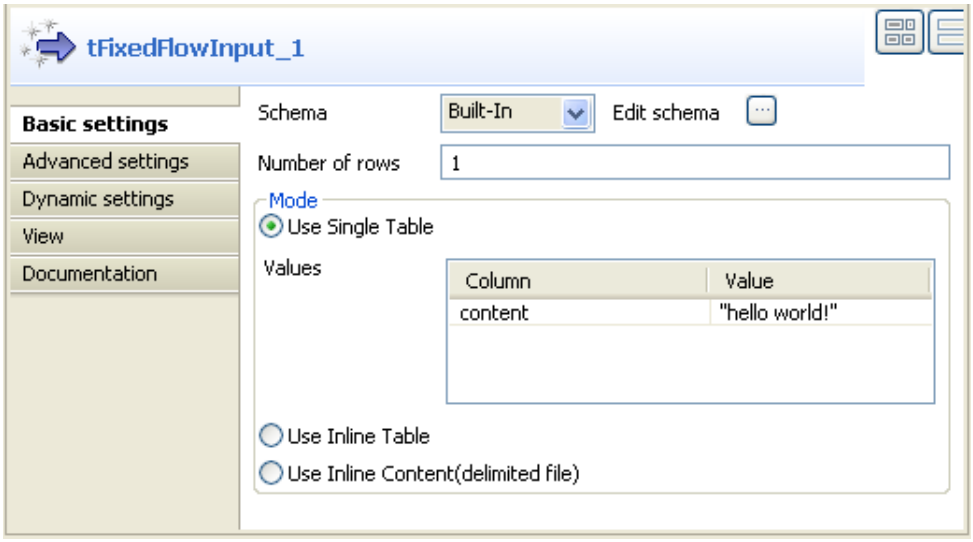


Configuring the input component

1. Double-click **tFixedFlowInput** to define the component in its **Basic settings** view.
2. Set the **Schema** to **Built-In** and click the three-dot **[...]** button next to **Edit Schema** to describe the data structure you want to create from internal variables. In this scenario, the schema contains one column: *content*.

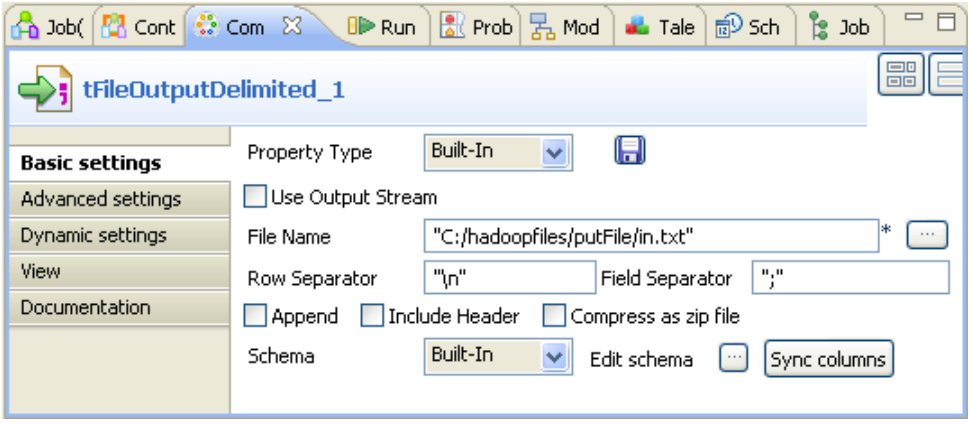


3. Click the plus button to add the parameter line.
4. Click **OK** to close the dialog box and accept to propagate the changes when prompted by the studio.
5. In **Basic settings**, define the corresponding value in the **Mode** area using the **Use Single Table** option. In this scenario, the value is *"Hello world!"*.



Configuring the tFileOutputDelimited component

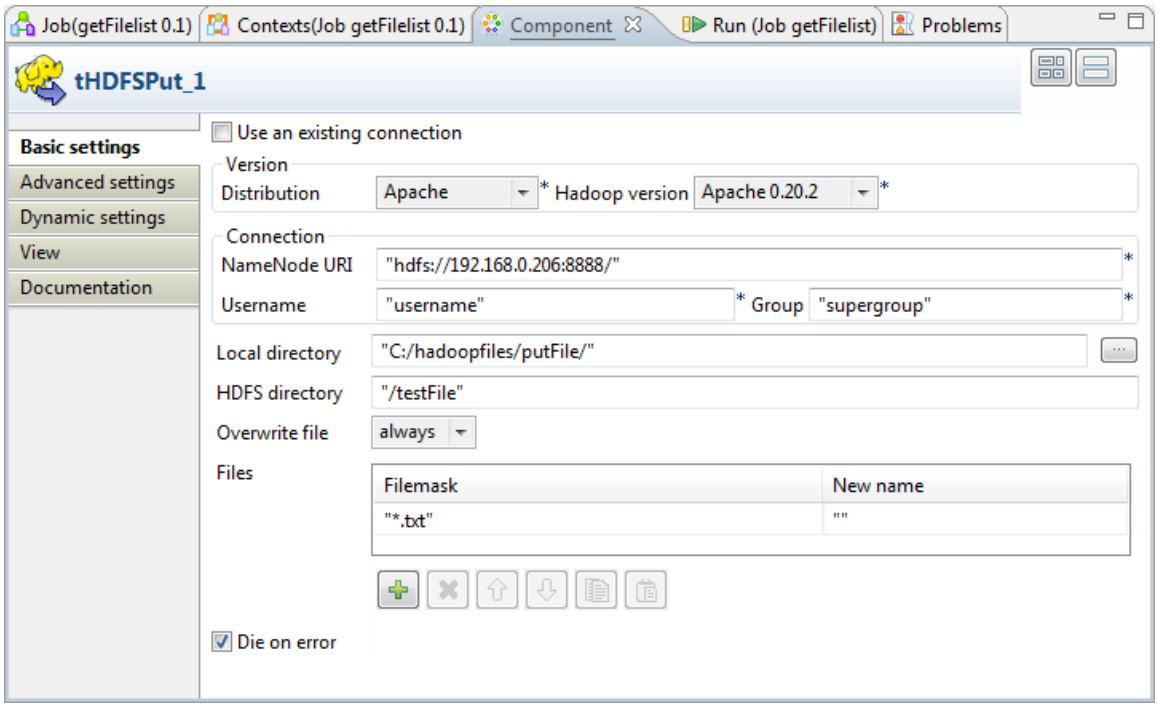
1. Double-click **tFileOutputDelimited** to define the component in its **Basic settings** view.



2. Click the **[...]** button next to the **File Name** field and browse to the output file you want to write data in, *in.txt* in this example.

Loading the data from the local file

1. Double-click **tHDFSPut** to define the component in its **Basic settings** view.



2. Select, for example, Apache 0.20.2 from the **Hadoop version** list.

3. In the **NameNode URI**, the **Username** and the **Group** fields, enter the connection parameters to the HDFS.

4. Next to the **Local directory** field, click the three-dot **[...]** button to browse to the folder with the file to be loaded into the HDFS. In this scenario, the directory has been specified while configuring **tFileOutputDelimited**: *C:/hadoopfiles/putFile/*.

5. In the **HDFS directory** field, type in the intended location in HDFS to store the file to be loaded. In this example, it is */testFile*.

6. Click the **Overwrite file** field to stretch the drop-down.

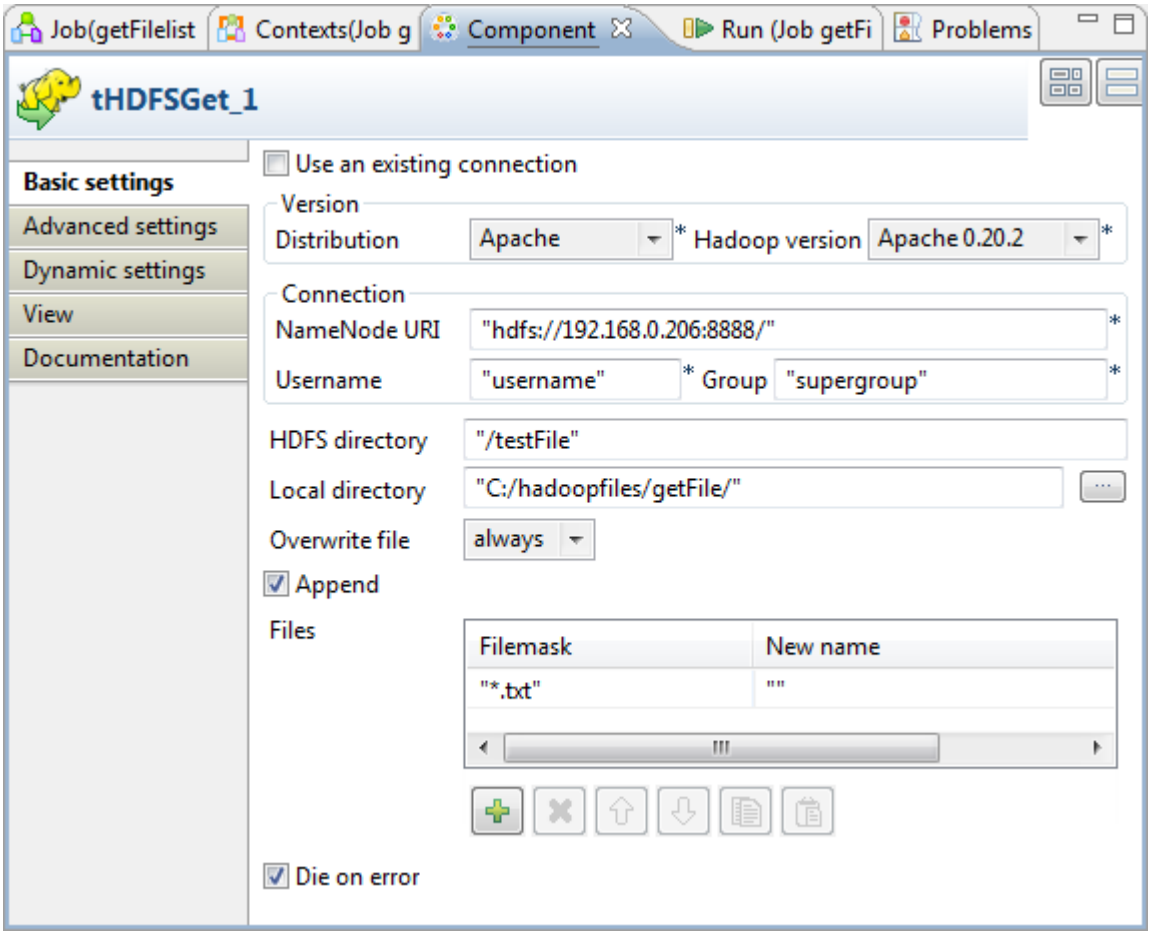
7. From the menu, select **always**.

8. In the **Files** area, click the plus button to add a row in which you define the file to be loaded.

9. In the **File mask** column, enter **.txt* to replace *newLine* between quotation marks and leave the **New name** column as it is. This allows you to extract all the *.txt* files in the specified directory without changing their names. In this example, the file is *in.txt*.

Getting the data from the HDFS

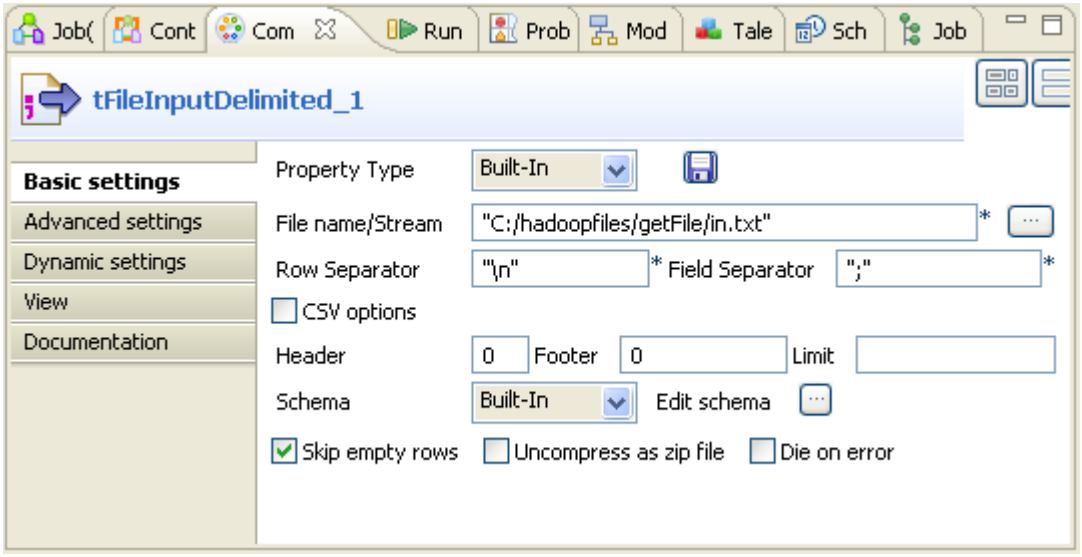
1. Double-click **tHDFSGet** to define the component in its **Basic settings** view.



2. Select, for example, Apache 0.20.2 from the **Hadoop version** list.
3. In the **NameNode URI**, the **Username**, the **Group** fields, enter the connection parameters to the HDFS.
4. In the **HDFS directory** field, type in location storing the loaded file in HDFS. In this example, it is */testFile*.
5. Next to the **Local directory** field, click the three-dot **[...]** button to browse to the folder intended to store the files that are extracted out of the HDFS. In this scenario, the directory is: *C:/hadoopfiles/getFile/*.
6. Click the **Overwrite file** field to stretch the drop-down.
7. From the menu, select **always**.
8. In the **Files** area, click the plus button to add a row in which you define the file to be extracted.
9. In the **File mask** column, enter **.txt* to replace *newLine* between quotation marks and leave the **New name** column as it is. This allows you to extract all the *.txt* files from the specified directory in the HDFS without changing their names. In this example, the file is *in.txt*.

Reading data from the HDFS and saving the data locally

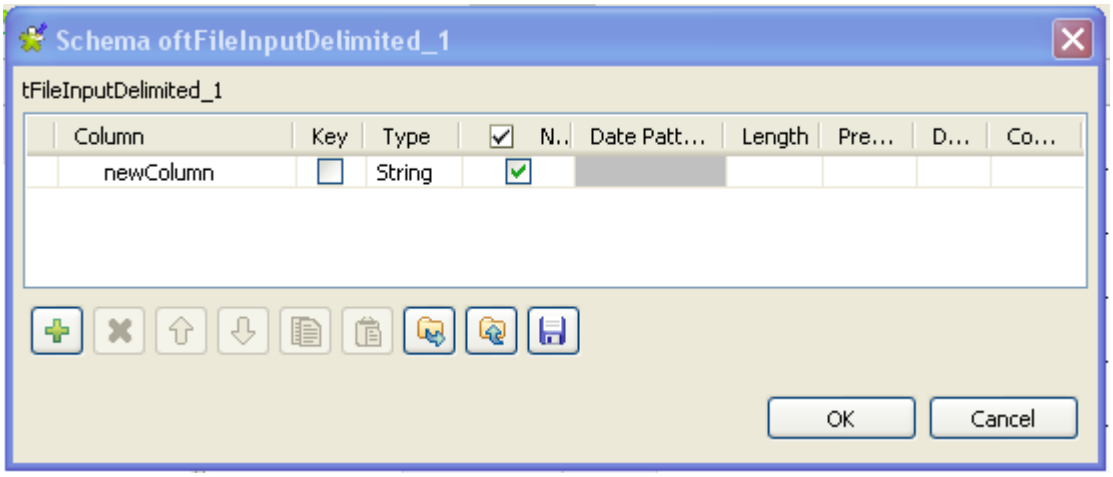
1. Double-click **tFileInputDelimited** to define the component in its **Basic settings** view.



2. Set property type to **Built-In**.

3. Next to the **File Name/Stream** field, click the three-dot button to browse to the file you have obtained from the HDFS. In this scenario, the directory is *C:/hadoopfiles/getFile/in.txt*.

4. Set **Schema** to **Built-In** and click **Edit schema** to define the data to pass on to the **tLogRow** component.



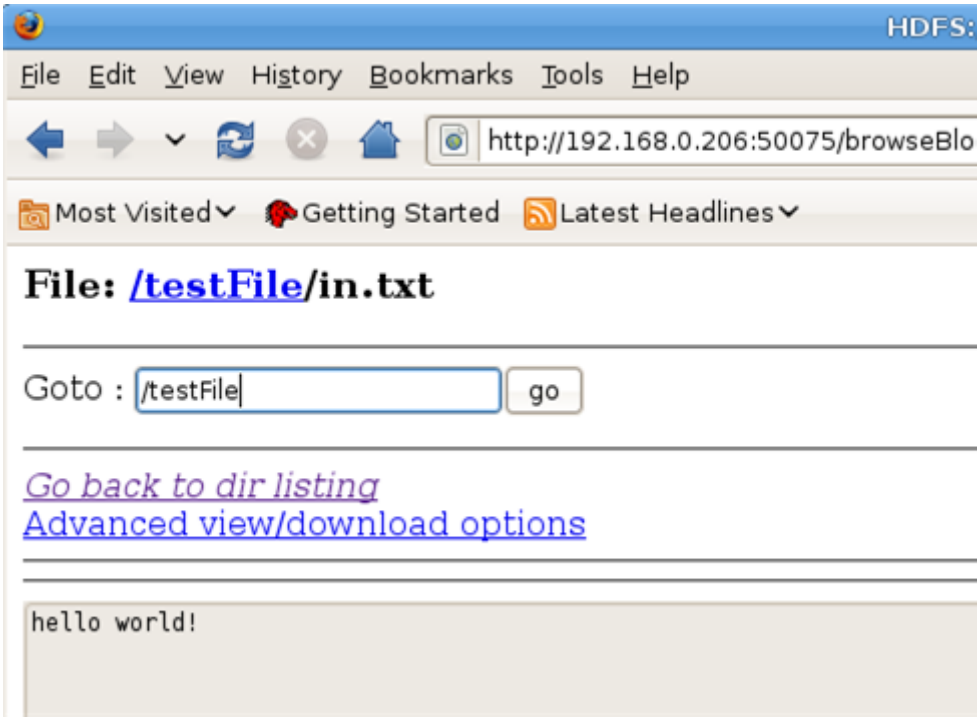
5. Click the plus button to add a new column.

6. Click **OK** to close the dialog box and accept to propagate the changes when prompted by the studio.

Executing the Job

Save the Job and press **F6** to execute it.

The *in.txt* file is created and loaded into the HDFS.



The file is also extracted from the HDFS by **tHDFSGet** and is read by **tFileInputDelimited**.

```
Starting job feature11717_hadoop_get_put at 18:09 04/05/2010.

[statistics] connecting to socket on port 3637
[statistics] connected
hello world!
[statistics] disconnected
Job feature11717_hadoop_get_put ended at 18:09 04/05/2010.
```

tHDFSInput

tHDFSInput properties

Component family	Big Data / Hadoop	
Function	**tHDFSInput** reads a file located on a given Hadoop distributed file system (HDFS) and puts the data of interest from this file into a Talend schema. Then it passes the data to the component that follows.	
Purpose	**tHDFSInput** extracts the data in a HDFS file for other components to process it.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use an existing connection	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	File Name	Browse to, or enter the directory in HDFS where the data you need to use is.
	Row separator	Enter the separator used to identify the end of a row.
	Field separator	Enter the separator used to identify the end of a field.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop.

		For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component needs an output link.	
Limitations	JRE 1.6+ is required.	

Related scenario

- Related topic, see section *Scenario 1: Writing data in a delimited file*.
- Related topic, see section *Scenario: Computing data with Hadoop distributed file system*.

tHDFSList

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSList properties

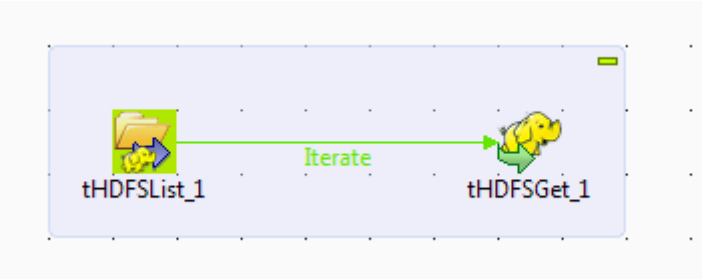
Component family	Big Data/File	
Function	**tHDFSList** iterates on files or folders of a set directory.	
Purpose	**tHDFSList** retrieves a list of files or folders based on a filemask pattern and iterates on each unity.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	HDFS Directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	FileList Type	Select the type of input you want to iterate on from the list: **Files** if the input is a set of files, **Directories** if the input is a set of directories, **Both** if the input is a set of the above two types.
	Include subdirectories	Select this check box if the selected input source type includes sub-directories.
	Case Sensitive	Set the case mode from the list to either create or not create case sensitive filter on filenames.
	Use Glob Expressions as Filemask	This check box is selected by default. It filters the results using a Global Expression (**Glob Expressions**).
	Files	Click the plus button to add as many filter lines as needed: **Filemask**: in the added filter lines, type in a filename or a filemask using special characters or regular expressions.
	Order by	The folders are listed first of all, then the files. You can choose to prioritise the folder and file order either: **By default**: alphabetical order, by folder then file;

		By file name: alphabetical order or reverse alphabetical order; **By file size**: smallest to largest or largest to smallest; **By modified date**: most recent to least recent or least recent to most recent. If ordering **by file name**, in the event of identical file names then **modified date** takes precedence. If ordering **by file size**, in the event of identical file sizes then **file name** takes precedence. If ordering by **modified date**, in the event of identical dates then **file name** takes precedence.
	Order action	Select a sort order by clicking one of the following radio buttons: **ASC**: ascending order; **DESC**: descending order;
Advanced settings	*Use Exclude Filemask*	Select this check box to enable **Exclude Filemask** field to exclude filtering condition based on file type: **Exclude Filemask**: Fill in the field with file types to be excluded from the **Filemasks** in the **Basic settings** view. File types in this field should be quoted with double quotation marks and seperated by comma.
	Hadoop properties	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tHDFSList** provides a list of files or folders from a defined HDFS directory on which it iterates	
Global Variables		**Current File Name**: Indicates the current file name. This is available as a **Flow** variable. Returns a string. **Current File Name with Path**: Indicates the current file name as well as the path to the file. This is available as a **Flow** variable. Returns a string. **Current File Extension:** Indicates the extension of the current file. This is available as a **Flow** variable. Returns a string. **Current File Directory**: Indicates the access path to the folder or subfolder in which the current file is stored. This is available as a **Flow** variable. Returns a string. **Number of files**: Indicates the number of files iterated upon so far. This is available as a **Flow** variable. Returns an integer.

		For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	JRE 1.6+ is required.	

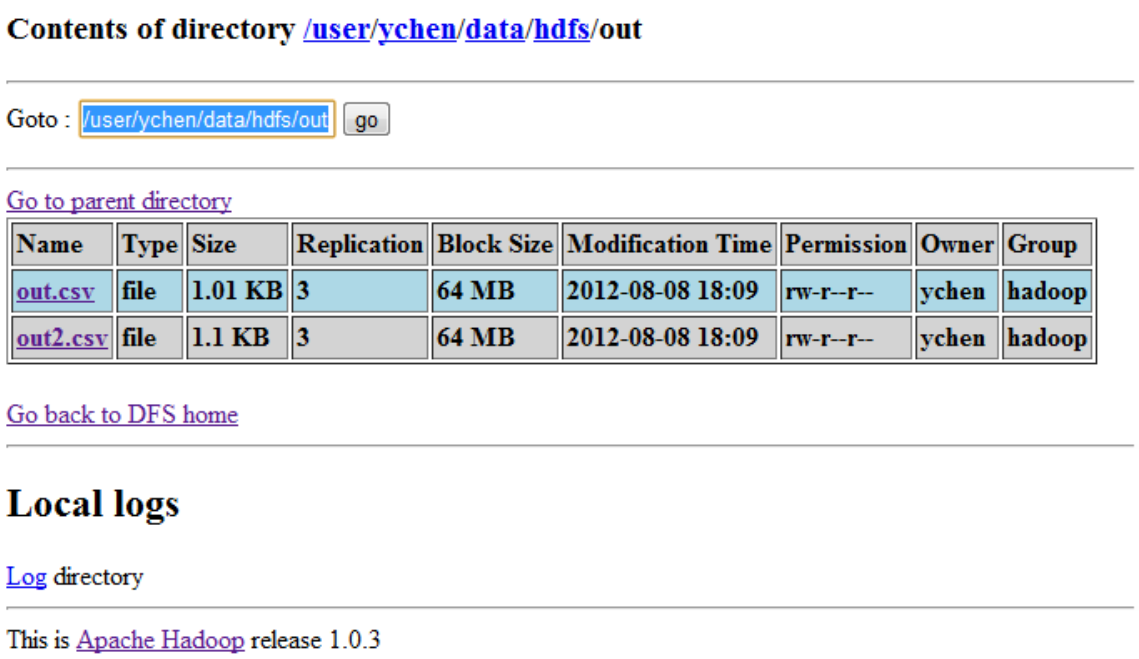
Scenario: Iterating on a HDFS directory

This scenario uses a two-component Job to iterate on a specified directory in HDFS so as to select the files from there towards a local directory.



Preparing the data to be used

- Create the files to be iterated on in the HDFS you want to use. In this scenario, two files are created in the directory: */user/ychen/data/hdfs/out*.



Contents of directory /user/ychen/data/hdfs/out

Goto : /user/ychen/data/hdfs/out go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
out.csv	file	1.01 KB	3	64 MB	2012-08-08 18:09	rw-r--r--	ychen	hadoop
out2.csv	file	1.1 KB	3	64 MB	2012-08-08 18:09	rw-r--r--	ychen	hadoop

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.3

You can design a Job in the Studio to create the two files. For further information, see section *tHDFSPut* or section *tHDFSOutput*.

Linking the components

1. In the **Integration** perspective of the Studio, create an empty Job, named *HDFSList* for example, from the **Job Designs** node in the **Repository** tree view.

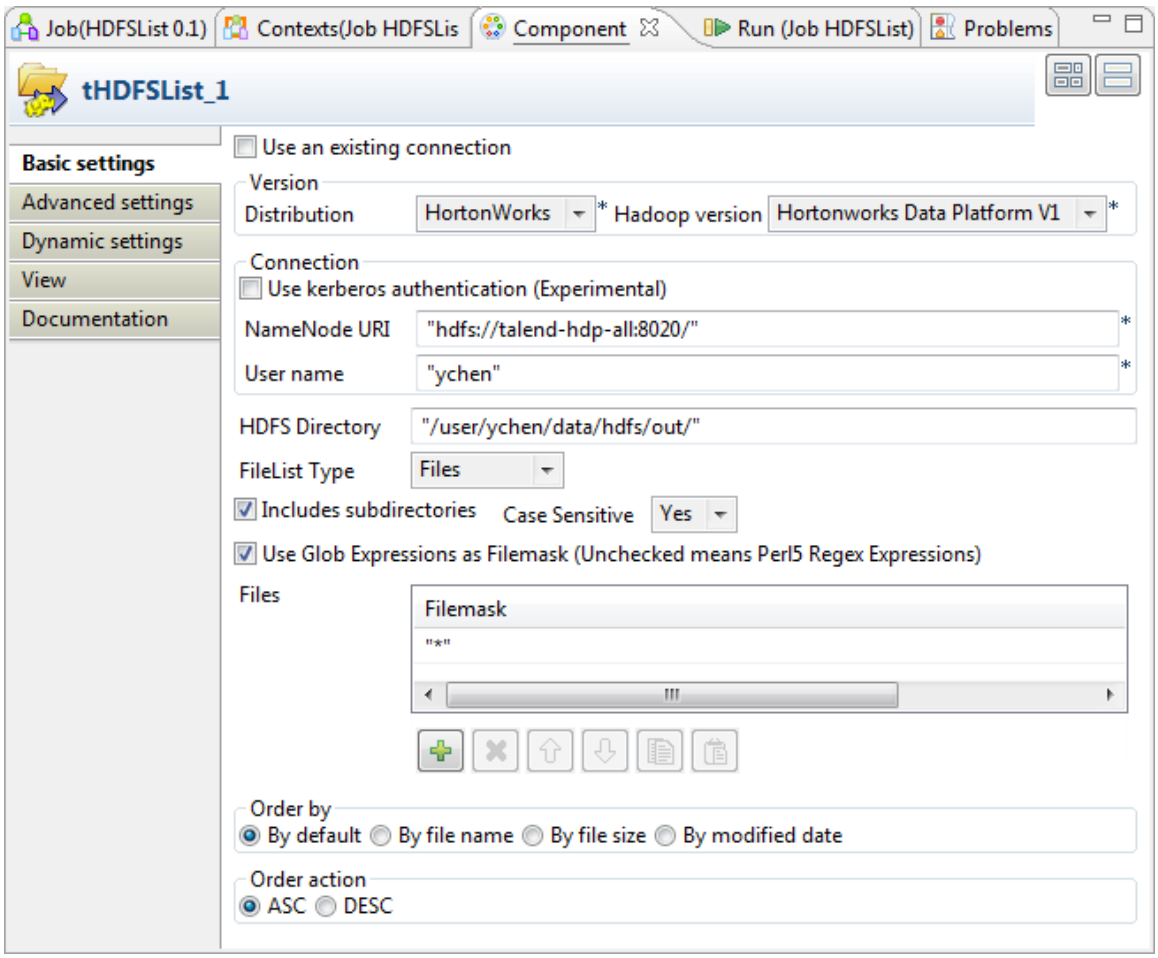
   For further information about how to create a Job, see the user guide of the Studio for integration.

2. Drop **tHDFSList** and **tHDFSGet** onto the workspace.

3. Connect them using the **Row > Iterate** link.

Configuring the iteration

1. Double-click **tHDFSList** to open its **Component** view.



2. In the **Version** area, select the Hadoop distribution you are connecting to and its version.

3. In the **Connection** area, enter the values of the parameters required to connect to the HDFS.

   In the real-world practice, you may use **tHDFSConnection** to create a connection and reuse it from the current component. For further information, see section *tHDFSConnection*.

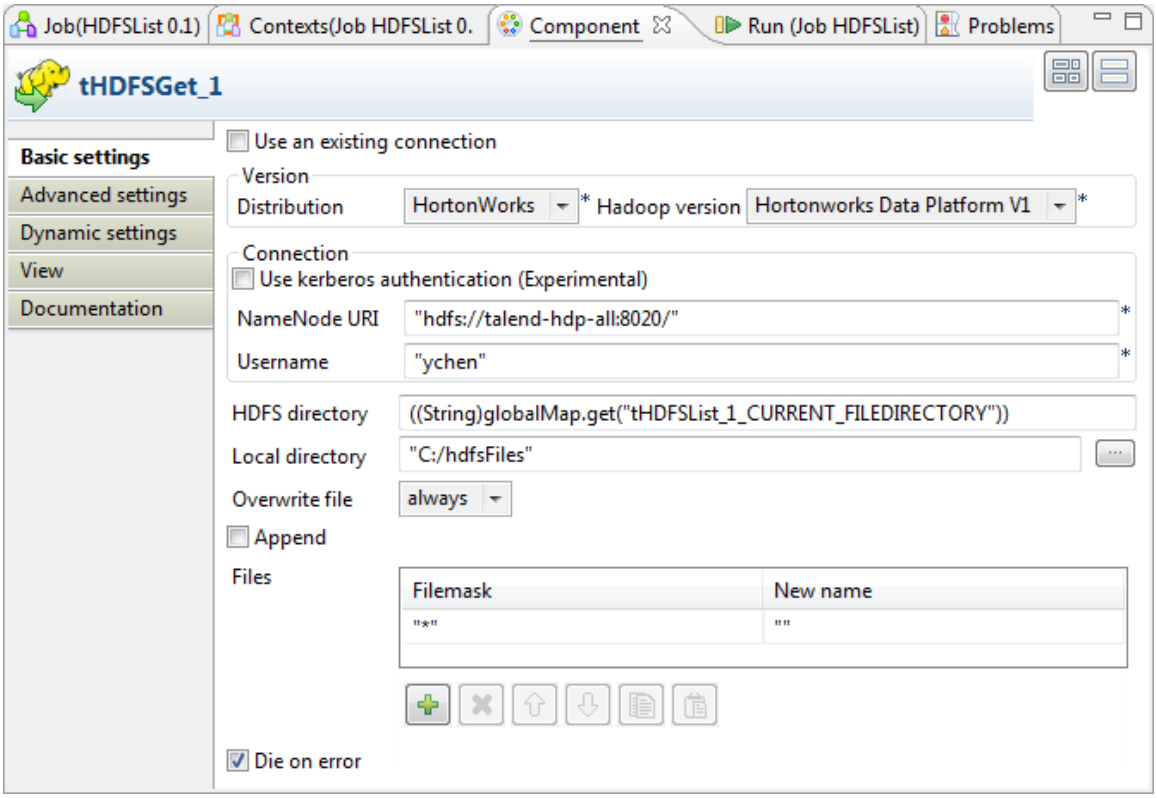
4. In the **HDFS Directory** field, enter the path to the folder where the files to be iterated on are. In this example, as presented earlier, the directory is */user/ychen/data/hdfs/out/*.

5. In the **FileList Type** field, select **File**.

6. In the **Files** table, click to add one row and enter * between the quotation marks to iterate on any files existing.

Selecting the files

1. Double-click **tHDFSGet** to open its **Component** view.



2. In the **Version** area, select the Hadoop distribution you are connecting to and its version.
3. In the **Connection** area, enter the values of the parameters required to connect to the HDFS.

   In the real-world practice, you may have used **tHDFSConnection** to create a connection; then you can reuse it from the current component. For further information, see section *tHDFSConnection*.
4. In the **HDFS directory** field, enter the path to the folder holding the files to be retrieved.

   You can put the mouse pointer in this field, then, press **Ctrl+Space** to display the auto-completion list and select the *tHDFSList_1_CURRENT_FILEDIRECTORY* variable to reuse the directory you have defined in **tHDFSList**. In this variable, *tHDFSList_1* is the label of the component. If you label it differently, select the variable accordingly.

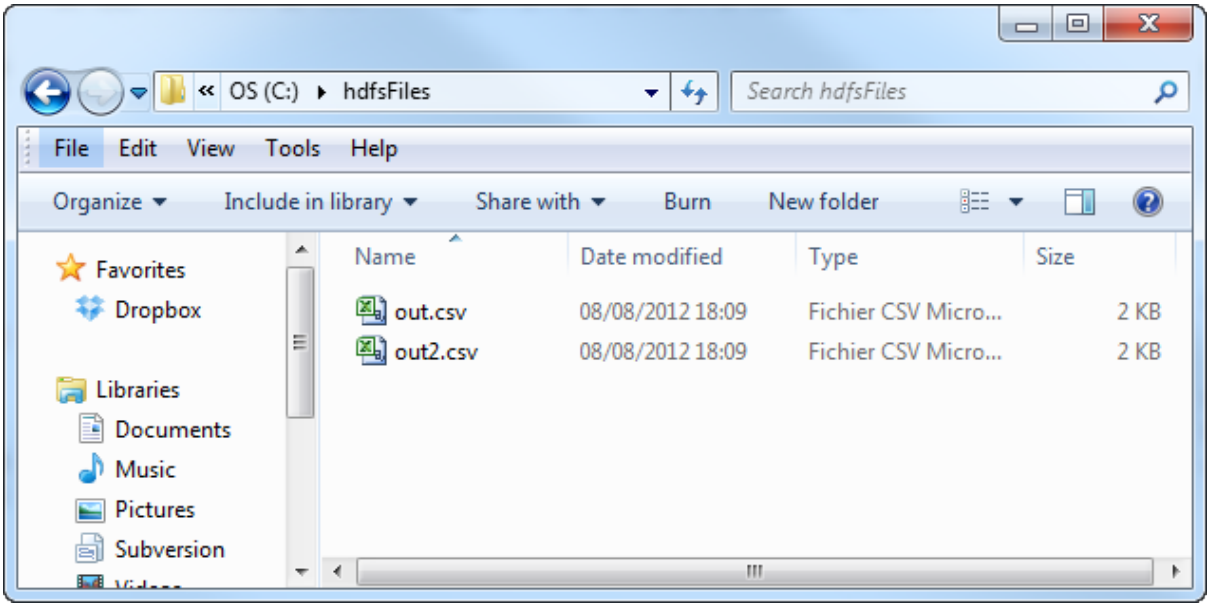
   Once selecting this variable, the directory reads, for example, *((String)globalMap.get("tHDFSList_1_CURRENT_FILEDIRECTORY"))* in this field.

   For further information about how to label a component, see the Studio user guide for integration.
5. In the **Local directory** field, enter the path, or browse to the folder you want to place the selected files in. This folder will be created if it does not exist. In this example, it is *C:/hdfsFiles*.
6. In the **Overwrite file** field, select **always**.
7. In the **Files** table, click to add one row and enter * between the quotation marks in the **Filemask** column in order to get any files existing.

Executing the Job

- Press **F6** to execute this Job.

Once done, you can check the files created in the local directory.



tHDFSOutput

tHDFSOutput properties

Component family	Big Data / Hadoop	
Function	**tHDFSOutput** writes data flows it receives onto a given Hadoop distributed file system (HDFS).	
Purpose	**tHDFSOnput** writes data of interest into a file and creates this file on a given HDFS.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. . Click **Edit Schema** to make changes to the schema. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use an existing connection	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
	NameNode URI	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	File Name	Browse to, or enter the location of the file which you write data to. This file is created automatically if it does not exist.
	Action	Select an operation in HDFS: **Create**: Creates a file with data using the file name defined in the **File Name** field. **Overwrite**: Overwrites the data in the file specified in the **File Name** field. **Append**: Inserts the data into the file specified in the **File Name** field. The specified file is created automatically if it does not exist.
	Row separator	Enter the separator used to identify the end of a row.
	Field separator	Enter the separator used to identify the end of a field.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component needs an input component.	
Limitations	JRE 1.6+ is required.	

Related scenario

- Related topic, see section *Scenario 1: Writing data in a delimited file*.
- Related topic, see section *Scenario: Computing data with Hadoop distributed file system*.

tHDFSProperties

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSProperties properties

Component family	Big Data/File	
Function	This component creates a single row flow that displays the properties of a file processed in HDFS.	
Purpose	This component presents the properties of a file processed in HDFS.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. . If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. The schema of this component is read-only. You can click **Edit schema** to view the schema.
	HDFS directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	Get file checksum	Select this check box to generate and output the MD5 information of the file processed.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tHDFSProperties** can be standalone component or send the information it generates to its following component.	

Limitation	JRE 1.6+ is required.

Related scenario

Related topic, see section *Scenario: Displaying the properties of a processed file*

Related topic, see section *Scenario: Iterating on a HDFS directory*

tHDFSPut

tHDFSPut properties

Component family	Big Data / Hadoop	
Function	**tHDFSPut** loads data into Hadoop distributed file system(HDFS).	
Purpose	**tHDFSPut** connects to Hadoop distributed file system to load large-scale files into it with optimized performance.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	HDFS directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	Local directory	Local directory where are stored the files to be loaded into HDFS.
	Overwrite file	Options to overwrite or not the existing file with the new one.
	Files	In the **Files** area, the fields to be completed are: - **File mask**: type in the file name to be selected from the local directory. Regular expression is available. - **New name**: give a new name to the loaded file.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component combines HDFS connection and data extraction, thus usually used as a single-component subjob but also possible as output or end object.	
Limitations	JRE 1.6+ is required.	

Related scenario

For related scenario, see section *Scenario: Computing data with Hadoop distributed file system*.

tHDFSRename

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSRename Properties

Component Family	Big Data/HDFS	
Function	**tHDFSRename** renames the selected files or specified directory on HDFS.	
Purpose	**tHDFSRename** renames files selected from a local directory towards a distant HDFS directory.	
Basic settings	*Use an existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	HDFS directory	Browse to, or enter the directory in HDFS where the data you need to use is.
	Overwrite file	Select the options to overwrite or not the existing file with the new one.
	Files	Click the **[+]** button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions. **New name**: name to give to the HDFS file after the transfer.
	Die on error	This check box is selected by default. Clear the check box to skip the row in error and complete the process for error-free rows.

Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used to compose a single-component Job or Subjob.	
Limitation	JRE 1.6+ is required.	

Related scenario

For related scenario, see section *Scenario: Computing data with Hadoop distributed file system*.

tHDFSRowCount

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of .*

tHDFSRowCount properties

Component family	Big Data/File	
Function	This component reads a file in HDFS row by row in order to determine the number of rows this file contains.	
Purpose	This component counts the number of rows in a file in HDFS.	
Basic settings	*Property Type*	**Built-in**: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*. **Repository**: You have already created the schema and stored it in the Repository. You can reuse it in various projects and Job designs. Related topic: see the user guide of the Studio for integration. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Use an existing connection	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hadoop version	Select the version of the Hadoop distribution you are using.
Connection	*NameNode URI*	Type in the URI of the Hadoop NameNode.
	User name	Enter the user authentication name of HDFS.
	Group	Enter the membership including the authentication user under which the HDFS instances were started. This field is available depending on the distribution you are using.
	Use kerberos authentication	If you are accessing the Hadoop cluster running with Kerberos security, select this check box, then, enter the Kerberos principal name for the NameNode in the field displayed. This check box is available depending on the Hadoop distribution you are connecting to.
	File name	Browse to, or enter the directory in HDFS where the data you need to use is.
	Row separator	Enter the separator used to identify the end of a row.
	Ignore empty rows	Select this check box to skip the empty rows.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop.

		For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tHDFSRowCount** is a standalone component; it must be used with a **OnSubjobOk** connection to **tJava** in order to return the row count. The valid code for **tJava** to get this count could be: `System.out.print(((Integer)globalMap.get("tHDFSRowCount_1_COUNT")));` In this example, *tHDFSRowCount_1* is the label of this component in a Job, so it may vary among different use cases; **COUNT** is the global variable of **tHDFSRowCount**, representing the integer flow of the row count. For further information about how to label a component or how to use a global variable in a Job, see the user guide of the Studio for integration.	
Limitation	JRE 1.6+ is required.	

Related scenario

No scenario is available for this component yet.

tHiveClose

tHiveClose properties

Component Family	Big Data / Hive	
Function	**tHiveClose** closes an active connection to a database.	
Purpose	This component closes connection to a Hive databases.	
Basic settings	*Component list*	If there is more than one connection used in the Job, select **tHiveConnection** from the list.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is generally used as an input component. It requires an output component.	
Limitation	n/a	

Related scenario

This component is for use with **tHiveConnection**. It is generally used along with **tHiveConnection** as the latter allows you to open a connection for the transaction which is underway.

For a scenario in which **tHiveClose** might be used, see section *tMysqlConnection*.

tHiveConnection

tHiveConnection properties

Database Family	Big Data / Hive	
Function	**tHiveConnection** opens a connection to a database in order that a transaction may be made.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
Version	*Distribution*	Select the product you are using as the Hadoop distribution. The options in the list vary depending on the component you are using.
	Hive version	Select the version of the Hadoop distribution you are using.
	Connection mode	Select a connection mode from the list. The options vary depending on the distribution you are using.
	Host	Database server IP address.
	Port	DB server listening port.
	Database	Fill this field with the name of the database. This field is not available when you select **Embedded** from the **Connection mode** list.
	Username and Password	DB user authentication data.
Hadoop properties	*Set Jobtracker URI*	Select this check box to indicate the location of the Hadoop Jobtracker service to be used. For example, we assume that you have chosen a machine called *machine1* as the JobTracker, then set its location as *machine1:portnumber*. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Set NameNode URI	Select this check box to indicate the location of the NameNode of the Hadoop cluster to be used. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting

		the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is generally used with other Hive components, particularly **tHiveClose**. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed.	
Limitation	n/a	

Related scenario

For a scenario in which **tHiveConnection**, might be used, see section *Scenario: Inserting data in mother/daughter tables*.

tHiveInput

tHiveInput properties

Component family	Big Data / Hive	
Function	**tHiveInput** is the dedicated component to the Hive database (the Hive data warehouse system). It executes the given HiveQL query in order to extract the data of interest from Hive. It provides the SQLBuilder tool to help you write your HiveQL statements easily.	
Purpose	**tHiveInput** executes the select queries to extract the corresponding data and sends the data to the component that follows.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tHiveConnection** component from the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see*Talend Open Studio for Big Data User Guide*.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hive version	Select the version of the Hadoop distribution you are using.
	Connection mode	Select a connection mode from the list. The options vary depending on the distribution you are using.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Fill this field with the name of the database. This field is not available when you select **Embedded** from the **Connection mode** list.
	Username and *Password*	DB user authentication data.
Hadoop properties	*Set Jobtracker URI*	Select this check box to indicate the location of the Hadoop Jobtracker service to be used. For example, we assume that you have

		chosen a machine called *machine1* as the JobTracker, then set its location as *machine1:portnumber*. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Set NameNode URI	Select this check box to indicate the location of the NameNode of the Hadoop cluster to be used. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Guess schema	Click this button to retrieve the schema from the table.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition. For further information about the Hive query language, see https://cwiki.apache.org/Hive/languagemanual.html.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns. Clear the **Trim all the String/Char columns** check box to enable **Trim column** in this field.
	Hadoop properties	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop.

		For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Temporary path	If you do not want to set the Jobtracker and the NameNode when you execute the query `select * from your_table_name`, you need to set this temporary path. For example, */C:/select_all* in Windows.
Usage	This component offers the benefit of flexible DB queries and covers all possible Hive QL queries. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed.	

Related scenarios

For related topic, see section *Scenario 1: Writing columns from a MySQL database to an output file*.

tHiveRow

tHiveRow properties

Component family	Big Data / Hive	
Function	**tHiveRow** is the dedicated component for this database. It executes the HiveQL query stated in the specified database. The row suffix means the component implements a flow in the Job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tHiveRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write your HiveQL statements easily.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tHiveConnection** component from the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list. The options in the list vary depending on the component you are using.
	Hive version	Select the version of the Hadoop distribution you are using.
	Connection mode	Select a connection mode from the list. The options vary depending on the distribution you are using.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Fill this field with the name of the database. This field is not available when you select **Embedded** from the **Connection mode** list.
	Username and *Password*	DB user authentication data.

Hadoop properties	*Set Jobtracker URI*	Select this check box to indicate the location of the Hadoop Jobtracker service to be used. For example, we assume that you have chosen a machine called *machine1* as the JobTracker, then set its location as *machine1:portnumber*. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Set NameNode URI	Select this check box to indicate the location of the NameNode of the Hadoop cluster to be used. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition. For further information about the Hive query language, see https://cwiki.apache.org/Hive/languagemanual.html.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Hadoop properties	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized.

		Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Temporary path	If you do not want to set the Jobtracker and the NameNode when you execute the query `select * from your_table_name`, you need to set this temporary path. For example, */C:/select_all* in Windows.
Usage	This component offers the benefit of flexible DB queries and covers all possible Hive QL queries. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tMongoDBClose

tMongoDBClose properties

Component family	Big Data / MongoDB	
Function	**tMongoDBClose** closes a Mongo database connection.	
Purpose	This component closes a connection to the Mongo database.	
Basic settings	*Component list*	Select the **tMongoDBConnection** component in the list if more than one connection is created for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with other Mongo components, especially **tMongoDBConnection**.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario 1: Creating a collection and writing data to it*.

tMongoDBConnection

tMongoDBConnection properties

Component family	Big Data / MongoDB	
Function	**tMongoDBConnection** opens a connection to a database in order that a transaction may be made.	
Purpose	This component allows you to create a connection to a Mongo database and reuse that connection in other components.	
Basic settings	*Use replica set address*	Select this check box to show the **Replica address** table. In the **Replica address** table, you can define multiple Mongo database servers for failover.
	Server and *Port*	IP address and listening port of the database server. Available when the **Use replica set address** check box is not selected.
	Database	Name of the database.
	Required authentication	Select this check box to enable the database authentication.
	Username and *Password*	DB user authentication data. Available when the **Required authentication** check box is selected.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is generally used with other Mongo components, particularly **tMongoClose**.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario 1: Creating a collection and writing data to it*.

tMongoDBInput

tMongoDBInput Properties

Component family	Big Data / MongoDB	
Function	**tMongoDBInput** retrieves records from a collection in the Mongo database and transfers them to the following component.	
Purpose	This component allows you to retrieve records from a collection in the Mongo database and transfer them to the following component for display or storage.	
Basic settings	*Use existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component List** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Use replica set address	Select this check box to show the **Replica address** table. In the **Replica address** table, you can define multiple Mongo database servers for failover. Available when the **Use existing connection** check box is not selected.
	Server and *Port*	IP address and listening port of the database server. Available when the **Use existing connection** or **Use replica set address** check box is not selected.
	Database	Name of the database.
	Required authentication	Select this check box to enable the database authentication.
	Username and *Password*	DB user authentication data. Available when the **Required authentication** check box is selected.
	Collection	Name of the collection in the Mongo database.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Query	Specify the query condition.
	Mapping	Specify the parent node for the column in the Mongo database.
	Sort by	Specify the column and choose the order for the sort operation.
	Limit	Type in the maximum number of records to be retrieved.

Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at the component level.
Usage	**tMongoDBInput** allows you to retrieve records from a collection in the Mongo database and transfer them to the following component for display or storage.	
Limitation	N/A	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Creating a collection and writing data to it*
- section *Scenario: Using Mongo functions to create a collection and write data to it*

tMongoDBOutput

tMongoDBOutput Properties

Component family	Big Data / MongoDB	
Function	**tMongoDBOutput** executes the action defined on the collection in the Mongo database based on the flow incoming from the preceding component in the Job.	
Purpose	This component executes the action defined on the collection in the Mongo database.	
Basic settings	*Use existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component List** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Use replica set address	Select this check box to show the **Replica address** table. In the **Replica address** table, you can define multiple Mongo database servers for failover. Available when the **Use existing connection** check box is not selected.
	Server and *Port*	IP address and listening port of the database server. Available when the **Use existing connection** or **Use replica set address** check box is not selected.
	Database	Name of the database.
	Required authentication	Select this check box to enable the database authentication.
	Username and *Password*	DB user authentication data. Available when the **Required authentication** check box is selected.
	Collection	Name of the collection in the Mongo database.
	Drop collection if exist	Select this check box to drop the collection if it already exists.
	Action on data	The following operations are available: **Insert**: insert data. **Update**: update data. **Upsert**: update and insert data. **Delete**: delete data.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema.

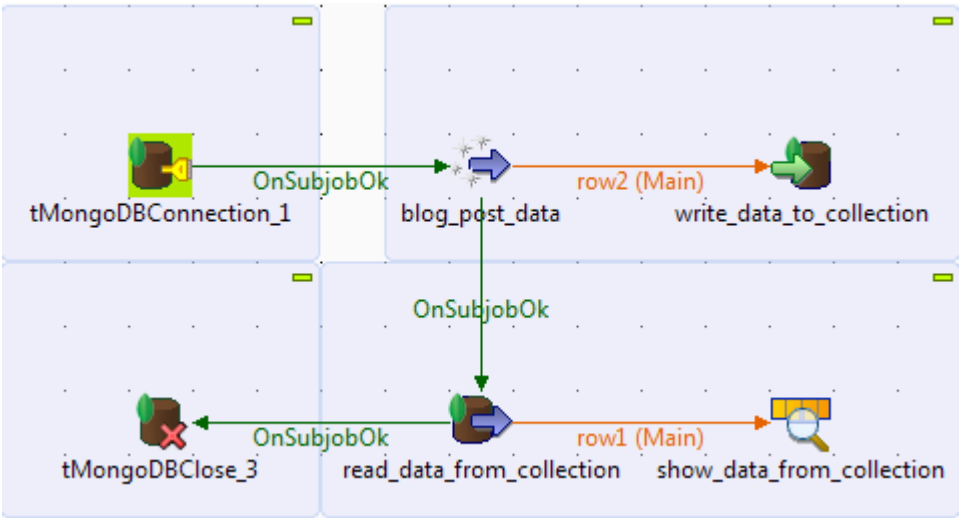
		Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Mapping	Specify the parent node for the column in the Mongo database. Not available when the **Generate JSON Document** check box is selected in **Advanced settings**.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Generate JSON Document*	Select this check box for JSON configuration: **Configure JSON Tree**: click the **[...]** button to open the interface for JSON tree configuration. **Group by**: click the **[+]** button to add lines and choose the input columns for grouping the records. **Remove root node**: select this check box to remove the root node. **Data node** and **Query node** (available for update and upsert actions): type in the name of data node and query node configured on the JSON tree. They are not stored in the database and are intended solely to start the update and upsert actions.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	**tMongoDBOutput** executes the action defined on the collection in the Mongo database based on the flow incoming from the preceding component in the Job.	
Limitation	• The "multi" parameter, which allows to update multiple documents at a time, is not supported. Therefore, if two documents have the same key, the first is always updated, but the second never will. • For the update operation, the key cannot be a JSON array.	

Scenario 1: Creating a collection and writing data to it

This scenario creates the collection *blog* and writes post data to it.

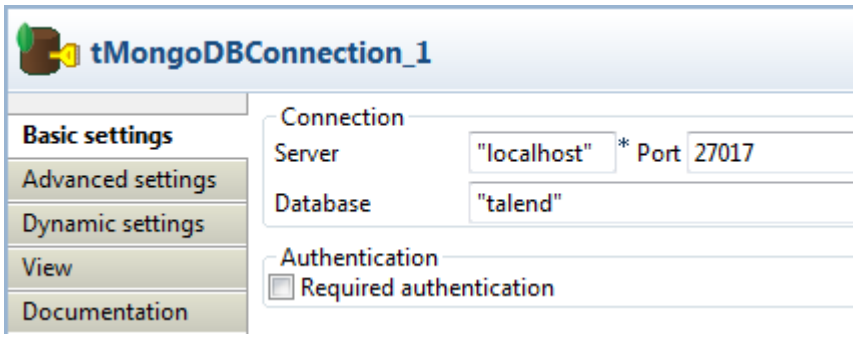
Linking the components

1. Drop **tMongoDBConnection**, **tFixedFlowInput**, **tMongoDBOutput**, **tMongoDBClose**, **tMongoDBInput** and **tLogRow** onto the workspace.
2. Rename **tFixedFlowInput** as **blog_post_data**, **tMongoDBOutput** as **write_data_to_collection**, **tMongoDBInput** as **read_data_from_collection** and **tLogRow** as **show_data_from_collection**.
3. Link **tMongoDBConnection** to **tFixedFlowInput** using the **OnSubjobOk** trigger.
4. Link **tFixedFlowInput** to **tMongoDBOutput** using a **Row** > **Main** connection.
5. Link **tFixedFlowInput** to **tMongoDBInput** using the **OnSubjobOk** trigger.
6. Link **tMongoDBInput** to **tMongoDBClose** using the **OnSubjobOk** trigger.
7. Link **tMongoDBInput** to **tLogRow** using a **Row** > **Main** connection.



Configuring the components

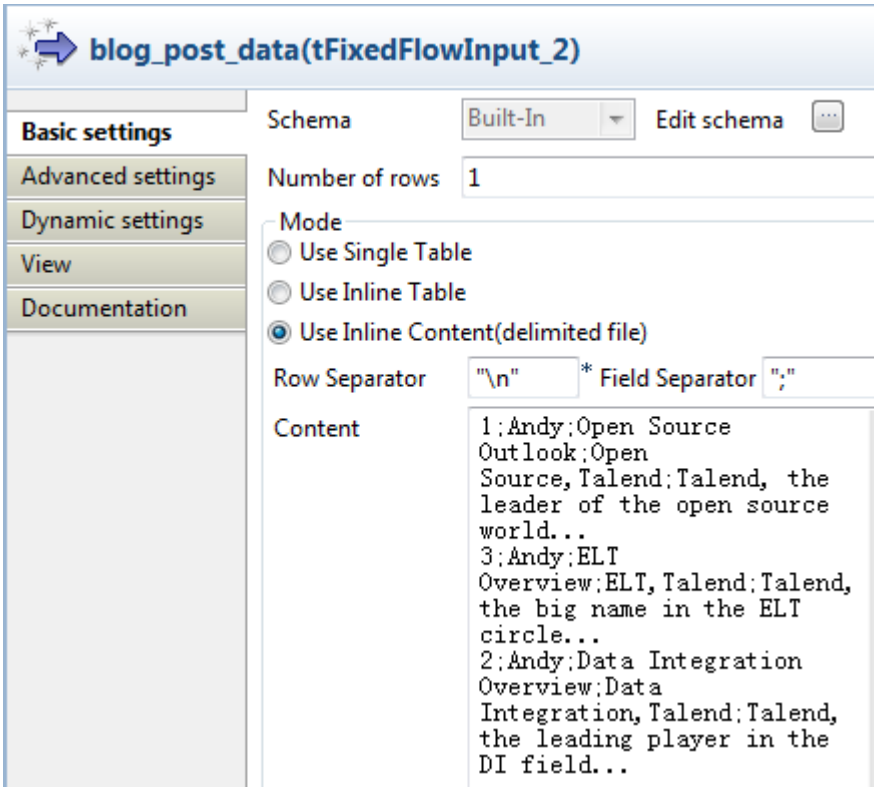
1. Double-click **tMongoDBConnection** to open its **Basic settings** view.



2. In the **Server** and **Port** fields, enter the connection details.

   In the **Database** field, enter the name of the Mongo database.

3. Double-click **tFixedFlowInput** to open its **Basic settings** view.

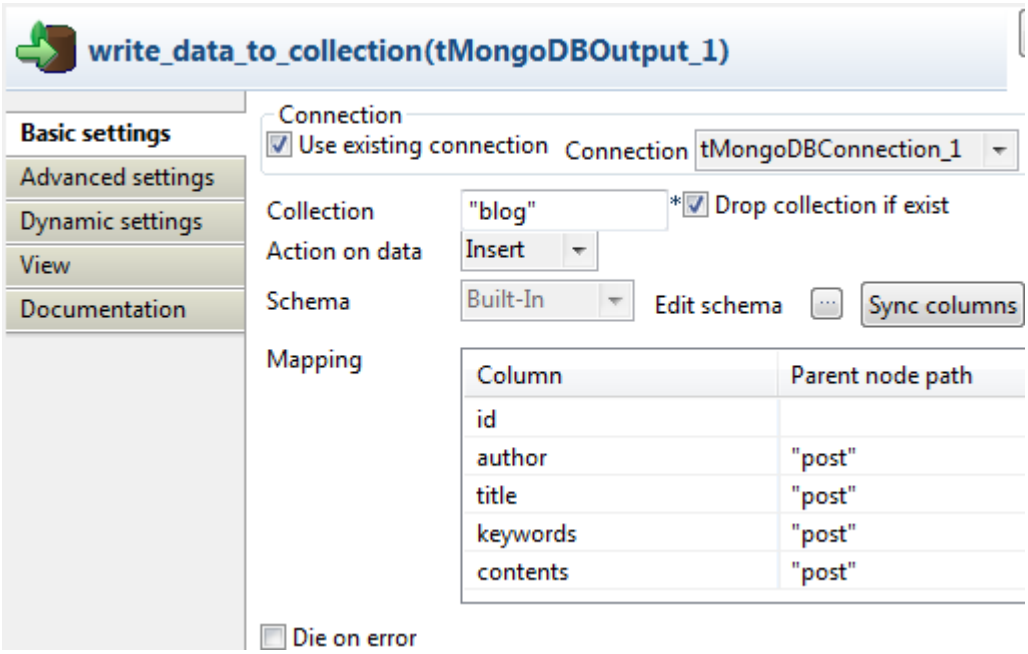


Select **Use Inline Content (delimited file)** in the **Mode** area.

In the **Content** field, enter the data to write to the Mongo database, for example:

```
1;Andy;Open Source Outlook;Open Source,Talend;Talend, the leader of the open source
 world...
3;Andy;ELT Overview;ELT,Talend;Talend, the big name in the ELT circle...
2;Andy;Data Integration Overview;Data Integration,Talend;Talend, the leading player
 in the DI field...
```

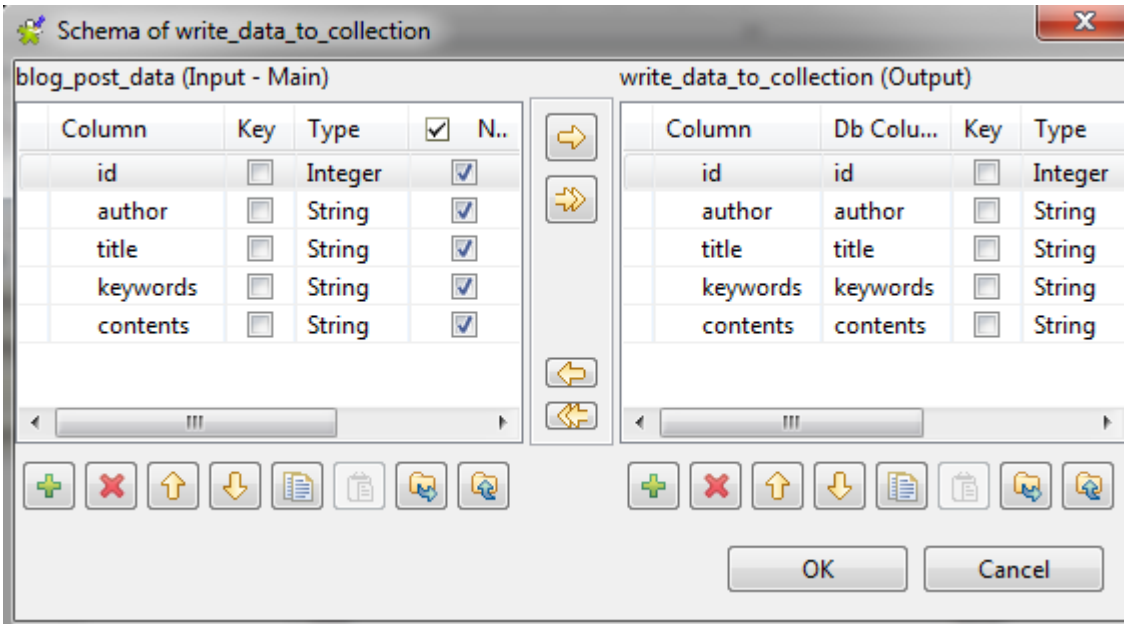
4. Double-click **tMongoDBOutput** to open its **Basic settings** view.



Select the **Use existing connection** and **Drop collection if exist** check boxes.

In the **Collection** field, enter the name of the collection, namely *blog*.

5. Click the **[...]** button next to **Edit schema** to open the schema editor.



6. Click the **[+]** button to add five columns in the right part, namely *id*, *author*, *title*, *keywords* and *contents*, with the type as *Integer* and *String* respectively.

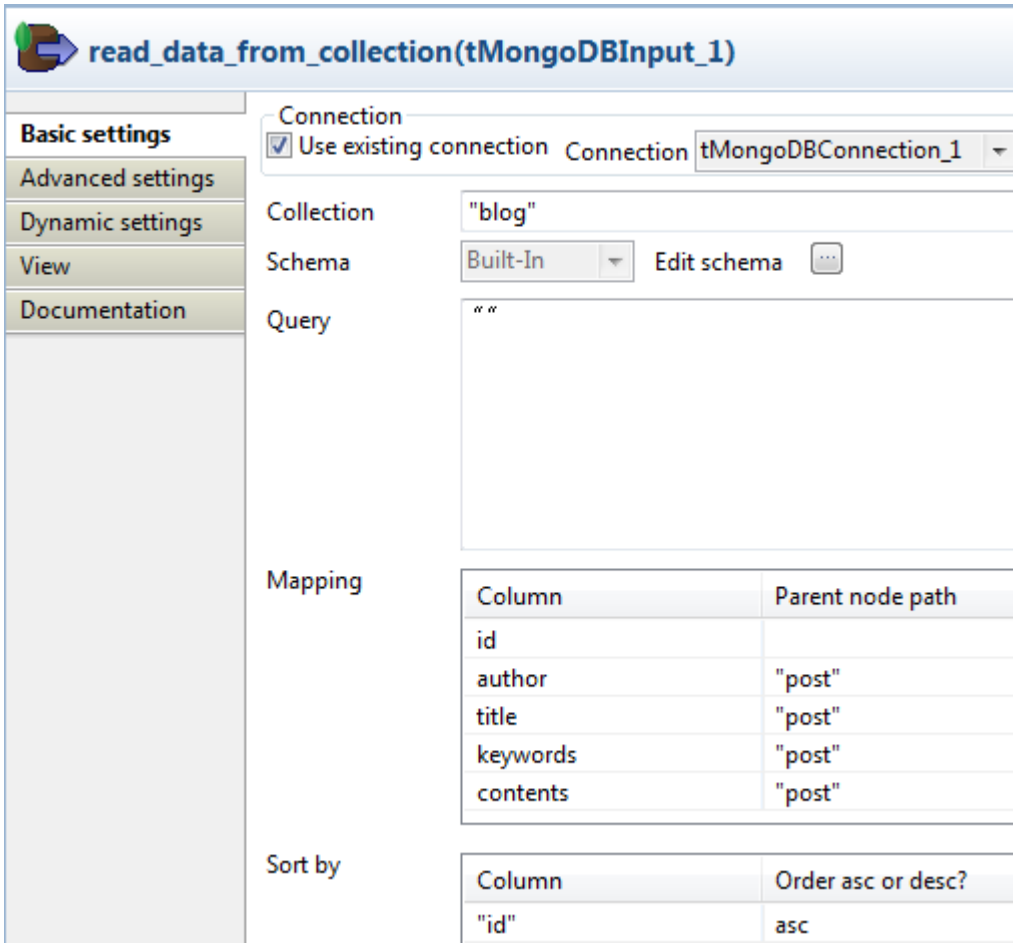
   Click to copy all the columns to the input table.

   Click **Ok** to close the editor.

7. The columns now appear in the left part of the **Mapping** area.

   For columns *author*, *title*, *keywords* and *contents*, enter their parent node *post*. By doing so, those nodes reside under the node *post* in the Mongo collection.

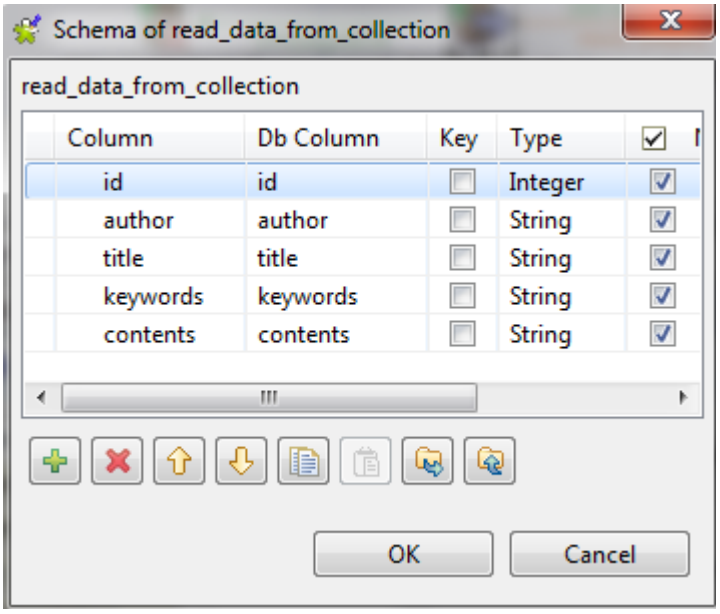
8. Double-click **tMongoDBInput** to open its **Basic settings** view.



Select the **Use existing connection** check box.

In the **Collection** field, enter the name of the collection, namely *blog*.

9. Click the **[...]** button next to **Edit schema** to open the schema editor.



10. Click the **[+]** button to add five columns, namely *id*, *author*, *title*, *keywords* and *contents*, with the type as *Integer* and *String* respectively.

Click **OK** to close the editor.

11. The columns now appear in the left part of the **Mapping** area.

    For columns *author*, *title*, *keywords* and *contents*, enter their parent node *post* so that the data can be retrieved from the correct positions.

12. In the **Sort by** area, click the **[+]** button to add one line and enter *id* under **Column**.

    Select **asc** from the **Order asc or desc?** column to the right of the *id* column. This way, the retrieved records will appear in ascending order of the *id* column.

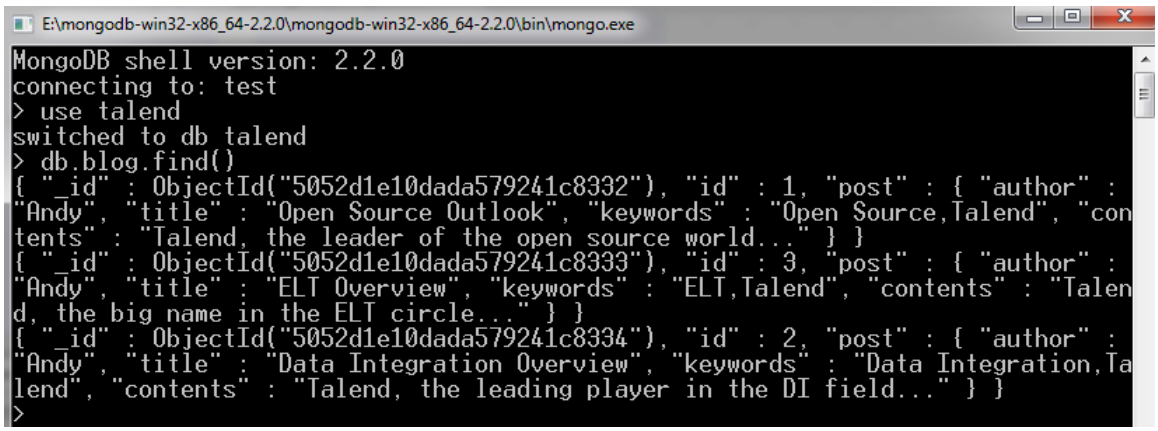
Executing the Job

1. Press **Ctrl+S** to save the Job.

2. Press **F6** to run the Job.

```
[statistics] connecting to socket on port 3630
[statistics] connected
.--+------+-------------------------+--------------------+-----------------------------------------.
|                                        show_data_from_collection                                    |
|=-+------+-------------------------+--------------------+-----------------------------------------=|
|id|author|title                    |keywords            |contents                                 |
|=-+------+-------------------------+--------------------+-----------------------------------------=|
|1 |Andy  |Open Source Outlook      |Open Source,Talend  |Talend, the leader of the open source world...|
|2 |Andy  |Data Integration Overview|Data Integration,Talend|Talend, the leading player in the DI field...|
|3 |Andy  |ELT Overview             |ELT,Talend          |Talend, the big name in the ELT circle...|
'--+------+-------------------------+--------------------+-----------------------------------------'

[statistics] disconnected
Job Try_MongoDB_components ended at 11:45 14/09/2012. [exit code=0]
```

3. Switch to the database *talend* and read data from the collection *blog* in the Mongo command line client. You can find that *author*, *title*, *keywords* and *contents* all reside under the node *post*. Meanwhile, the records are stored in the same order as the source input.



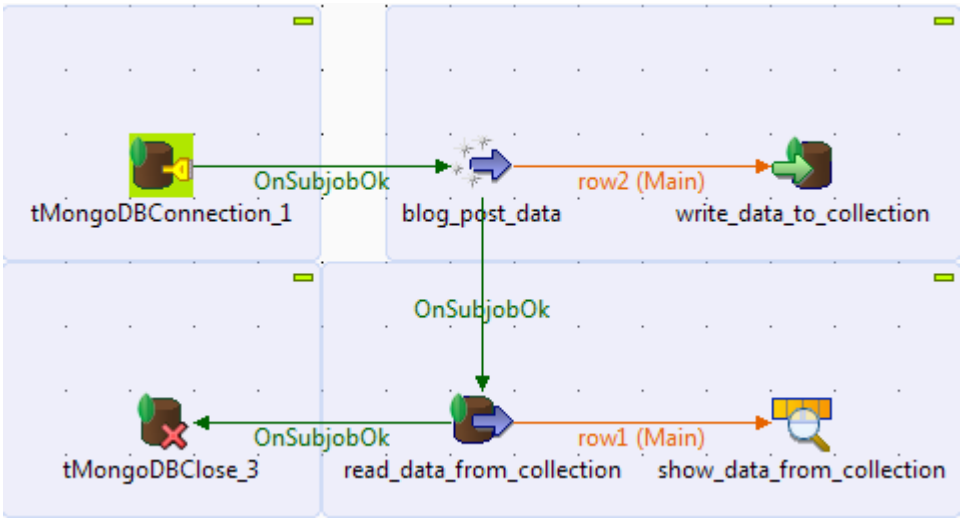
Scenario 2: Upserting records in a collection

This scenario upserts the collection *blog* as an existing record has its author changed and a new record is added. Before the upsert, the collection *blog* looks like:

```
1;Andy;Open Source Outlook;Open Source,Talend;Talend, the leader of the open source
 world...
2;Andy;Data Integration Overview;Data Integration,Talend;Talend, the leading player in
 the DI field...
3;Andy;ELT Overview;ELT,Talend;Talend, the big name in the ELT circle...
```

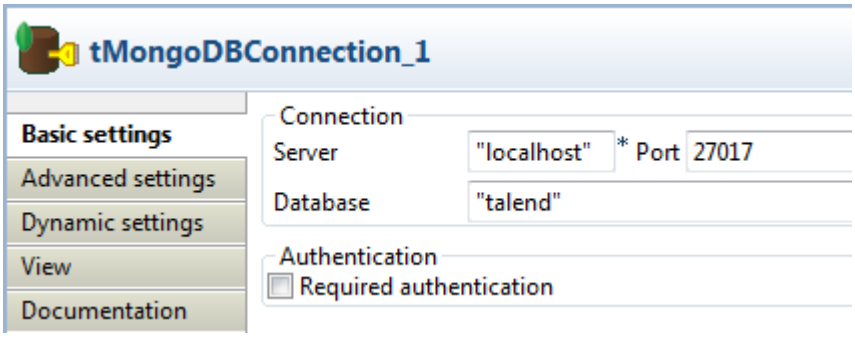
Linking the components

1. Drop **tMongoDBConnection**, **tFixedFlowInput**, **tMongoDBOutput**, **tMongoDBClose**, **tMongoDBInput** and **tLogRow** from the **Palette** onto the design workspace.
2. Rename **tFixedFlowInput** as **blog_post_data**, **tMongoDBOutput** as **write_data_to_collection**, **tMongoDBInput** as **read_data_from_collection** and **tLogRow** as **show_data_from_collection**.
3. Link **tMongoDBConnection** to **tFixedFlowInput** using the **OnSubjobOk** trigger.
4. Link **tFixedFlowInput** to **tMongoDBOutput** using a **Row** > **Main** connection.
5. Link **tFixedFlowInput** to **tMongoDBInput** using the **OnSubjobOk** trigger.
6. Link **tMongoDBInput** to **tMongoDBClose** using the **OnSubjobOk** trigger.
7. Link **tMongoDBInput** to **tLogRow** using a **Row** > **Main** connection.



Configuring the components

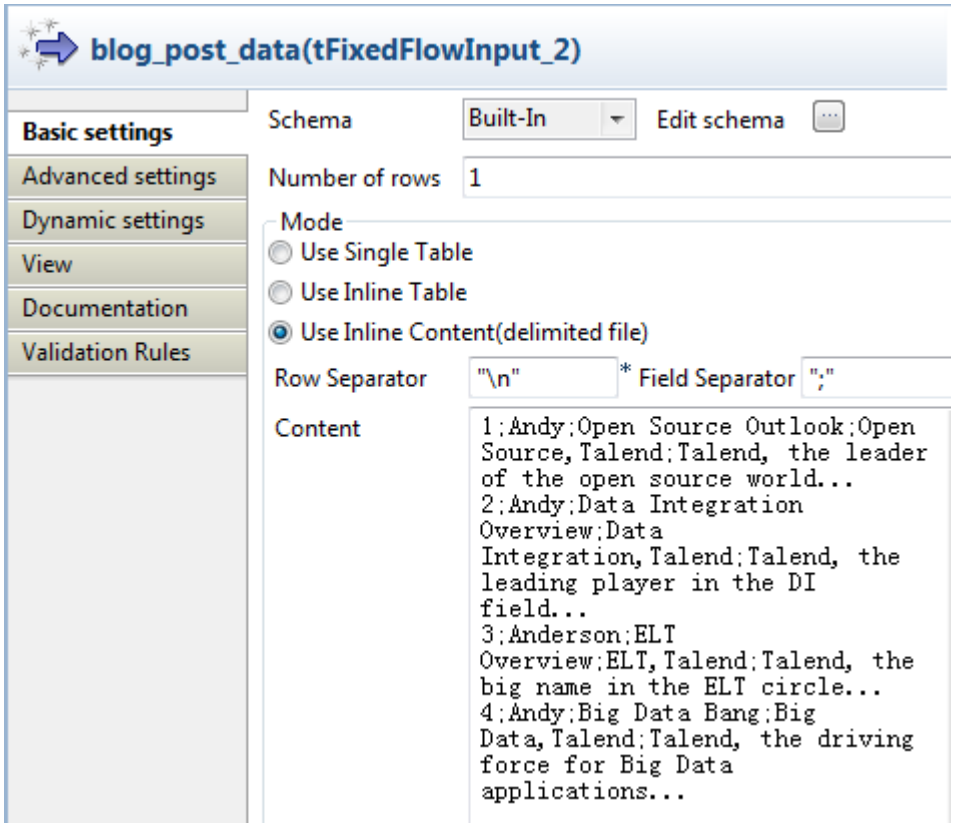
1. Double-click **tMongoDBConnection** to open its **Basic settings** view.



2. In the **Server** and **Port** fields, enter the connection details.

   In the **Database** field, enter the name of the Mongo database.

3. Double-click **tFixedFlowInput** to open its **Basic settings** view.



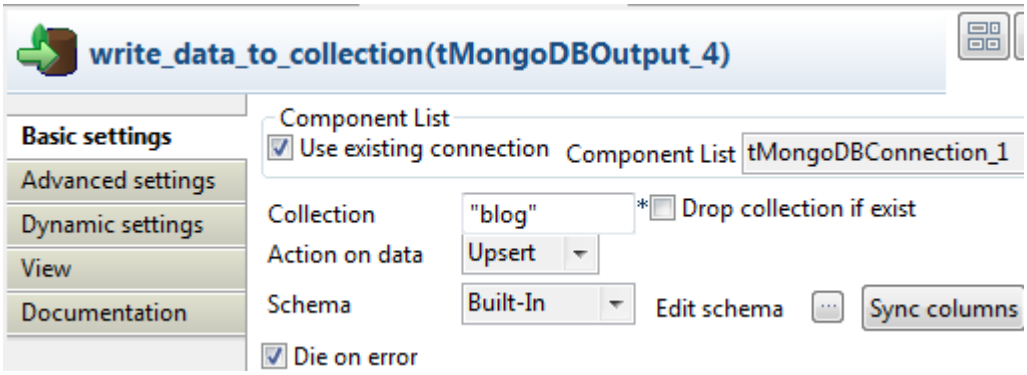
Select **Use Inline Content (delimited file)** in the **Mode** area.

In the **Content** field, enter the data for upserting the Mongo database, for example:

```
1;Andy;Open Source Outlook;Open Source,Talend;Talend, the leader of the open source
 world...
2;Andy;Data Integration Overview;Data Integration,Talend;Talend, the leading player
 in the DI field...
3;Anderson;ELT Overview;ELT,Talend;Talend, the big name in the ELT circle...
4;Andy;Big Data Bang;Big Data,Talend;Talend, the driving force for Big Data
 applications...
```

As shown above, the 3rd record has its author changed and the 4th record is new.

4. Double-click **tMongoDBOutput** to open its **Basic settings** view.

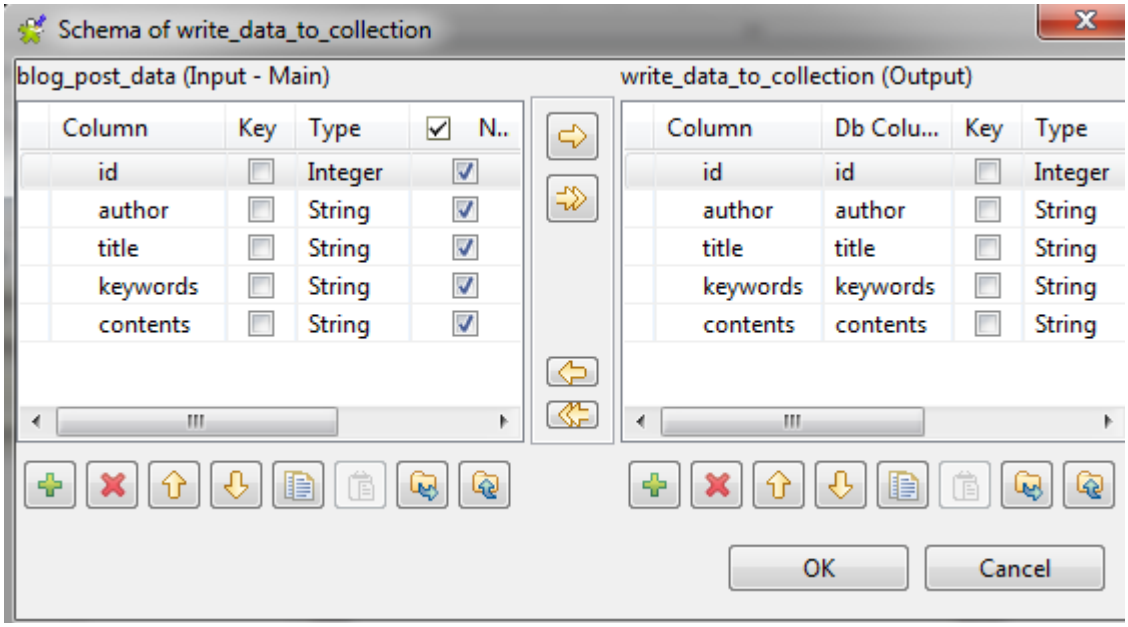


Select the **Use existing connection** and **Die on error** check boxes.

In the **Collection** field, enter the name of the collection, namely *blog*.

Select **Upsert** from the **Action on data** list.

5. Click the **[...]** button next to **Edit schema** to open the schema editor.



6. Click the **[+]** button to add five columns in the right part, namely *id*, *author*, *title*, *keywords* and *contents*, with the type as *Integer* and *String* respectively.

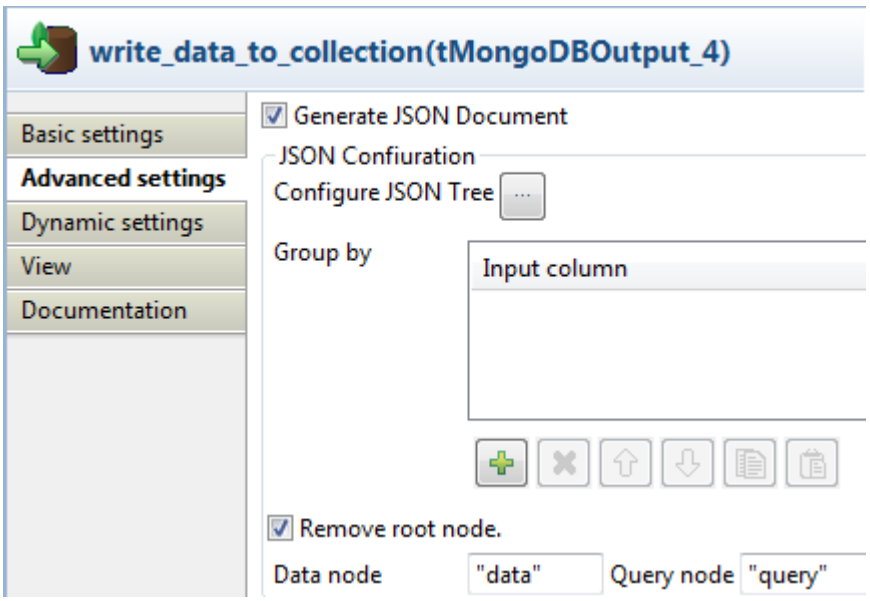
   Click to copy all the columns to the input table.

   Click **Ok** to close the editor.

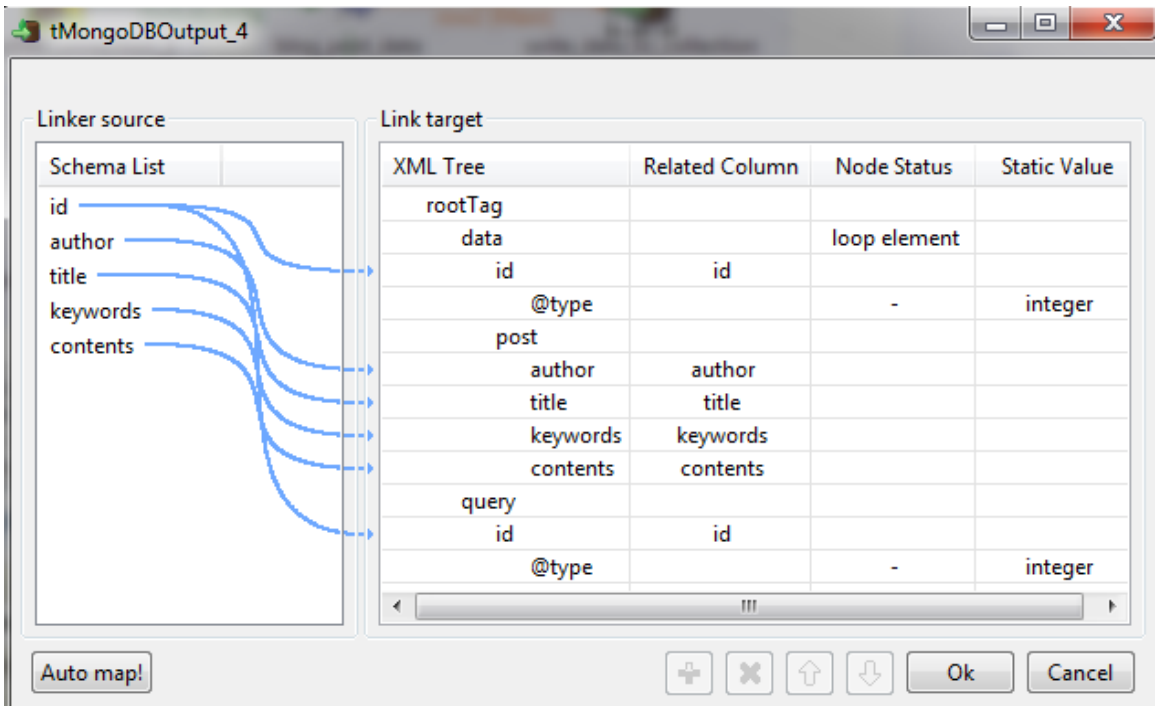
7. In the **Advanced Settings** view, select the **Generate JSON Document** check box.

   Select the **Remove root node** check box.

   In the **Data node** and **Query node** fields, enter *"data"* and *"query"*.

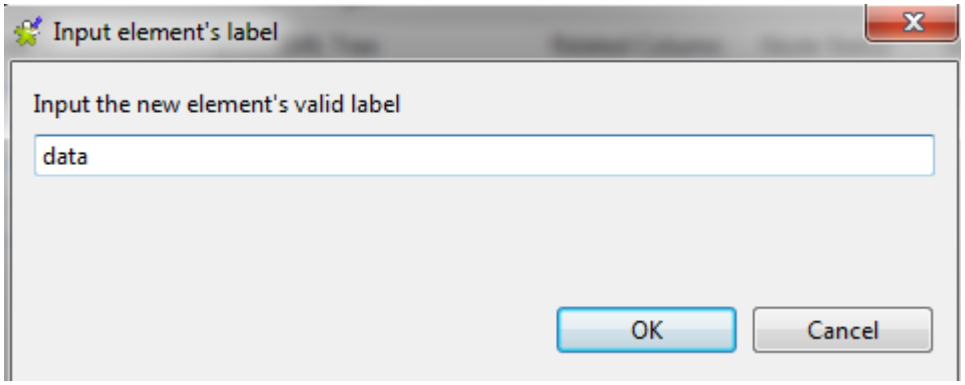


8. Click the **[...]** button next to **Configure JSON Tree** to open the configuration interface.



9. Right-click the node *rootTag* and select **Add Sub-element** from the contextual menu.

   In the dialog box that appears, type in *data* for the **Data node**:



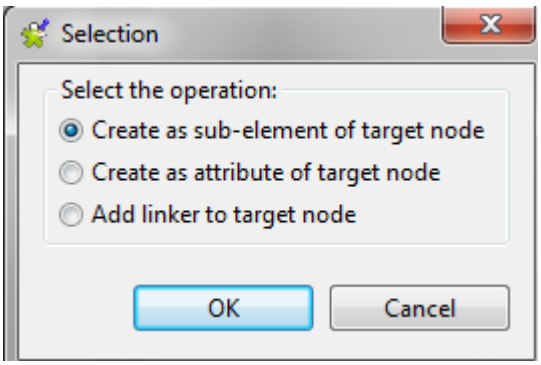
   Click **OK** to close the window.

   Repeat this operation to define *query* as the **Query node**.

   Right-click the node *data* and select **Set As Loop Element** from the contextual menu.

10. Select all the columns under the **Schema list** and drop them to the *data* node.

    In the window that appears, select **Create as sub-element of target node**.

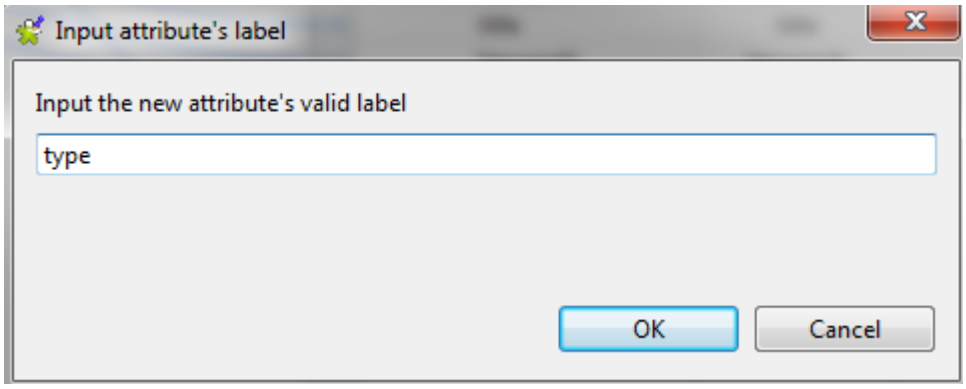


Click **OK** to close the window.

Repeat this operation to drop the *id* column from the **Schema list** under the *Query* node.

11. Right-click the node *id* under *data* and select **Add Attribute** from the contextual menu.

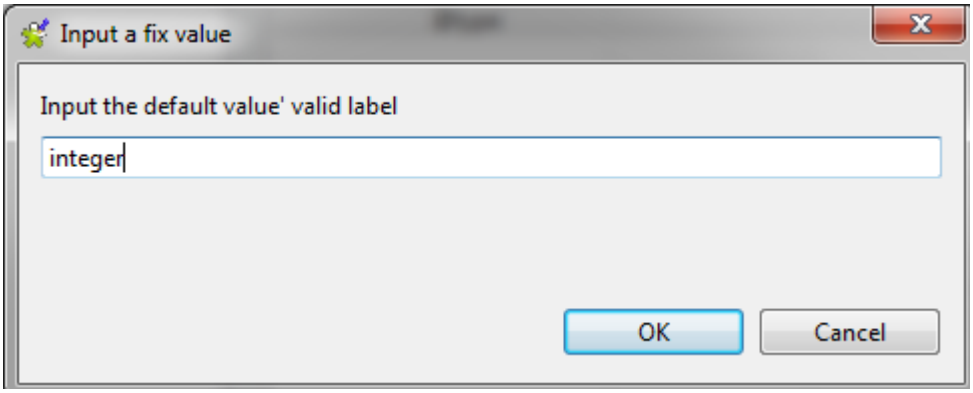
    In the dialog box that appears, type in *type* as the attribute name:



    Click **OK** to close the window.

    Right-click the node *@type* under *id* and select **Set A Fix Value** from the contextual menu.

    In the dialog box that appears, type in *integer* as the attribute value, ensuring the *id* values are stored as integers in the database.

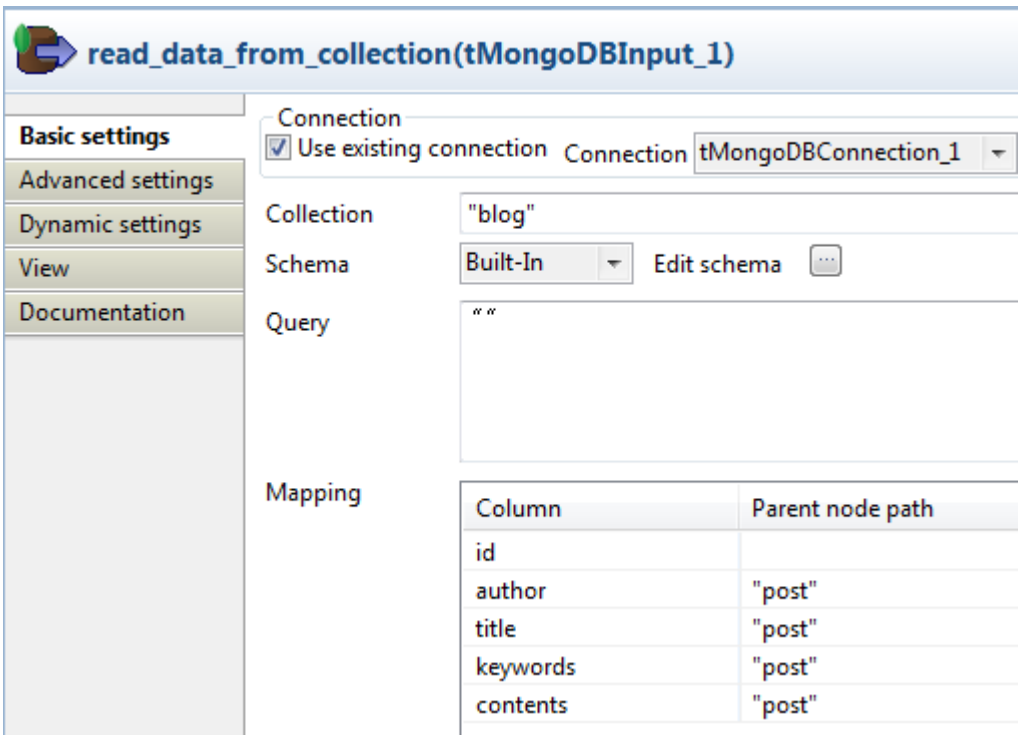


    Click **OK** to close the window.

    Repeat this operation to set this attribute for the *id* node under *Query*.

    Click **OK** to close the JSON Tree configuration interface.

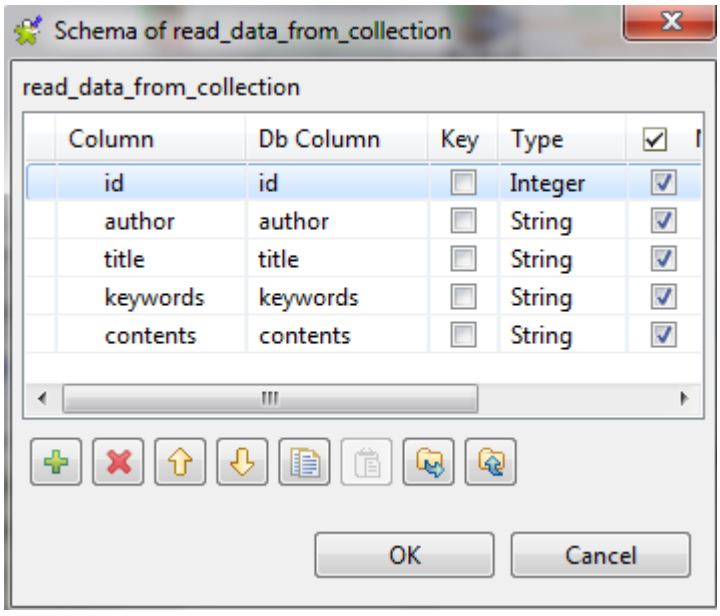
12. Double-click **tMongoDBInput** to open its **Basic settings** view.



Select the **Use existing connection** check box.

In the **Collection** field, enter the name of the collection, namely *blog*.

Click the **[...]** button next to **Edit schema** to open the schema editor.



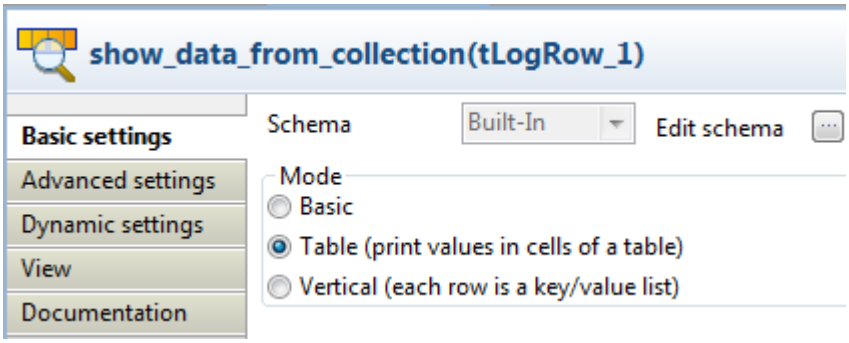
Click the **[+]** button to add five columns, namely *id*, *author*, *title*, *keywords* and *contents*, with the type as *Integer* and *String* respectively.

Click **OK** to close the editor.

The columns now appear in the left part of the **Mapping** area.

For columns *author*, *title*, *keywords* and *contents*, enter their parent node *post* so that the data can be retrieved from the correct positions.

13. Double-click **tLogRow** to open its **Basic settings** view.



In the **Mode** area, select **Table (print values in cells of a table** for better display.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run the Job.

```
+--+--------+-------------------------+---------------------+-----------------------------------------------------+
|                                                  show_data_from_collection                                        |
+==+========+=========================+=====================+=====================================================+
|id|author  |title                    |keywords             |contents                                             |
+==+========+=========================+=====================+=====================================================+
|1 |Andy    |Open Source Outlook      |Open Source,Talend   |Talend, the leader of the open source world...       |
|2 |Andy    |Data Integration Overview|Data Integration,Talend|Talend, the leading player in the DI field...      |
|3 |Anderson|ELT Overview             |ELT,Talend           |Talend, the big name in the ELT circle...            |
|4 |Andy    |Big Data Bang            |Big Data,Talend      |Talend, the driving force for Big Data applications...|
+--+--------+-------------------------+---------------------+-----------------------------------------------------+
```

As shown above, the 3rd record has its author updated and the 4th record is inserted.

tMongoDBRow

tMongoDBRow Properties

Component family	Big Data / MongoDB	
Function	**tMongoDBRow** executes the commands and functions provided by the Mongo database.	
Purpose	This component allows you to execute the commands and functions of the Mongo database.	
Basic settings	*Use existing connection*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component List** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Use replica set address	Select this check box to show the **Replica address** table. In the **Replica address** table, you can define multiple Mongo database servers for failover. Available when the **Use existing connection** check box is not selected.
	Server and *Port*	Address and listening port of the database server. Available when the **Use existing connection** or **Use replica set address** check box is not selected.
	Database	Name of the database.
	Required authentication	Select this check box to enable the database authentication.
	Username and *Password*	DB user authentication data. Available when the **Required authentication** check box is selected.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Execute command	Select this check box to enter Mongo commands in the **Command** field for execution.
	Function	Enter Mongo functions in the **Function** field for execution. Not available when the **Execute command** check box is selected.
	Parameters value	Click the **[+]** button to add lines as needed and then define the parameter values in the form of variables or constant values, for example *row1.author* or *"Andy"*. Note that the parameter values

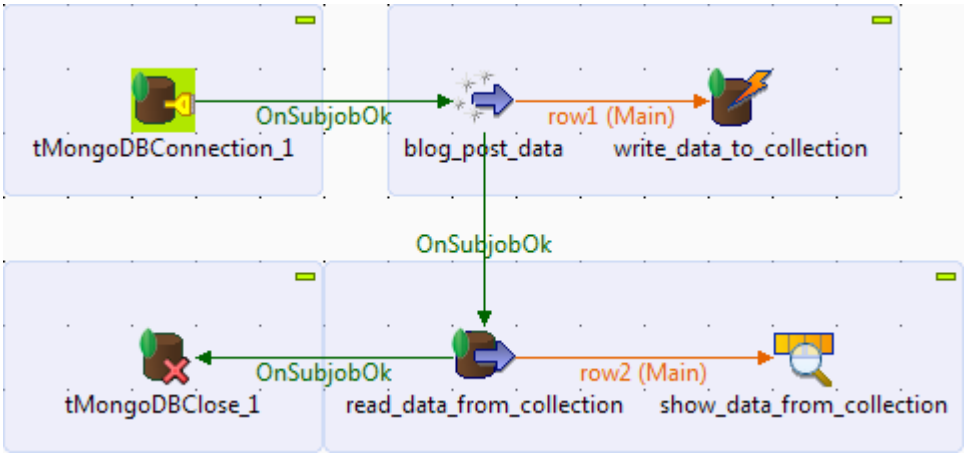
		correspond to the parameters defined in the **Function** field, in the same order. Not available when the **Execute command** check box is selected.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at the component level.
Usage	**tMongoDBRow** allows you to manipulate the Mongo database through the Mongo commands and functions.	
Limitation	n/a	

Scenario: Using Mongo functions to create a collection and write data to it

This scenario creates the collection *blog* and writes post data to it through the Mongo functions.

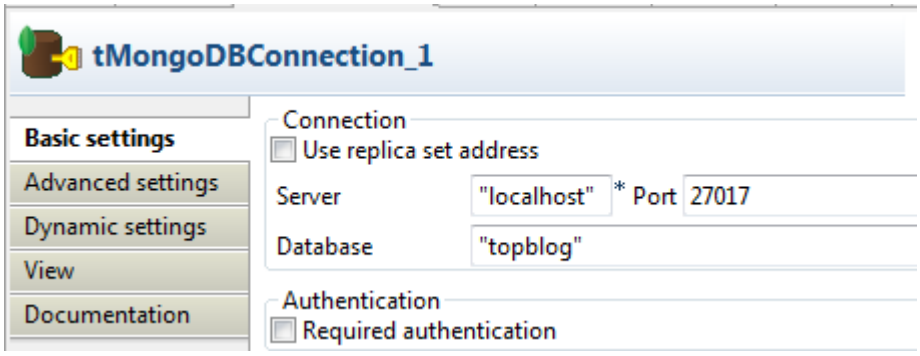
Linking the components

1. Drop **tMongoDBConnection**, **tFixedFlowInput**, **tMongoDBRow**, **tMongoDBClose**, **tMongoDBInput** and **tLogRow** onto the workspace.
2. Rename **tFixedFlowInput** as **blog_post_data**, **tMongoDBRow** as **write_data_to_collection**, **tMongoDBInput** as **read_data_from_collection** and **tLogRow** as **show_data_from_collection**.
3. Link **tMongoDBConnection** to **tFixedFlowInput** using the **OnSubjobOk** trigger.
4. Link **tFixedFlowInput** to **tMongoDBRow** using a **Row** > **Main** connection.
5. Link **tFixedFlowInput** to **tMongoDBInput** using the **OnSubjobOk** trigger.
6. Link **tMongoDBInput** to **tMongoDBClose** using the **OnSubjobOk** trigger.
7. Link **tMongoDBInput** to **tLogRow** using a **Row** > **Main** connection.



Configuring the components

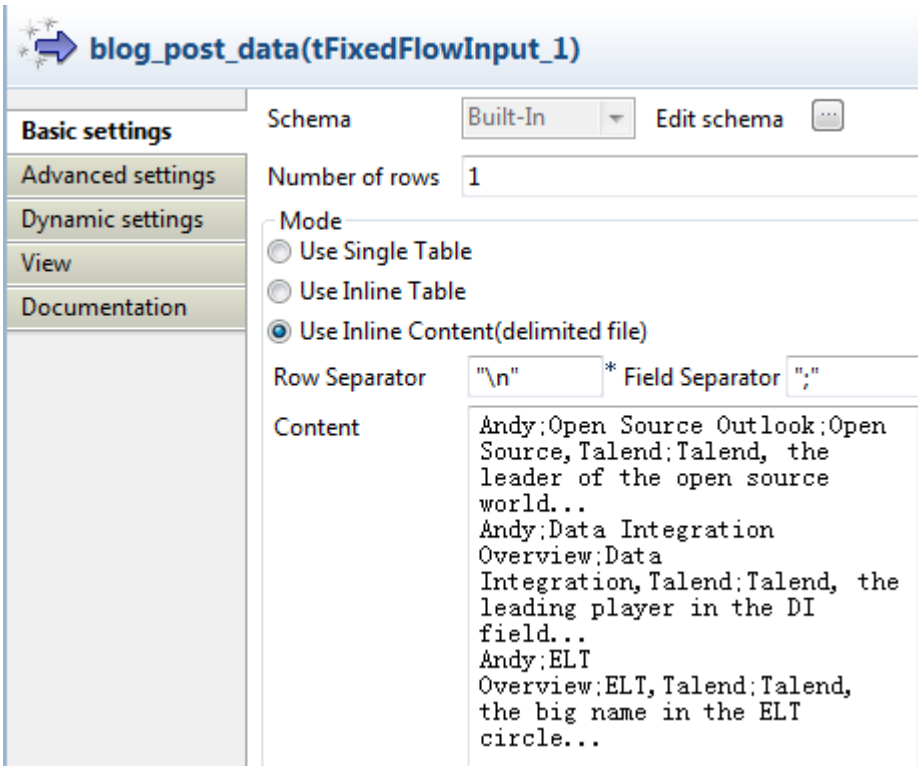
1. Double-click **tMongoDBConnection** to open its **Basic settings** view.



2. In the **Server** and **Port** fields, enter the connection details.

   In the **Database** field, enter the name of the Mongo database.

3. Double-click **tFixedFlowInput** to open its **Basic settings** view.

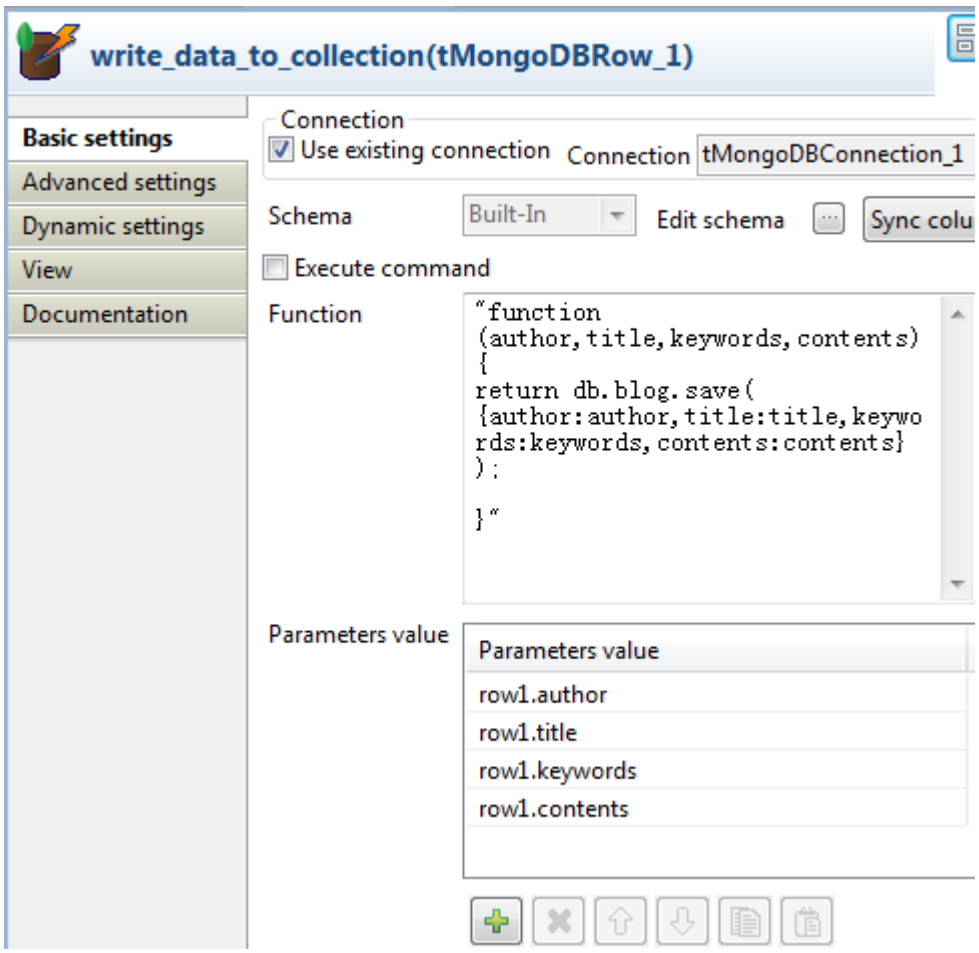


   Select **Use Inline Content (delimited file)** in the **Mode** area.

   In the **Content** field, enter the data to write to the Mongo database, for example:

```
Andy;Open Source Outlook;Open Source,Talend;Talend, the leader of the open source
 world...
Andy;Data Integration Overview;Data Integration,Talend;Talend, the leading player in
 the DI field...
Andy;ELT Overview;ELT,Talend;Talend, the big name in the ELT circle...
```

4. Double-click **tMongoDBRow** to open its **Basic settings** view.

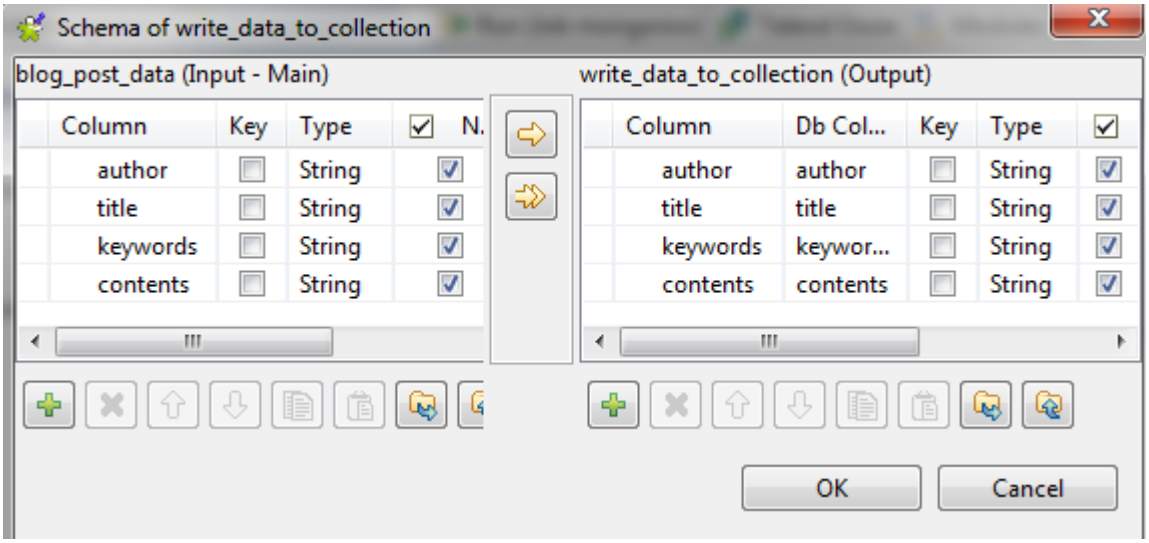


Select the **Use existing connection** check box.

In the **Function** field, enter the Mongo function to create the collection *blog* and insert data to it:

```
"function(author,title,keywords,contents){
return db.blog.save(
{author:author,title:title,keywords:keywords,contents:contents}
);

}"
```

5. Click the **[...]** button next to **Edit schema** to open the schema editor.



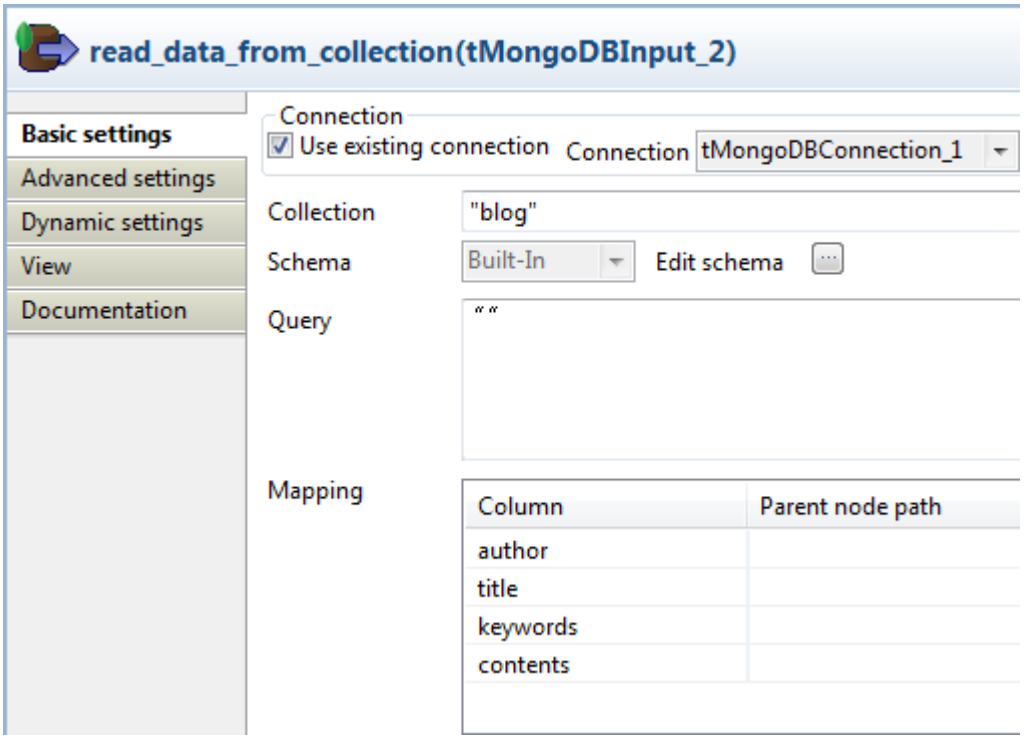
6. Click the **[+]** button to add four columns in the right part, namely *author*, *title*, *keywords* and *contents*, with the type of *String*.

   Click to copy all the columns to the input table.

   Click **OK** to close the editor.

7. In the **Parameters value** table, click the **[+]** button to add four lines and enter the values in sequence: *row1.author*, *row1.title*, *row1.keywords* and *row1.contents*. By doing so, data of *row1* will be transferred to the parameters defined in the function.

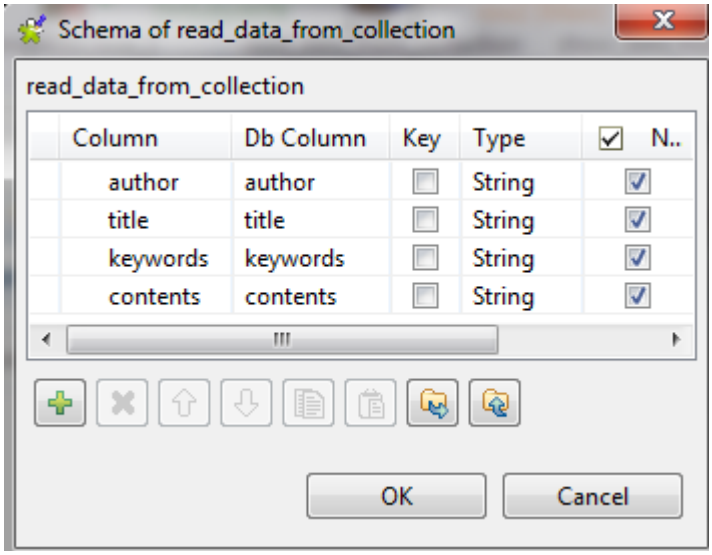
8. Double-click **tMongoDBInput** to open its **Basic settings** view.



   Select the **Use existing connection** check box.

   In the **Collection** field, enter the name of the collection, namely *blog*.

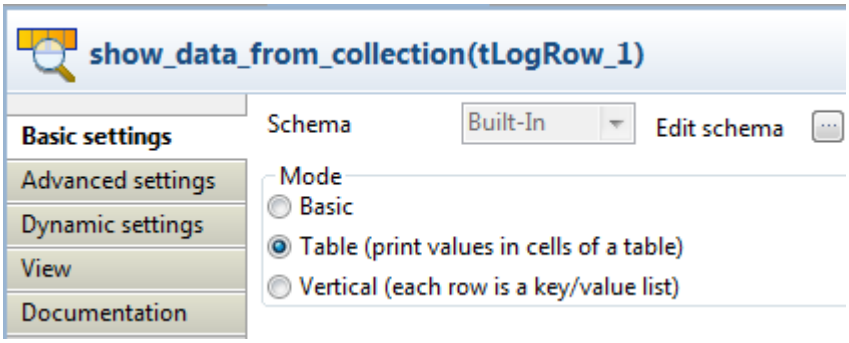
9. Click the **[...]** button next to **Edit schema** to open the schema editor.



10. Click the **[+]** button to add four columns, namely *author*, *title*, *keywords* and *contents*, with the type as *String*.

    Click **OK** to close the editor.

11. Double-click **tLogRow** to open its **Basic settings** view.



    In the **Mode** area, select **Table (print values in cells of a table** for better display.

Executing the Job

1. Press **Ctrl+S** to save the Job.

2. Press **F6** to run the Job.

```
.------+-------------------------+--------------------+-----------------------------------------.
|                                        show_data_from_collection                                 |
|=------+-------------------------+--------------------+-----------------------------------------=|
|author|title                     |keywords            |contents                                 |
|=------+-------------------------+--------------------+-----------------------------------------=|
|Andy   |Open Source Outlook      |Open Source,Talend  |Talend, the leader of the open source world...|
|Andy   |Data Integration Overview|Data Integration,Talend|Talend, the leading player in the DI field... |
|Andy   |ELT Overview             |ELT,Talend          |Talend, the big name in the ELT circle...|
'------+-------------------------+--------------------+-----------------------------------------'
```

tPigAggregate

tPigAggregate Properties

Component family	Big Data / Hadoop	
Function	This component allows you to group the original data by column and add one or more additional columns to the output of preceding grouped data.	
Purpose	The **tPigAggregate** component adds one or more additional columns to the output of the grouped data to create data to be used by Pig.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Talend Open Studio for Big Data User Guide*.
	Group by	Click the plus button to add one or more columns to set tuples in the source data as group condition.
	Operations	Click the plus button to add one or more columns to generate one or more additional output columns based on conditions: **Additional Output Column**: Select a column in the original data as output column. **Function**: Functions for operation on input data. **Input Column**: Select a column in the original data as input column.
Advanced settings	*Increase parallelism*	Select this check box to set the number of reduce tasks for the **MapReduce** Jobs.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig scripts is required.	

Related scenario

For a **tPigAggregate** related scenario, section *Scenario 1: Aggregating values and sorting data* of **tAggregateRow**.

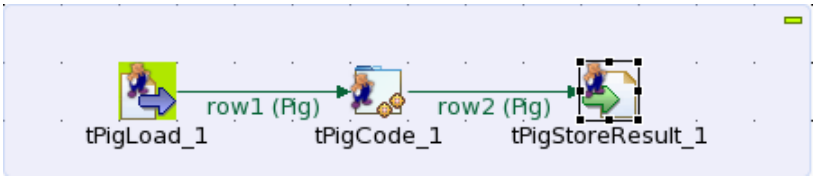
tPigCode

tPigCode Properties

Component family	Big Data / Hadoop	
Function	This component allows you to enter personalized Pig code to integrate it in **Talend** program. You can execute this code only once.	
Purpose	**tPigCode** extends the functionalities of a **Talend** Job through using Pig scripts.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Scripts	Type in Pig scripts you want to execute according to the task you need to perform. For further information about Pig functions syntax specific to **Talend**, see: http://pig.apache.org/docs/r0.8.0/udf.html
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig scripts is required.	

Scenario: Selecting a column of data from an input file and store it into a local file

This scenario describes a three-component Job that selects a column of data that matches filter condition defined in **tPigCode** and stores the result into a local file.



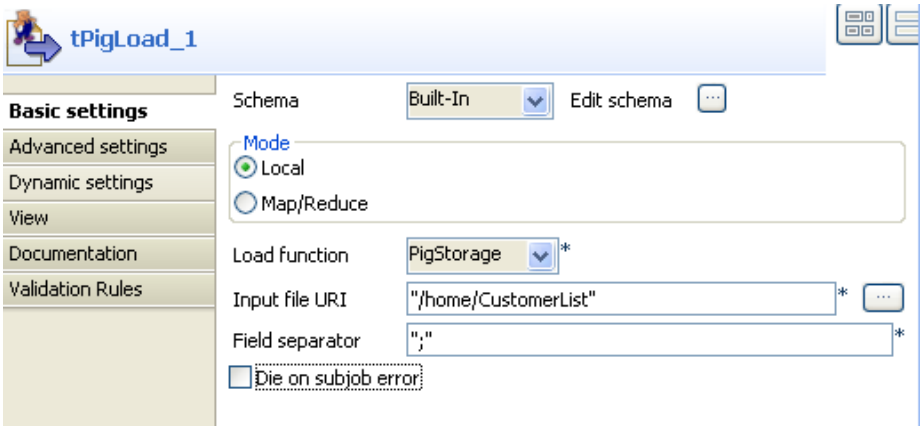
Setting up the Job

1. Drop the following components from the **Palette** to the design workspace: **tPigCode**, **tPigLoad**, **tPigStoreResult**.

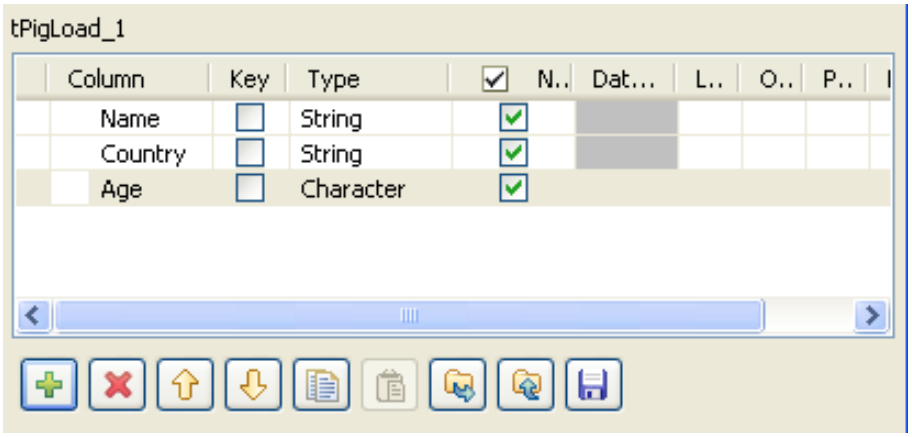
2. Right-click **tPigLoad** to connect it to **tPigCode** using a **Row > Pig Combine** connection.
3. Right-click **tPigCode** to connect it to **tPigStoreResult** using a **Row > Pig Combine** connection.

Loading the data

1. Double-click **tPigLoad** to open its **Basic settings** view.



2. Click the three-dot button next to **Edit schema** to add columns for **tPigLoad**.

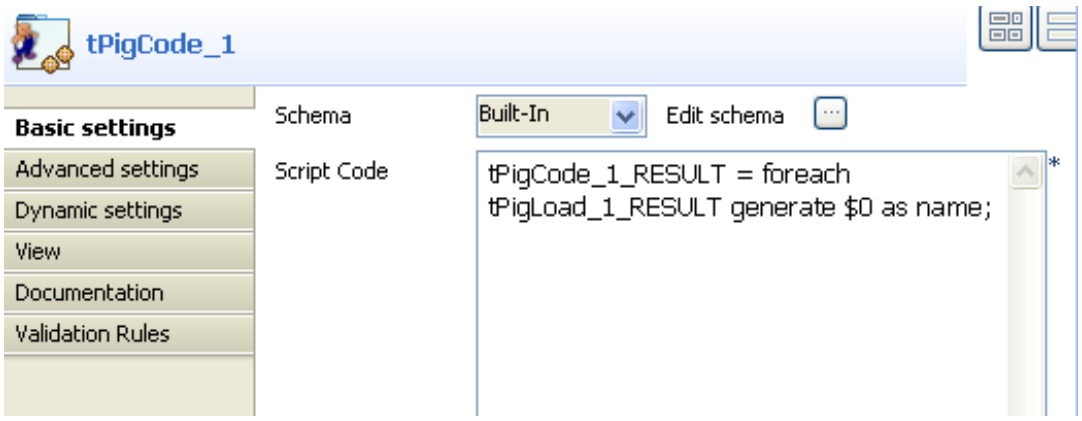


3. Click the plus button to add *Name*, *Country* and *Age* and click **OK** to save the setting.
4. Select **Local** from the **Mode** area.
5. Fill in the **Input filename** field with the full path to the input file.

   In this scenario, the input file is *CustomerList* which contains rows of names, country names and age.
6. Select **PigStorage** from the **Load function** list.
7. Leave rest of the settings as they are.

Configuring the tPigCode component

1. Double-click **tPigCode** component to open its **Basic settings** view.



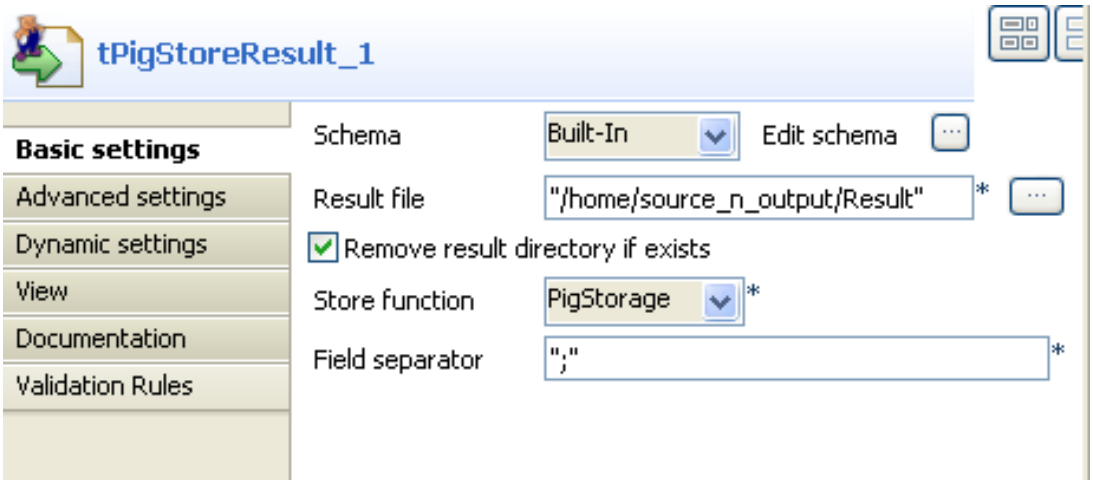
2. Click **Sync columns** to retrieve the schema structure from the preceding component.
3. Fill in the **Script Code** field with following expression:

```
tPigCode_1_RESULT = foreach tPigLoad_1_RESULT generate $0 as name;
```

This filter expression selects column *Name* from *CustomerList*.

Saving the result data to a local file

1. Double-click **tPigStoreResult** to open its **Basic settings** view.



2. Click **Sync columns** to retrieve the schema structure from the preceding component.
3. Fill in the **Result file** field with the full path to the result file.

   In this scenario, the result is saved in *Result* file.
4. Select **Remove result directory if exists**.
5. Select **PigStorage** from the **Store function** list.
6. Leave rest of the settings as they are.

Executing the Job

Save your Job and press **F6** to run it.

The *Result* file is generated containing the selected column of data.

tPigCross

tPigCross Properties

Component family	Big Data / Hadoop	
Function	This component allows you to compute the cross data of two or more relations.	
Purpose	The **tPigCross** component uses CROSS operator to compute the Cartesian product.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Cross filename	Fill in the path of the file to be computed.
	Field separator	Enter the separator used to identify the end of a field.
Advanced settings	*Use partitioner*	Select this check box to specify the Hadoop Partitioner that controls the partitioning of the keys of the intermediate map-outputs. For further information about the usage of Hadoop Partitioner, see: http://hadoop.apache.org/common/docs/r0.20.2/api/org/apache/hadoop/mapred/Partitioner.html
	Increase parallelism	Select this check box to set the number of reduce tasks for the **MapReduce** Jobs
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig scripts is required.	

Related scenario

No scenario is available for this component yet.

tPigDistinct

tPigDistinct Properties

Component family	Big Data / Hadoop	
Function	This component allows you to remove duplicate tuples in a relation.	
Purpose	The **tPigDistinct** component removes duplicate tuples in a relation.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Increase parallelism*	Select this check box to set the number of reduce tasks for the **MapReduce** Jobs
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component. *This component will not maintain the original order in the input file.*	
Limitation	Knowledge of Pig scripts is required.	

Related scenario

For more infomation regarding the **tPigDistinct** component in use, see section *Scenario: Filtering rows of data based on a condition and saving the result to a local file* of **tPigFilterRow**.

tPigFilterColumns

tPigFilterColumns Properties

Component family	Big Data / Hadoop	
Function	This component allows you to select one or more columns from a relation based on defined condition.	
Purpose	This **tPigFilterColumns** component selects data or filters out data from a relation based on defined filter conditions.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig script is required.	

Related Scenario

For a **tPigFilterColumns** related scenario, see section *Scenario: Joining two files based on an exact match and saving the result to a local file* of **tPigJoin**.

tPigFilterRow

tPigFilterRow Properties

Component family	Big Data / Hadoop	
Function	The **tPigFilterRow** component filters the input flow in a Pig chain based on conditions set on given column(s).	
Purpose	This component is used to filter the input flow in a Pig chain based on conditions set on one or more columns.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Filter configuration	Click the **Add** button beneath the **Filter configuration** table to set one or more filter conditions. This table disappears if you select **Use advanced filter**.
	Use advanced filter	Select this check box to define advanced filter condition by entering customized filter expression in the **Filter** field.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as an intermediate step in a Pig chain.	
Limitation	Knowledge of Pig scripts is required.	

Scenario: Filtering rows of data based on a condition and saving the result to a local file

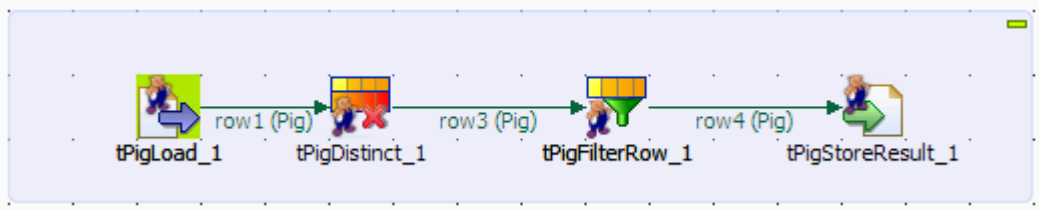
This scenario describes a four-component Job that filters a list of customers to find out customers from a particular country, and saves the result list to a local file. Before the input data is filtered, duplicate entries are first removed from the list.

The input file contains three columns: *Name*, *Country*, and *Age*, and it has some duplicate entries, as shown below:

```
Mario;PuertoRico;49
Mike;USA;22
Ricky;PuertoRico;37
Silvia;Spain;20
Billy;Canada;21
Ricky;PuertoRico;37
Romeo;UK;19
Natasha;Russia;25
Juan;Cuba;23
Bob;Jamaica;55
Mario;PuertoRico;49
```

Dropping and linking components

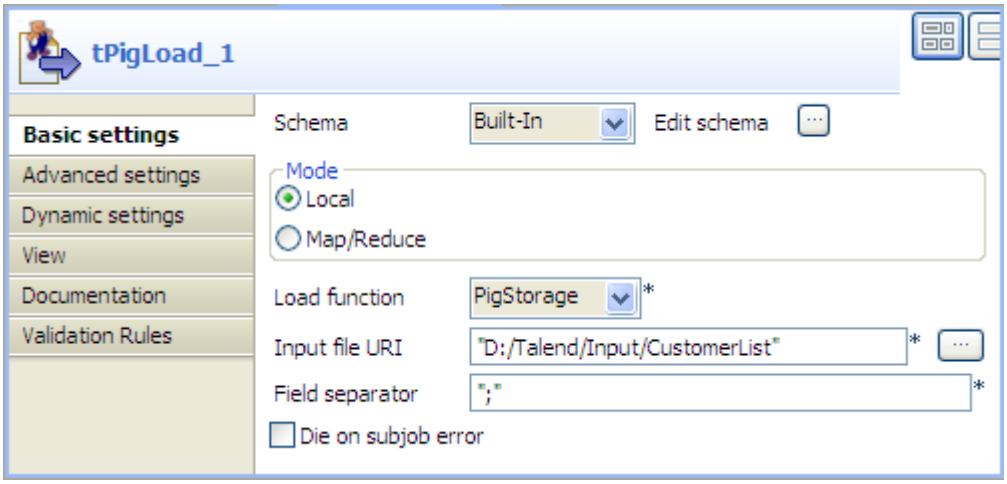
1. Drop the following components from the **Palette** to the design workspace: **tPigLoad**, **tPigDistinct**, **tPigFilterRow**, and **tPigStoreResult**.
2. Right-click **tPigLoad**, select **Row** > **Pig Combine** from the contextual menu, and click **tPigDistinct** to link these two components.
3. Repeat this operation to link **tPigDistinct** to **tPigFilterRow**, and **tPigFilterRow** to **tPigStoreResult** using **Row** > **Pig Combine** connections to form a Pig chain.



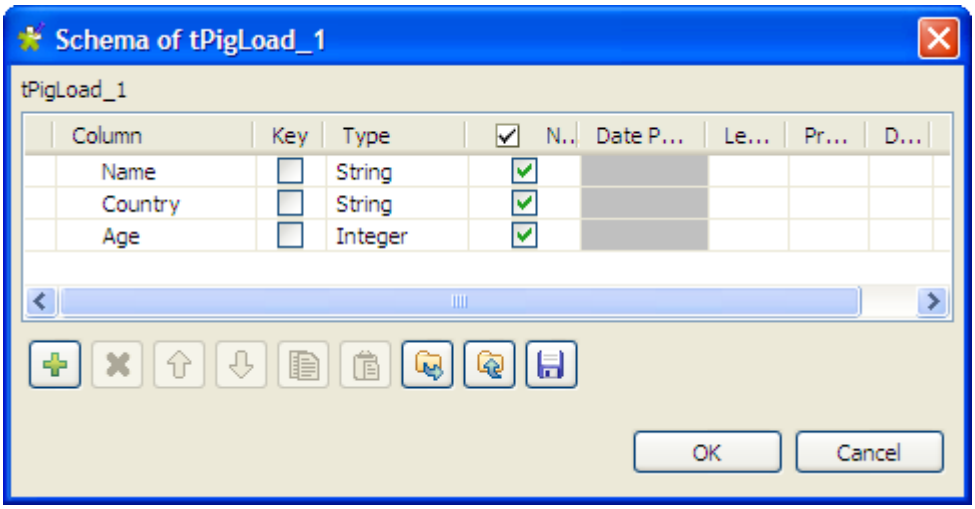
Configuring the components

Loading the input data and removing duplicates

1. Double-click **tPigLoad** to open its **Basic settings** view.



2. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box.



3. Click the **[+]** button to add three columns according to the data structure of the input file: *Name* (string), *Country* (string) and *Age* (integer), and then click **OK** to save the setting and close the dialog box.

4. Click **Local** in the **Mode** area.

5. Fill in the **Input file URI** field with the full path to the input file.

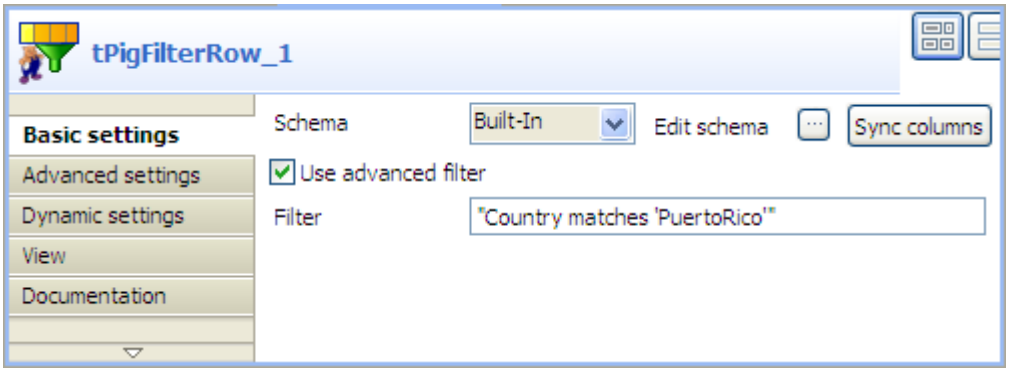
6. Select **PigStorage** from the **Load function** list, and leave rest of the settings as they are.

7. Double-click **tPigDistinct** to open its **Basic settings** view, and click **Sync columns** to make sure that the input schema structure is correctly propagated from the preceding component.

   This component will remove any duplicates from the data flow.

Configuring the filter

1. Double-click **tPigFilterRow** to open its **Basic settings** view.



2. Click **Sync columns** to make sure that the input schema structure is correctly propagated from the preceding component.

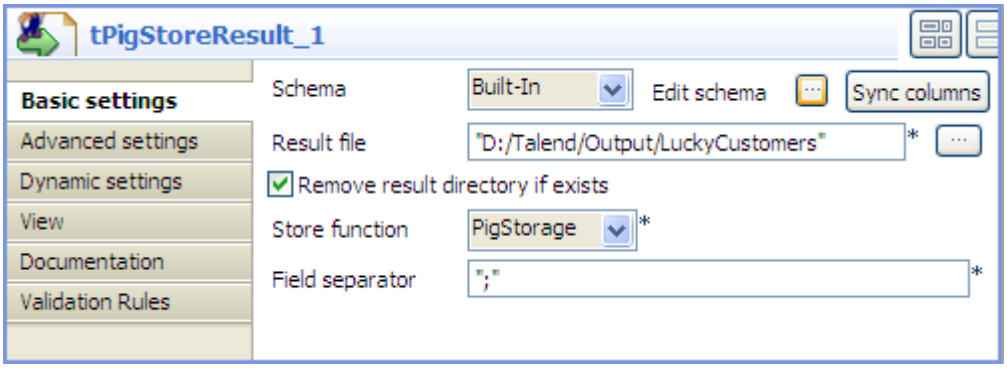
3. Select **Use advanced filter** and fill in the **Filter** field with filter expression:

```
"Country matches 'PuertoRico'"
```

   This filter expression selects rows of data that contains "PuertoRico" in the *Country* column.

Configuring the file output

1. Double-click **tPigStoreResult** to open its **Basic settings** view.



2. Click **Sync columns** to make sure that the input schema structure is correctly propagated from the preceding component.

3. Fill in the **Result file** field with the full path to the result file.

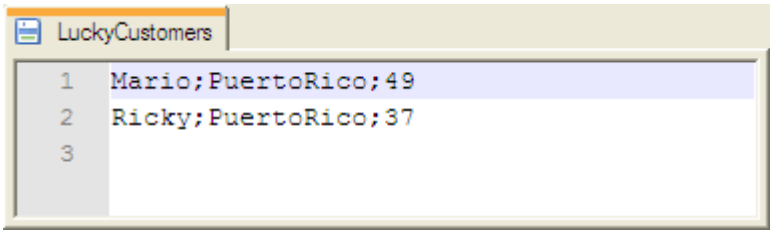
4. If the target file already exists, select the **Remove result directory if exists** check box.

5. Select **PigStorage** from the **Store function** list, and leave rest of the settings as they are.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click the **Run** button on the **Run** tab to run the Job.

   The result file contains the information of customers from the specified country.



tPigJoin

tPigJoin Properties

Component family	Big Data / Hadoop	
Function	This component allows you to perform join of two files based on join keys.	
Purpose	The **tPigJoin** component is used to perform inner joins and outer joins of two files based on join keys to create data that will be used by Pig.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. To make this component work, two schemas must be set: the schema of the main flow and the schema of the lookup flow. In the output part of the main schema, the columns of the main input file must be manually concatenated with those of the lookup file.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Filename	Fill in the path of the **Lookup file**.
	Join key	Click the plus button to add lines to set the **Join key** for **Input file** and **Lookup file**.
	Join mode	Select a join mode from the list: **inner-join**: Select this mode to perform an inner join of two or more relations based on Join keys. **left-outer-join**: Select this mode to performs a left outer join of two or more relations based on Join keys. **right-outer-join**: Select this mode to performs a right outer join of two or more relations based on Join keys. **full-outer-join**: Select this mode to combine the effect of applying both left and right outer joins. For further information about inner join and outer join, see: http://en.wikipedia.org/wiki/Join_%28SQL%29
Advanced settings	*Optimize the join*	Select this check box to optimize the performance of joins using **REPLICATED**, **SKEWED**, or **MERGE** joins. For further information about optimized joins, see: http://pig.apache.org/docs/r0.8.0/piglatin_ref1.html#Specialized+Joins
	Use partitioner	Select this check box to specify the Hadoop Partitioner that controls the partitioning of the keys of the intermediate map-outputs. For further information about the usage of Hadoop Partitioner, see: http://hadoop.apache.org/common/docs/r0.20.2/api/org/apache/hadoop/mapred/Partitioner.html
	Increase parallelism	Select this check box to set the number of reduce tasks for the **MapReduce** Jobs

	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig scripts is required.	

Scenario: Joining two files based on an exact match and saving the result to a local file

This scenario describes a four-component Job that combines data of an input file and a reference file that matches a given join key, removes unwanted columns, and then saves the final result to a local file.

The main input file contains the information about people's IDs, first names, last names, group IDs, and salaries, as shown below:

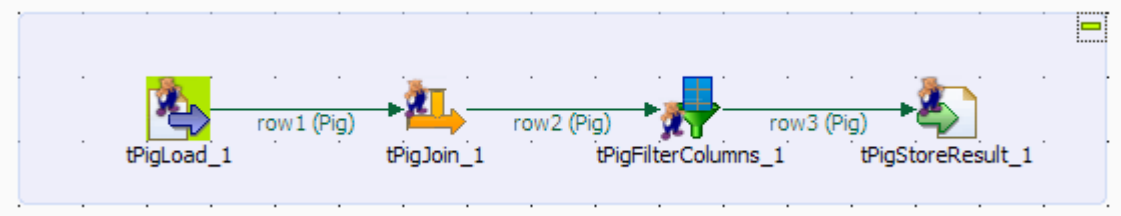
```
1;Woodrow;Johnson;3;1013.39
2;Millard;Monroe;2;8077.59
3;Calvin;Eisenhower;3;6866.88
4;Lyndon;Wilson;3;5726.28
5;Ronald;Garfield;2;4158.58
6;Rutherford;Buchanan;3;2897.00
7;Calvin;Coolidge;1;6650.66
8;Ulysses;Roosevelt;2;7854.78
9;Grover;Tyler;1;5226.88
10;Bill;Tyler;2;8964.66
```

The reference file contains only the information of group IDs and group names:

```
1;group_A
2;group_B
```

Dropping and linking the components

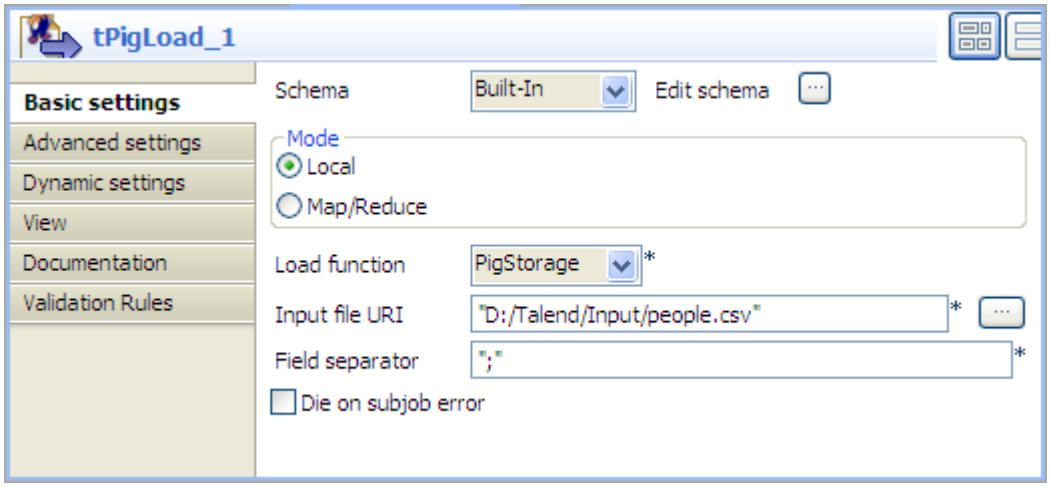
1. Drop the following components from the **Palette** to the design workspace: **tPigLoad**, **tPigJoin**, **tPigFilterColumns**, and **tPigStoreResult**.
2. Connect these components in a series using **Row > Pig Combine** connections.



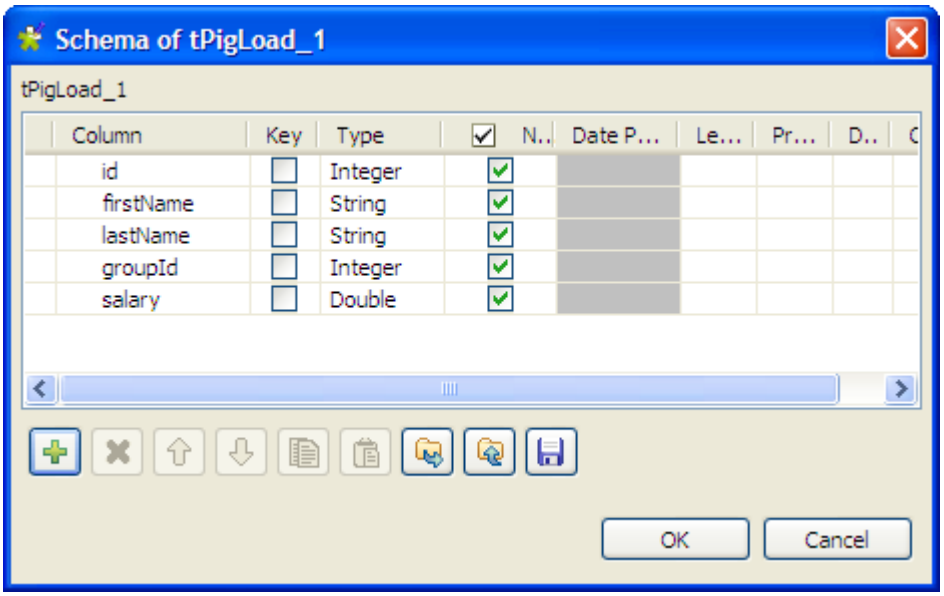
Configuring the components

Loading the main input file

1. Double-click **tPigLoad** to open its **Basic settings** view.



2. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box.



3. Click the **[+]** button to add columns, name them and define the column types according to the structure of the input file. In this example, the input schema has five columns: *id* (integer), *firstName* (string), *lastName* (string), *groupId* (integer), and *salary* (double).

   Then click **OK** to validate the setting and close the dialog box.

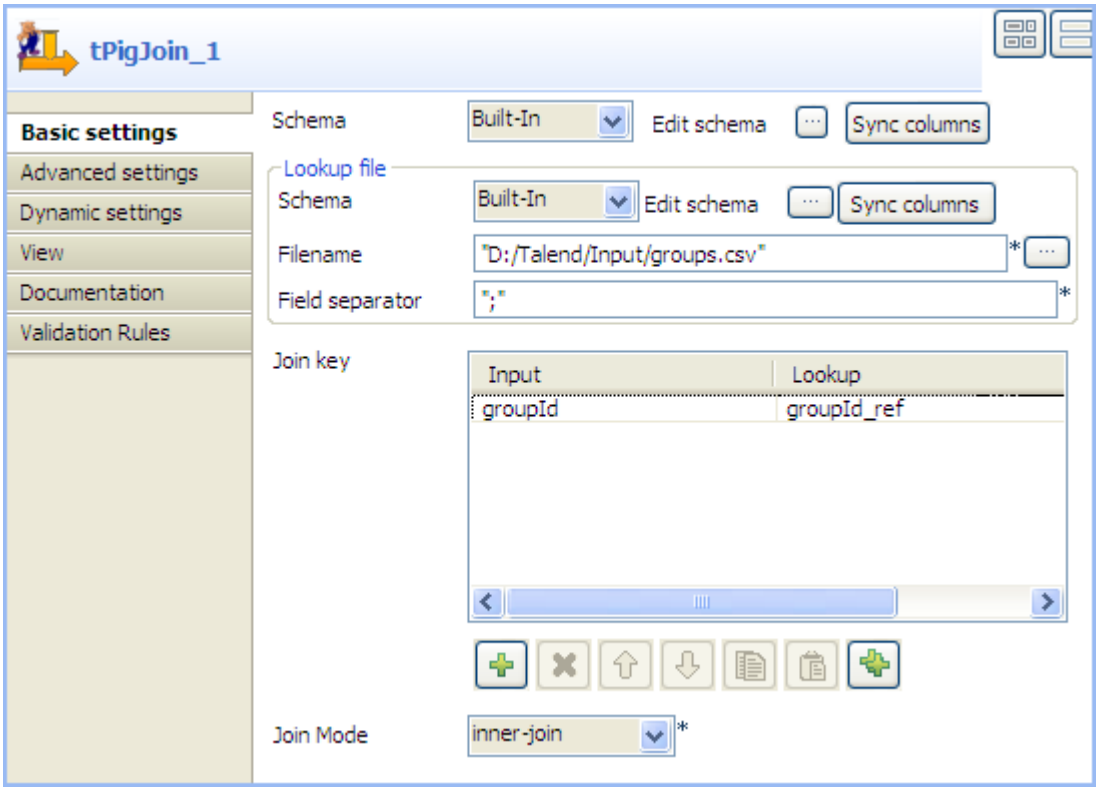
4. Click **Local** in the **Mode** area.

5. Select **PigStorage** from the **Load function** list.

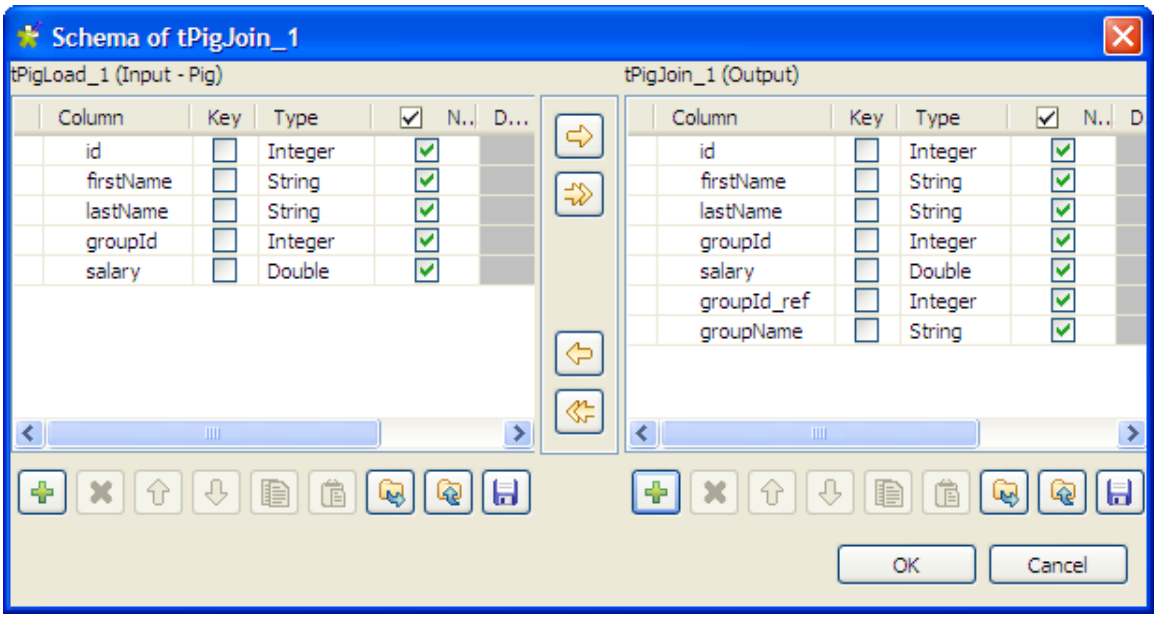
6. Fill in the **Input file URI** field with the full path to the input file, and leave the rest of the setting as they are.

Loading the reference file and setting up an inner join

1. Double-click **tPigJoin** to open its **Basic settings** view.



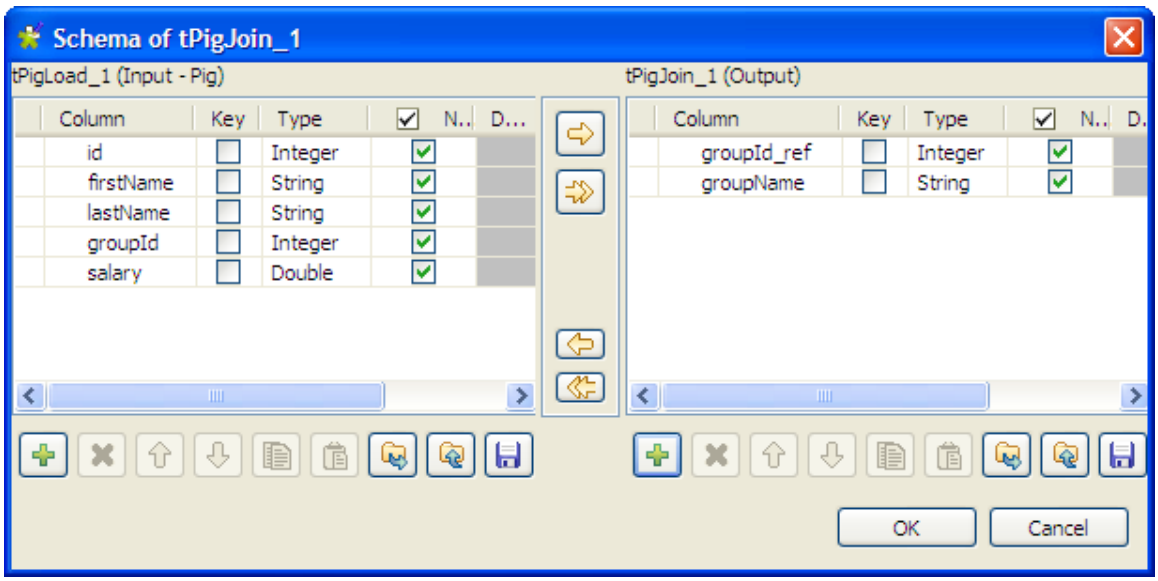
2. Click the **[...]** for the main schema to open the **[Schema]** dialog box.



3. Check that input schema is correctly retrieved from the preceding component. If needed, click the **[->>]** button to copy all the columns of the input schema to the output schema.

4. Click the **[+]** button under the output panel to add new columns according to the data structure of the reference file, *groupId_ref* (integer) and *groupName* (string) in this example. Then click **OK** to close the dialog box.

5. Click the **[...]** for the schema lookup flow to open the **[Schema]** dialog box.



6. Click the **[+]** button under the output panel to add two columns: *groupId_ref* (integer) and *groupName* (string), and then click **OK** to close the dialog box.

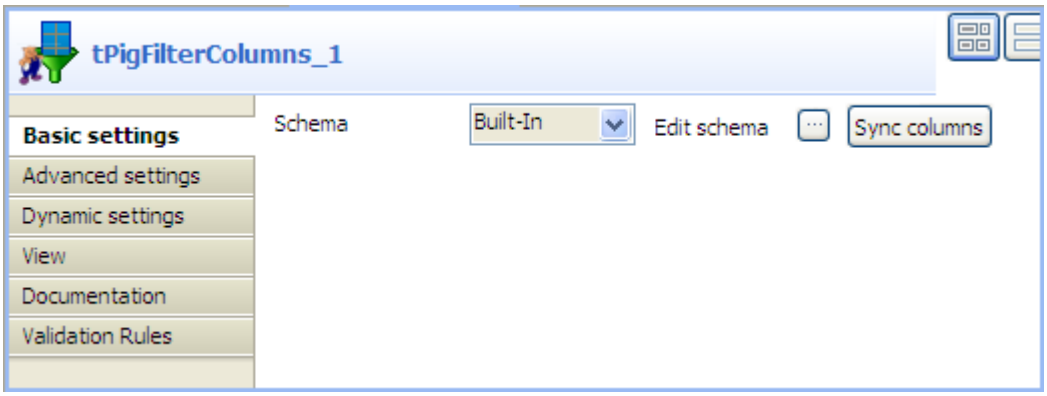
7. In the **Filename** field, specify the full path to the reference file.

8. Click the **[+]** button under the **Join key** table to add a new line, and select *groupId* and *groupId_ref* respectively from the **Input** and **Lookup** lists to match data from the main input flow with data from the lookup flow based on the group ID.

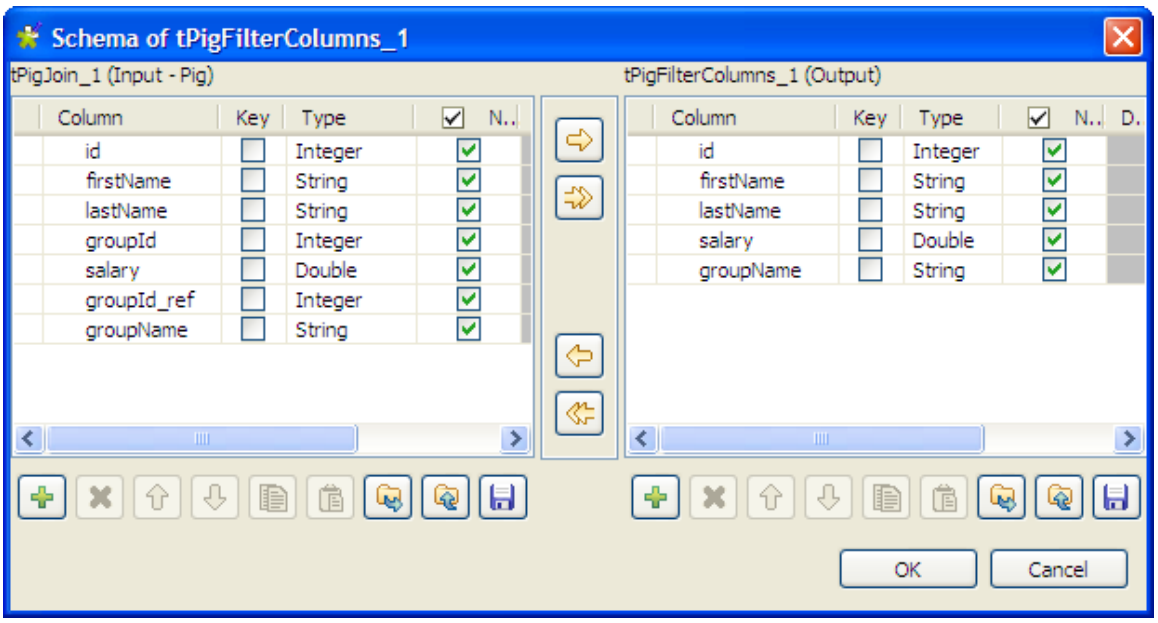
9. From the **Join Mode** list, select **inner-join**.

Defining the final output schema and the output file

1. Double-click **tPigFilterColumns** to open its **Basic settings** view.



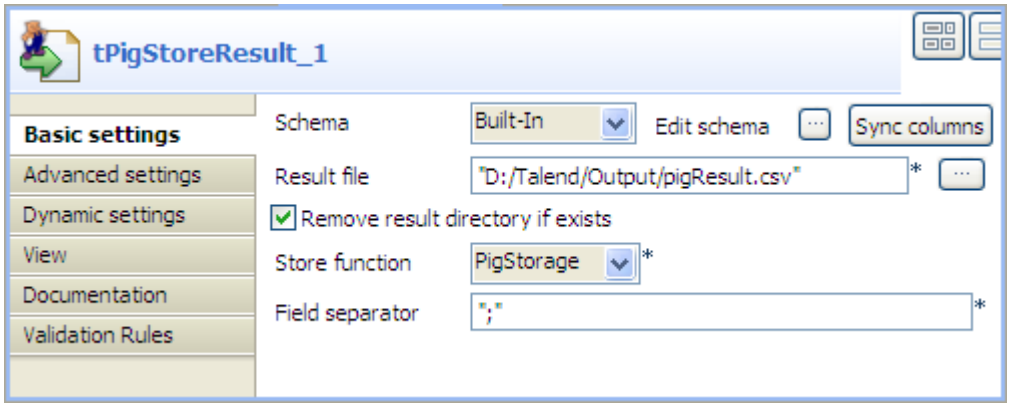
2. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box.



3. From the input schema, select the columns you want to include in your result file by clicking them one after another while pressing the **Shift** key, and click the **[->]** button to copy them to the output schema. Then, click **OK** to validate the schema setting and close the dialog box.

   In this example, we want the result file to include all the information except the group IDs.

4. Double-click **tPigStoreResult** to open its **Basic settings** view.



5. Click **Sync columns** to retrieve the schema structure from the preceding component.

6. Fill in the **Result file** field with the full path to the result file, and select the **Remove result file directory if exists** check box.

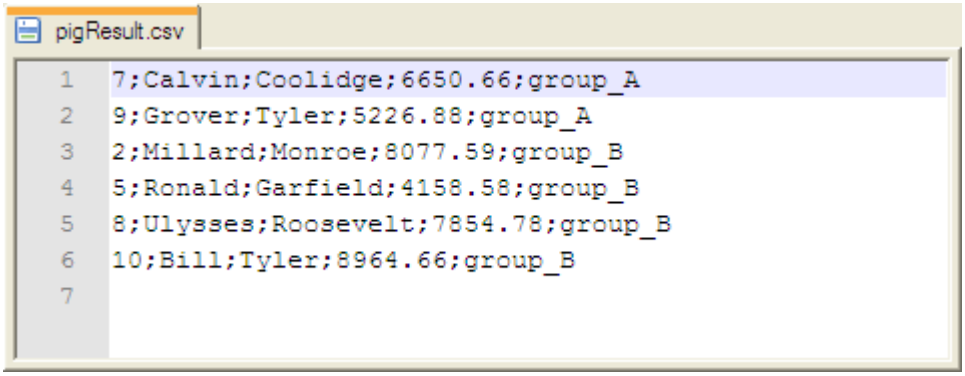
7. Select **PigStorage** from the **Store function** list, and leave rest of the settings as they are.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to run the Job.

   The result file includes all the information related to people of group A and group B, except their group IDs.


pigResult.csv
1 7;Calvin;Coolidge;6650.66;group_A
2 9;Grover;Tyler;5226.88;group_A
3 2;Millard;Monroe;8077.59;group_B
4 5;Ronald;Garfield;4158.58;group_B
5 8;Ulysses;Roosevelt;7854.78;group_B
6 10;Bill;Tyler;8964.66;group_B
7

tPigLoad

tPigLoad Properties

Component family	Big Data / Hadoop	
Function	This component allows you to set up a connection to the data source for a current transaction.	
Purpose	The **tPigLoad** component loads original input data to an output stream in just one single transaction, once the data has been validated.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Local	Click this radio button to run Pig scripts in **Local** mode. In this mode, all files are installed and run from your local host and file system.
	Map/Reduce	Click this radio button to run Pig scripts in **Map/Reduce** mode. Once selecting this mode, you need to complete the following fields: • **NameNode URI**: Type in the URI of the NameNode corresponding to the Map/Reduce version to be used. • **JobTracker host**: Type in the URI of the JobTracker corresponding to the Map/Reduce version to be used. • **HCat Metastore**: Type in the URI of the HCat metastore: "*thrift://<hcatalog server hostname>:9933*". For further information about the usage of HCat Metastore, see http://incubator.apache.org/hcatalog/docs/r0.2.0/loadstore.html.
	Load function	Select a load function for data to be loaded: **PigStorage**: Loads data in UTF-8 format. **BinStorage**: Loads data in machine-readable format. **TextLoader**: Loads unstructured data in UTF-8 format. **HCatLoader**: Loads data from HCataLog managed tables using Pig scripts. **HCatLoader** is available only when you select **Hortonworks Data Platform 1.0** from the **Version** list. For further information about the usage of HCatLoader, see http://incubator.apache.org/hcatalog/docs/r0.2.0/loadstore.html#HCatLoader.
	Input file URI	Fill in this field with the full local path to the input file. This field is not available when you select **HCatLoader** from the **Load function** list.

	HCataLog Configuration	Fill the following fields to configure HCataLog managed tables on HDFS(Hadoop distributed file system): **Database**: The database in which tables are placed. **Table**: The table in which data is stored. **Partition filter**: Fill this field with the partition keys to list partitions by filter. **Register jar**: Fill the **Jar path** field with the path to the jar files listed in the **Column** field. **HCataLog Configuration area** is enabled only when you select **HCatLoader** from the **Load function** list. For further information about the usage of HCataLog, see http://incubator.apache.org/hcatalog/docs/r0.2.0/. For further information about the usage of **Partition filter**, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.
	Field separator	Enter the separator used to identify the end of a field. This field is enabled only when you select **PigStorage** from the **Load function** list.
	Die on subjob error	This check box is cleared by default, meaning to skip the row on subjob error and to complete the process for error-free rows.
Advanced settings	*Hadoop Properties*	Complete this table using the customized Hadoop parameters, if any, to override the default ones.
	Properties	Complete this table using the customized Pig set configuration, if any, to override the default one. For example, the *default_parallel* key used in Pig could be set as *20*.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is always used to start a Pig chain.	
Limitation	Knowledge of Pig scripts is required. If you select HCatLoader as the load function, knowledge of HCataLog DDL(HCataLog Data Definition Language, a subset of Hive Data Definition Language) is required. For further information about HCataLog DDL, see http://incubator.apache.org/hcatalog/docs/r0.2.0/cli.html.	

Related Scenario

For further infomation regarding the **tPigLoad** component in use, see section *Scenario: Filtering rows of data based on a condition and saving the result to a local file* of **tPigFilterRow**.

tPigSort

tPigSort Properties

Component family	Big Data / Hadoop	
Function	This component allows you to sort a relation based on one or more defined sort keys.	
Purpose	The **tPigSort** component is used to sort relation based on one or more defined sort keys.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sort key	Click the **Add** button beneath the **Sort key** table to add one or more lines to specify column and sorting order for each sort key.
Advanced settings	*Increase parallelism*	Select this check box to set the number of reduce tasks for the **MapReduce** Jobs
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is commonly used as intermediate step together with input component and output component.	
Limitation	Knowledge of Pig scripts is required.	

Scenario: Sorting data in ascending order

This scenario describes a three-component Job that sorts rows of data based on one or more sorting conditions and stores the result into a local file.

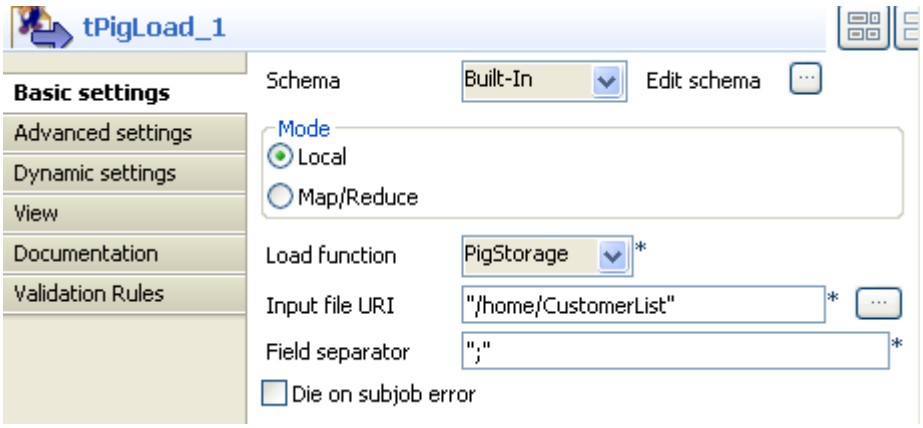
Setting up the Job

1. Drop the following components from the **Palette** to the design workspace: **tPigSort**, **tPigLoad**, **tPigStoreResult**.
2. Connect **tPigLoad** to **tPigFilterRow** using a **Row > Pig Combine** connection.

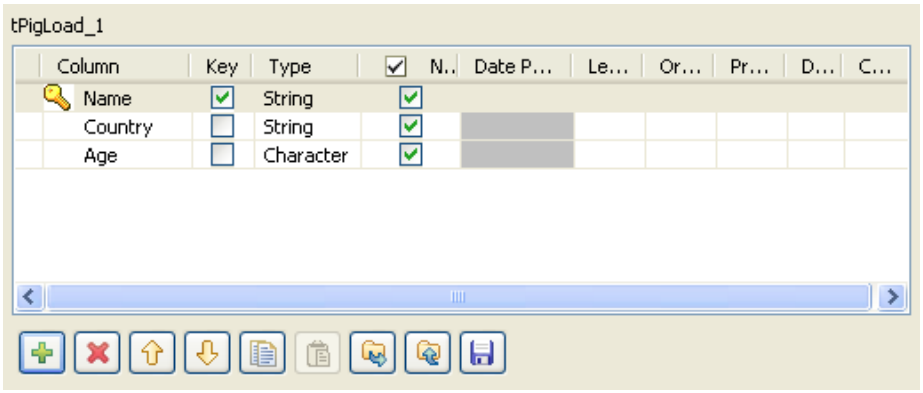
3. Connect **tPigFilterRow** to **tPigStoreResult** using a **Row > Pig Combine** connection.

Loading the data

1. Double-click **tPigLoad** to open its **Basic settings** view.



2. Click the **[...]** button next to **Edit schema** to add columns for **tPigLoad**.



3. Click the **[+]** button to add *Name*, *Country* and *Age* and click **OK** to save the setting.

4. Select **Local** from the **Mode** area.

5. Fill in the **Input filename** field with the full path to the input file.

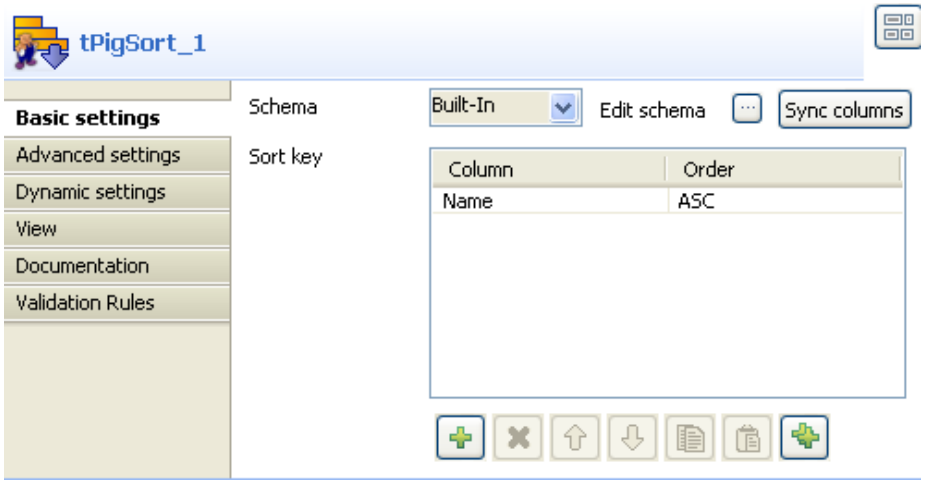
   In this scenario, the input file is *CustomerList* that contains rows of names, country names and age.

6. Select **PigStorage** from the **Load function** list.

7. Leave rest of the settings as they are.

Setting the sorting condition

1. Double-click **tPigSort** to open its **Basic settings** view.

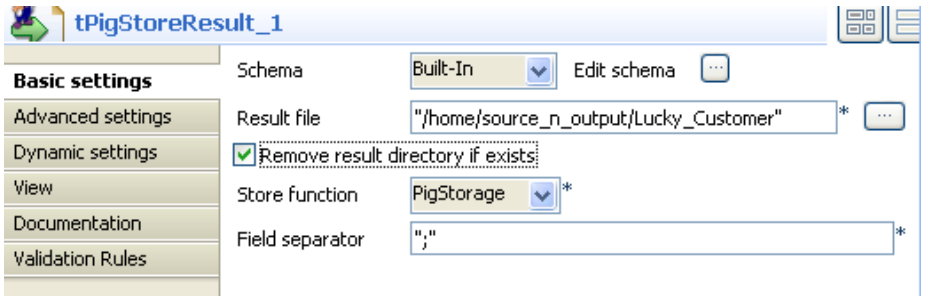


2. Click **Sync columns** to retrieve the schema structure from the preceding component.
3. Click the **[+]** button beneath the **Sort key** table to add a new sort key. Select *Age* from the **Column** list and select **ASC** from the **Order** list.

   This sort key will sort the data in *CustomerList* in ascending order based on *Age*.

Saving the data to a local file

1. Double-click **tPigStoreResult** to open its **Basic settings** view.



2. Click **Sync columns** to retrieve the schema structure from the preceding component.
3. Select **Remove result directory if exists**.
4. Fill in the **Result file** field with the full path to the result file.

   In this scenario, the result of filter is saved in *Lucky_Customer* file.
5. Select **PigStorage** from the **Store function** list.
6. Leave rest of the settings as they are.

Executing the Job

Save your Job and press **F6** to run it.

The *Lucky_Customer* file is generated containing the data in ascending order based on *Age*.

tPigStoreResult

tPigStoreResult Properties

Component family	Big Data / Hadoop	
Function	This component allows you to store the result of your Pig Job into a defined data storage space.	
Purpose	The **tPigStoreResult** component is used to store the result into defined data storage space.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Result file	Select the path to the result file in which data is stored.
	Remove result directory if exists	Select this check box to remove an existing result directory. This check box is disabled when you select **HCatStorer** from the **Store function** list.
	Store function	Select a store function for data to be stored: **PigStorage**: Stores data in UTF-8 format. **BinStorage**: Stores data in machine-readable format. **PigDump**: Stores data as tuples in human-readable UTF-8 format. **HCatStorer**: Stores data in HCatalog managed tables using Pig scripts.
	HCataLog Configuration	Fill the following fields to configure HCataLog managed tables on HDFS(Hadoop distributed file system): **Database**: The database in which tables are placed. **Table**: The table in which data is stored. **Partition filter**: Fill this field with the partition keys to list partitions by filter. **Register jar**: Fill the **Jar path** field with the path to the jar files listed in the **Column** field. **HCataLog Configuration area** is enabled only when you select **HCatStorer** from the **Store function** list. For further information about the usage of HCataLog, see http://incubator.apache.org/hcatalog/docs/r0.2.0/. For further information about the usage of **Partition filter**, see https://cwiki.apache.org/confluence/display/HCATALOG/Design+Document+-+Java+APIs+for+HCatalog+DDL+Commands.
	Field separator	Enter the separator used to identify the end of a field. This field is enabled only when you select **PigStorage** from the **Store function** list.

Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is always used to end a Pig chain.	
Limitation	Knowledge of Pig scripts is required. If you select **HCatStorer** as the store function, knowledge of HCataLog DDL(HCataLog Data Definition Language, a subset of Hive Data Definition Language) is required. For further information about HCataLog DDL, see http://incubator.apache.org/hcatalog/docs/r0.2.0/cli.html.	

Related Scenario

For more infomation regarding the **tPigStoreResult** component in use, see section *Scenario: Sorting data in ascending order* of **tPigSort**.

tSqoopExport

tSqoopExport Properties

Component family	Big Data / Sqoop	
Function	**tSqoopExport** calls sqoop to transfer data from the Hadoop Distributed File System (HDFS) to a relational database management system (RDBMS). Ensure that you have Sqoop installed in the host used to run this component and ensure as well that you have added the Sqoop command line to the PATH variable of this host.	
Purpose	**tSqoopExport** is used to define the arguments required by Sqoop for transferring data to a RDBMS.	
Basic settings	*Connection*	Type in the JDBC URL used to connect to the target database.
	Table Name	Type in the name of the target table to which data is transferred from HDFS. This table must already exist in the target database. The input files are read and parsed into a set of records according to the user-specified delimiters.
	Export Dir	Enter the path to the source data to be transferred in HDFS.
	User name and *Password*	Type in the authentication information to the target database.
	Specify Number of Mappers	Select this check box to indicate the number of map tasks (parallel processes) used to perform the data transfer. If you do not want Sqoop to work in parallel, enter *1* in the displayed field.
	Print Log	Select this check box to activate the **Verbose** check box.
	Verbose	Select this check box to print more information while working, for example, the debugging information.
Advanced settings	*Direct*	Select this check box to use the export fast path.
	Use MySQL default delimiters	Select this check box to use MySQL's default delimiter set.
	Additional arguments	Complete this table to use an additional argument if you need one. At each time, only one additional argument is allowed. Use of multiple additional arguments at the same time may lead to exception error.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used standalone. It respects the Sqoop prerequisites. You need necessary knowledge about Sqoop to use it. We recommend using the Sqoop of version 1.4+ in order to benefit the full functions of these components. For further information about Sqoop, see the Sqoop manual on: http://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html	
Limitation	You need to take the host, where Sqoop is installed, to run the Job using this component. If the Studio is not deployed on this host, you can install the Job server provided by **Talend** on the host and use the server to run this Job.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error.

		Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tSqoopImport

tSqoopImport Properties

Component family	Big Data / Sqoop	
Function	**tSqoopImport** calls Sqoop to transfer data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS). Ensure that you have Sqoop installed in the host used to run this component and ensure as well that you have added the Sqoop command line to the PATH variable of this host.	
Purpose	**tSqoopImport** is used to define the arguments required by Sqoop for writing the data of your interest into HDFS.	
Basic settings	*Connection*	Enter the JDBC URL used to connect to the database where the source data is stored.
	User name and *Password*	Enter the authentication information used to connect to the source database.
	Table Name	Type in the name of the table to be transferred into HDFS. This field is not available when you are using the free-form query mode by selecting the **Use query** check box.
	Append	Select this check box to append transferred data to an existing dataset in HDFS.
	File format	Select a file format for the data to be transferred. By default, the file format is **textfile**, but you can as well choose the **sequencefile** format instead.
	Compress	Select this check box to enable compression.
	Print Log	Select this check box to activate the **Verbose** check box.
	Verbose	Select this check box to print more information while working, for example, the debugging information.
Advanced settings	*Direct*	Select this check box to use the import fast path.
	Specify columns	Select this check box to display the column table where you can specify the columns you want to transfer into HDFS.
	Use WHERE clause	Select this check box to use a WHERE clause that controls the rows to be transferred. In the field displayed, you can type in the condition used to select the rows you want. For example, type in `id >400` to import only the rows where the *id* column has a value greater than 400.
	Use MySQL default delimiters	Select this check box to use MySQL's default delimiter set.
	Use query	Select this check box to use the free-form query mode provided by Sqoop. Once selecting it, you are able to enter the free-form query you need to use. Then, you must specify the target directory and if the Sqoop imports data in parallel, specify as well the `Split by` argument.
	Specify Target Dir	Select this check box to enter the path to the target location, in HDFS, where you want to transfer the source data to. This location should be a new directory; otherwise, you must select the **Append** check box.

	Specify Split by	Select this check box, then, enter the table column you want to use as the splitting column to split the workload. For example, enter *tablename.id*. Then Sqoop will split the data to be transferred according to their ID values and imports them in parallel.
	Specify Number of Mappers	Select this check box to indicate the number of map tasks (parallel processes) used to perform the data transfer. If you do not want Sqoop to work in parallel, enter *1* in the displayed field.
	Additional arguments	Complete this table to use an additional argument if you need one. At each time, only one additional argument is allowed. Use of multiple additional arguments at the same time may lead to exception error.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used standalone. It respects the Sqoop prerequisites. You need necessary knowledge about Sqoop to use it. We recommend using the Sqoop of version 1.4+ in order to benefit the full functions of these components. For further information about Sqoop, see the Sqoop manual on: http://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html	
Limitation	You need to take the host, where Sqoop is installed, to run the Job using this component. If the Studio is not deployed on this host, you can install the Job server provided by **Talend** on the host and use the server to run this Job.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tSqoopImportAllTables

tSqoopImportAllTables Properties

Component family	Big Data / Sqoop	
Function	**tSqoopImportAllTables** calls Sqoop to transferT all of the tables of a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS). Ensure that you have Sqoop installed in the host used to run this component and ensure as well that you have added the Sqoop command line to the PATH variable of this host.	
Purpose	**tSqoopImportAllTables** is used to define the arguments required by Sqoop for writing all of the tables of a database into HDFS.	
Basic settings	*Connection*	Enter the JDBC URL used to connect to the database where the source data is stored.
	User name and *Password*	Enter the authentication information used to connect to the source database.
	File format	Select a file format for the data to be transferred. By default, the file format is **textfile**, but you can as well choose the **sequencefile** format instead.
	Specify Number of Mappers	Select this check box to indicate the number of map tasks (parallel processes) used to perform the data transfer. If you do not want Sqoop to work in parallel, enter *1* in the displayed field.
	Compress	Select this check box to enable compression.
	Print Log	Select this check box to activate the **Verbose** check box.
	Verbose	Select this check box to print more information while working, for example, the debugging information.
Advanced settings	*Direct*	Select this check box to use the import fast path.
	Use MySQL default delimiters	Select this check box to use MySQL's default delimiter set.
	Additional arguments	Complete this table to use an additional argument if you need one. At each time, only one additional argument is allowed. Use of multiple additional arguments at the same time may lead to exception error.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used standalone. It respects the Sqoop prerequisites. You need necessary knowledge about Sqoop to use it. We recommend using the Sqoop of version 1.4+ in order to benefit the full functions of these components. For further information about Sqoop, see the Sqoop manual on: http://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html	
Limitation	You need to take the host, where Sqoop is installed, to run the Job using this component. If the Studio is not deployed on this host, you can install the Job server provided by **Talend** on the host and use the server to run this Job.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error.

		Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

Business components

This chapter details the major components that you can find in **Business** group of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Business component family groups connectors that covers specific Business needs, such as reading and writing CRM, or ERP types of database and reading from or writing to an SAP system.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAlfrescoOutput

tAlfrescoOutput Properties

Component family	Business	
Function	Creates dematerialized documents in an Alfresco server where they are indexed under meaningful models.	
Purpose	Allows to create and manage documents in an Alfresco server.	
Basic settings	*URL*	Type in the URL to connect to the Alfresco Web application.
	Login and Password	Type in the user authentication data to the Alfresco server.
	Base	Type in the base path where to put the document, or Select the **Map...** check box and then in the **Column** list, select the target location column. **Note:** When you type in the base name, make sure to use the double backslash (\\) escape character.
	Document Mode	Select in the list the mode you want to use for the created document. **Create only**: creates a document if it does not exist. Note that an error message will display if you try to create a document that already exists **Create or update**: creates a document if it does not exist or updates the document if it exists.
	Container Mode	Select in the list the mode you want to use for the destination folder in Alfresco. **Update only**: updates a destination folder if the folder exists. Note that an error message will display if you try to update a document that does not exist **Create or update**: creates a destination folder if it does not exist or updates the destination folder if it exists.
	Define Document Type	Click the three-dot button to display the **tAlfrescoOutput** editor. This editor enables you to: - select the file where you defined the metadata according to which you want to save the document in Alfresco -define the type f the document -select any of the aspects in the **available aspects** list of the model file and click the plus button to add it in the list to the left.
	Property Mapping	Displays the parameters you set in the **tAlfrescoOutput** editor and according to which the document will be created in the Alfresco server. Note that in the **Property Mapping** area, you can modify any of the input schemas.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or remote in the **Repository**.

		Click **Edit Schema** to make changes to the schema. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Result Log File Name	Browse to the file where you want to save any logs related to the Job execution.
Advanced settings	*Configure Target Location Container*	Allows to configure the (by default) type of containers (folders) Select this check box to display new fields where you can modify the container type to use your own created types based on the father/child model.
	Configure Permissions	When selected, allows to manually configure access rights to containers and documents. Select the **Inherit Permissions** check box to synchronize access rights between containers and documents. Click the **Plus** button to add new lines to the **Permissions** list, then you can assign roles to user or group columns.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory.
	Association Target Mapping	Allows to create new documents in Alfresco with associated links towards other documents already existing in Alfresco, to facilitate the navigation process for example. To create associations: 1. Open the **tAlfresco** editor. 2. Click the **Add** button and select a model where you have already defined aspects that contain associations. 3. Click the drop-down arrow at the top of the editor and select the corresponding document type. 4. Click **OK** to close the editor and display the created association in the **Association Target Mapping** list.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as an output component. An input component is required.	
Limitation/Prerequisites	To be able to use the **tAlfrescoOutput** component, few relevant resources need to be installed: check the *Installation Procedure* sub section for more information. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Installation procedure

To be able to use **tAlfrescoOutput** in the **Integration** perspective of the **Talend** Studio, you need first to install the Alfresco server with few relevant resources.

The below sub sections detail the prerequisite and the installation procedure.

Prerequisites

Start with the following operations:

1. Download the file `alfresco-community-tomcat-2.1.0.zip`
2. Unzip the file in an installation folder, for example:

```
C:\Program Files\Java\jdk1.6.0_27
```

3. Install JDK 1.6.0+
4. Update the environment variable

```
JAVA_HOME (JAVA_HOME= C:\alfresco)
```

5. From the installation folder (`C:\alfresco`), launch the alfresco server using the script `alf_start.bat`

*Make sure that the Alfresco server is launched correctly before start using the **tAlfrescoOutput** component.*

Installing the Talend Alfresco module

Note that the `talendalfresco_20081014.zip` is provided with the **tAlfrescoOutput** component in the **Integration** perspective of the **Talend** Studio.

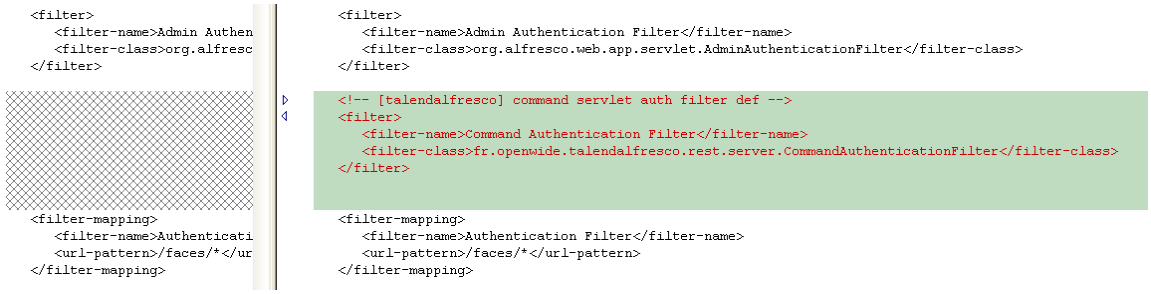
To install the `talendalfresco` module:

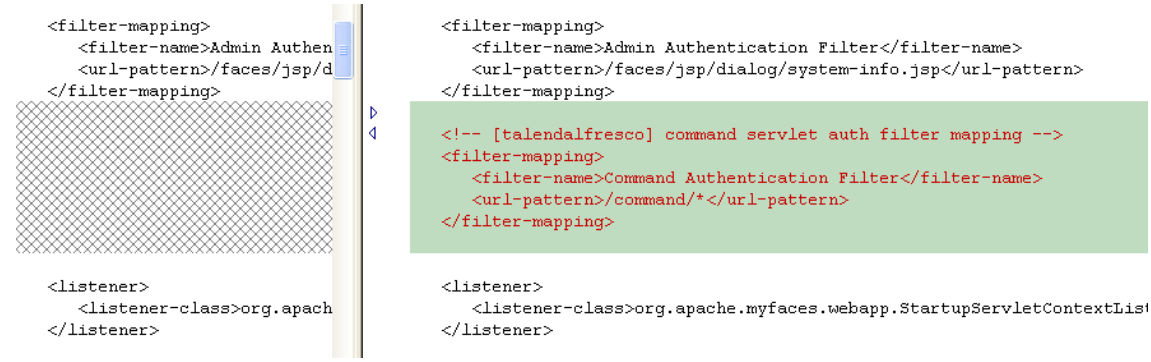
1. From `talendalfresco_20081014.zip` and in the `talendalfresco_20081014\alfresco` folder, look for the following jars: `stax-api-1.0.1.jar, wstx-lgpl-3.2.7.jar`, `talendalfresco-client_1.0.jar`, and `talendalfresco-alfresco_1.0.jar` and move them to `C:\alfresco\tomcat\webapps\alfresco\WEB-INF\lib`
2. Add the authentification filter of the commands to the `web.xml` file located in the path

```
C:\alfresco\tomcat\webapps\alfresco\WEB-INF
son WEB-INF/
```

following the model of the example provided in `talendalfresco_20081014/alfresco` folder of the zipped file `talendalfresco_20081014.zip`

The following figures show the portion of lines (in blue) to add in the file `web.xml alfresco.`





Useful information for advanced use

Installing new types for Alfresco:

From the `package_jeu_test.zip` and in the `package_jeu_test/fichiers_conf_alfresco2.1` folder, look for the following files: `xml H76ModelCustom.xml` (description of the model), `web-client-config-custom.xml` (web interface of the model), and `custom-model-context.xml` (registration of the new model) and paste them in the following folder: `C:/alfresco/tomcat/shared/classes/alfresco/extension`

Dates:

- The dates must be of the **Talend** date type `java.util.Date`.
- Columns without either mapping or default values, for example of the type `Date`, are written as empty strings.
- Solution: delete all columns without mapping or default values. Note that any modification of the type Alfresco will put them back.

Content:

- Do not mix up between the file path which content you want to create in Alfresco and its target location in Alfresco.
- Provide a URL! It can target various protocols, among which are file, HTTP and so on.
- For URLs referring to files on the file system, precede them by "file:" for Windows used locally, and by "file://" for Windows on a network (which accepts as well "file: \ \") or for Linux.
- Do not double the backslash in the target base path (automatic escape), unless you type in the path in the basic settings of the **tAlfrescoOutput** component, or doing concatenation in the **tMap** editor for example.

Multiple properties or associations:

- It is possible to create only one association by document if it is mapped to a string value, or one or more associations by document if it is mapped to a list value (object).
- You can empty an association by mapping it to an empty list, which you can create, for example, by using `new java.util.ArrayList()`in the **tMap** component.

However, it is impossible to delete an association.

Building `List(object)`with **tAggregate**:

- define the table of the relation n-n in a file, containing a `name` line for example (included in the input rows), and a `category` line (that can be defined with its mapping in a third file).
- group by: input name, output name.
- operation: output `categoryList`, function `list(object)`, input `category`. ATTENTION list (object) and non simple list.

- References (documents and folders):

- References are created by mapping one or more existing reference nodes (xpath or namepath) using `String` type or `List(object)`.
- An error in the association or the property of the reference type does not prevent the creation of the node that holds the reference.

- Properties of the reference type are created in the **Basic Settings** view.
- Associations are created in the **Advanced Settings** view.

Dematerialization, tAlfrescoOutput, and Enterprise Content Management

Dematerialization is the process that convert documents held in physical form into electronic form, and thus helps to move away from the use of physical documentation to the use of electronic Enterprise Content Management (ECM) systems. The range of documents that can be managed with an Enterprise Content Management system include just about everything from basic documents to stock certificates, for example.

Enterprises dematerialize their content via a manual document handling, done by man, or an automatic document handling, machine-based.

Considering the varied nature of the content to be dematerialized, enterprises have to use varied technologies to do it. Scanning paper documents, creating interfaces to capture electronic documents from other applications, converting document images into machine-readable/editable text documents, and so on are examples of the technologies available.

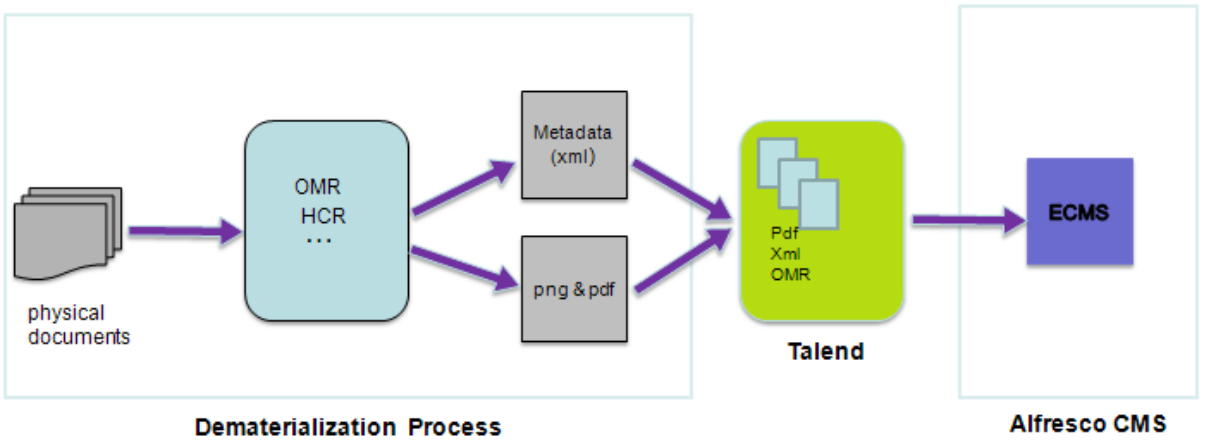
Furthermore, scanned documents and digital faxes are not readable texts. To convert them into machine-readable characters, different character recognition technologies are used. Handwritten Character Recognition (HCR) and Optical Mark Recognition (OMR) are two examples of such technologies.

Equally important as the content that is captured in various formats from numerous sources in the dematerialization process is the supporting metadata that allows efficient identification of the content via specific queries.

Now how can this document content along with the related metadata be aggregated and indexed in an Enterprise Content Management system so that it can be retrieved and managed in meaningful ways? **Talend** provides the answer through the **tAlfrescoOutput** component.

The **tAlfrescoOutput** component allows you to stock and manage your electronic documents and the related metadata on the Alfresco server, the leading open source enterprise content management system.

The following figure illustrates **Talend**'s role between the dematerialization process and the Enterprise Content Management system (Alfresco).



ECMS: Enterprise Content Management System
HCR: Handwritten Character Recognition
OMR: Optical Mark Recognition

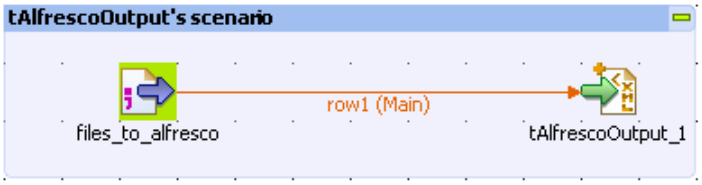
Scenario: Creating documents on an Alfresco server

This Java scenario describes a two-component Job which aims at creating two document files with the related metadata in an Alfresco server, the java-based Enterprise Control Management system.

Setting up your Job

1. Drop the **tFileInputDelimited** and **tAlfrescoOutput** components from the **Palette** onto the design workspace.

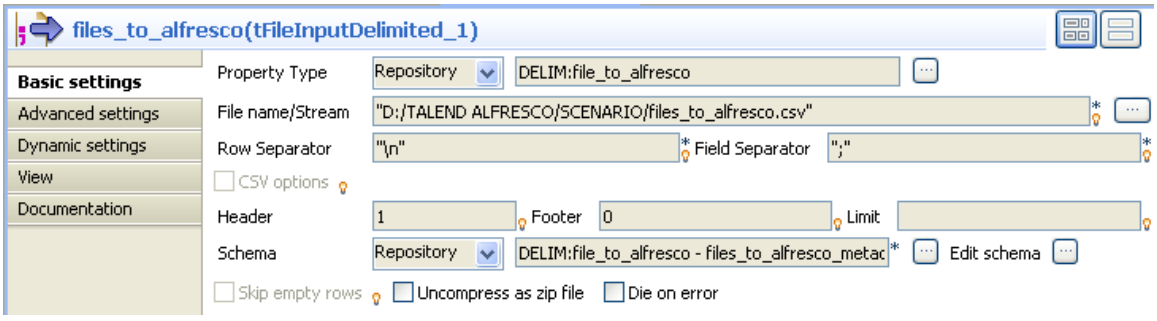
2. Connect the two components together using a **Main** > **Row** connection.



Setting up the schema

1. In the design workspace, double-click **tFileInputDelimited** to display its basic settings**.**

2. Set the **File Name** path and all related properties.

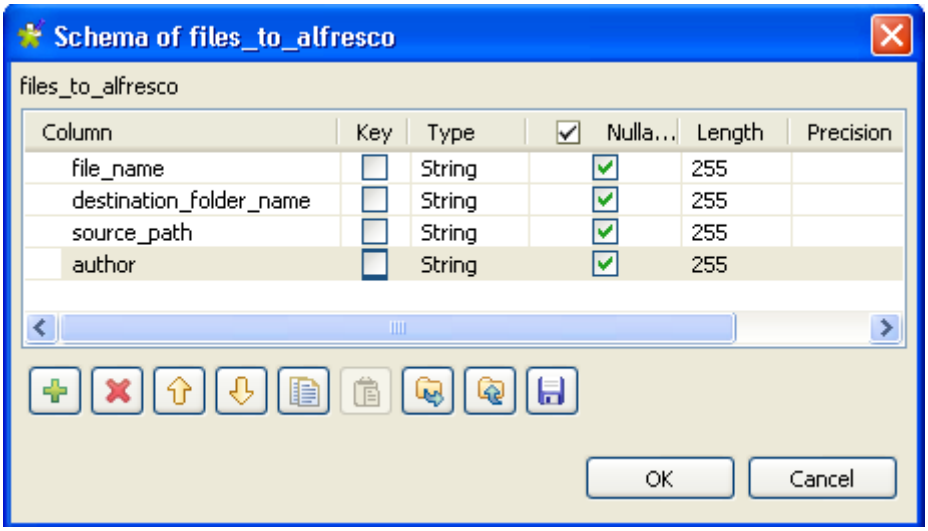


*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

In this scenario, the delimited file provides the metadata and path of two documents we want to create in the Alfresco server. The input schema for the documents consists of four columns: *file_name*, *destination_folder name*, *source_path*, and *author*.

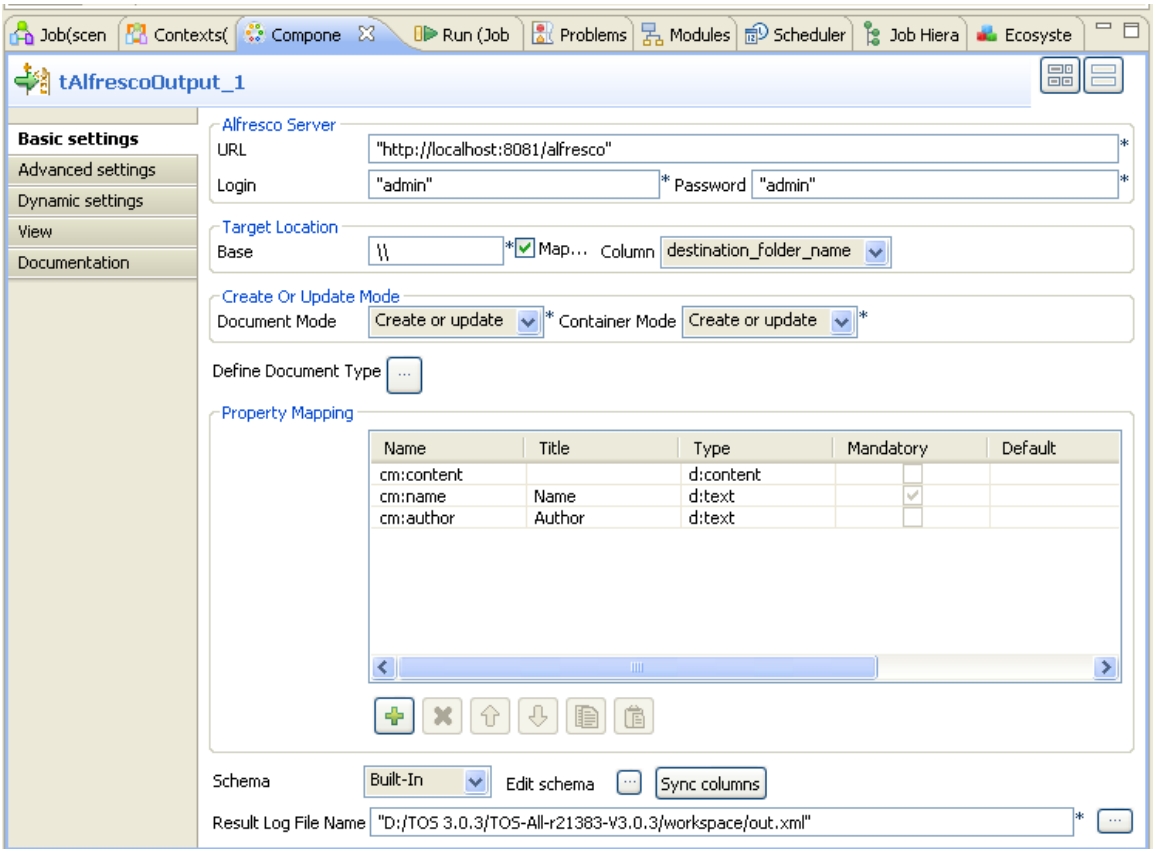
	A	B	C	D	E	F
1	file_name;destination_folder_name;source_path;author					
2	PO_43278.pdf;FINANCE\PO;file:D:/PO/customer1_2009-03-24.pdf;talend					
3	PO_43279.pdf;FINANCE\PO;file:D:/PO/customer2_2009-03-24.pdf;talend					

And therefore the input schema of the delimited file will be as the following:



Setting up the connection to the Alfresco server

1. In the design workspace, double-click **tAlfrescoOutput** to display its basic settings**.**



2. In the **Alfresco Server** area, enter the Alfresco server URL and user authentication information in the corresponding fields.

3. In the **TargetLocation** area, either type in the base name where to put the document in the server, or Select the **Map...** check box and then in the **Column** list, select the target location column, `destination_folder_name` in this scenario.

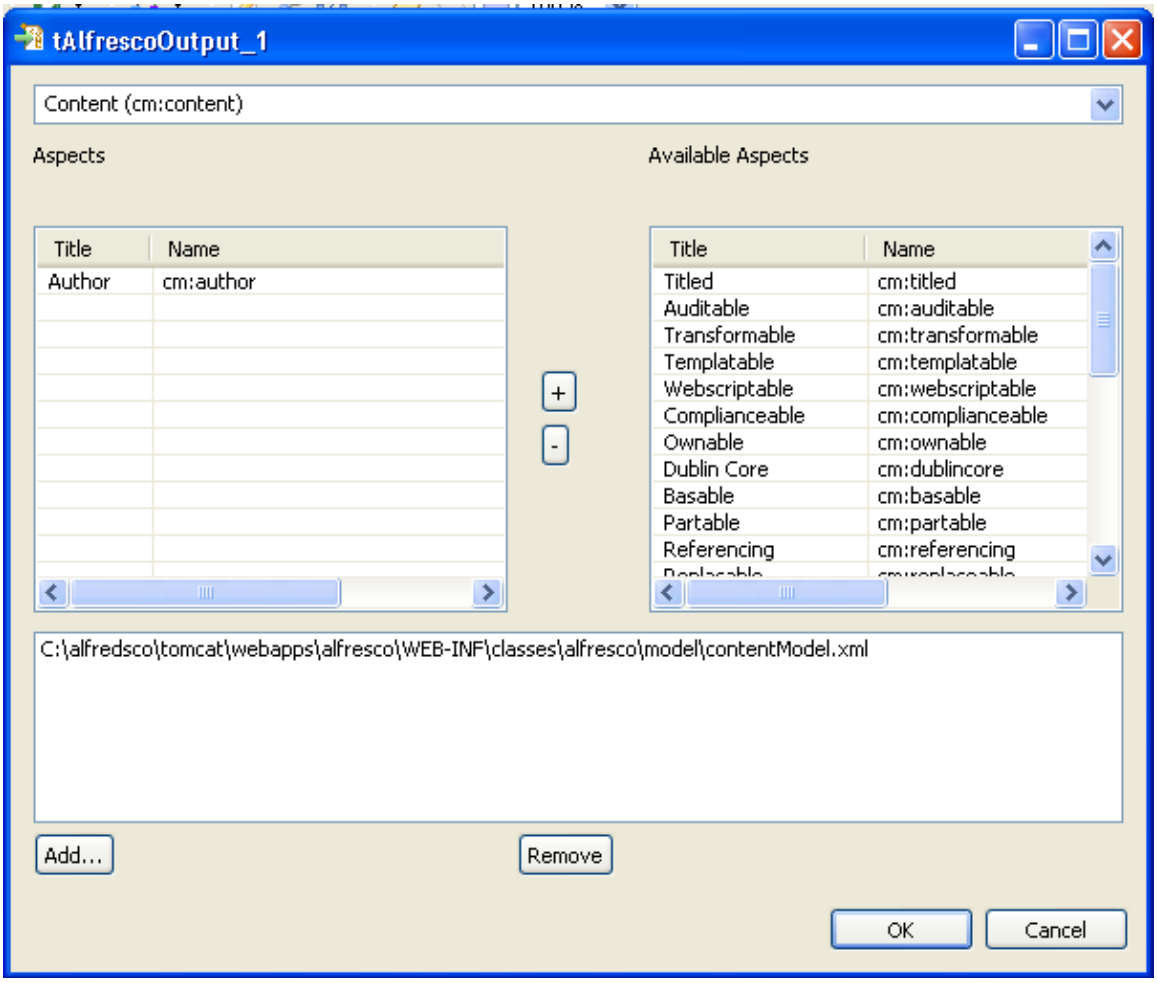
When you type in the base name, make sure to use the double backslash (\\) escape character.

4. In the **Document Mode** list, select the mode you want to use for the created documents.

5. In the **Container Mode** list, select the mode you want to use for the destination folder in Alfresco.

Defining the document

1. Click the **Define Document Type** three-dot button to open the **tAlfrescoOutput** editor.



2. Click the **Add** button to browse and select the xml file that holds the metadata according to which you want to save the documents in Alfresco.

   All available aspects in the selected model file display in the **Available Aspects** list.

   You can browse for this model folder locally or on the network. After defining the aspects to use for the document to be created in Alfresco, this model folder is not needed any more.

3. If needed, select in the **Available Aspects** list the aspect(s) to be included in the metadata to write in the Alfresco server. In this scenario we want the author name to be part of the metadata registered in Alfresco.

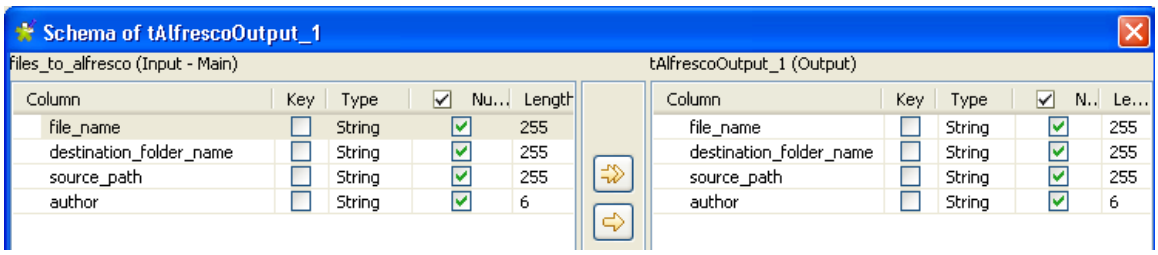
4. Click the drop-down arrow at the top of the editor to select from the list the type to give to the created document in Alfresco, `Content` in this scenario.

All the defined aspects used to select the metadata to write in the Alfresco server display in the **Property Mapping** list in the **Basic Settings** view of **tAlfrescoOutput**, three aspects in this scenario, two basic for the `Content` type (`content` and `name`) and an additional one (`author`).

Executing your Job

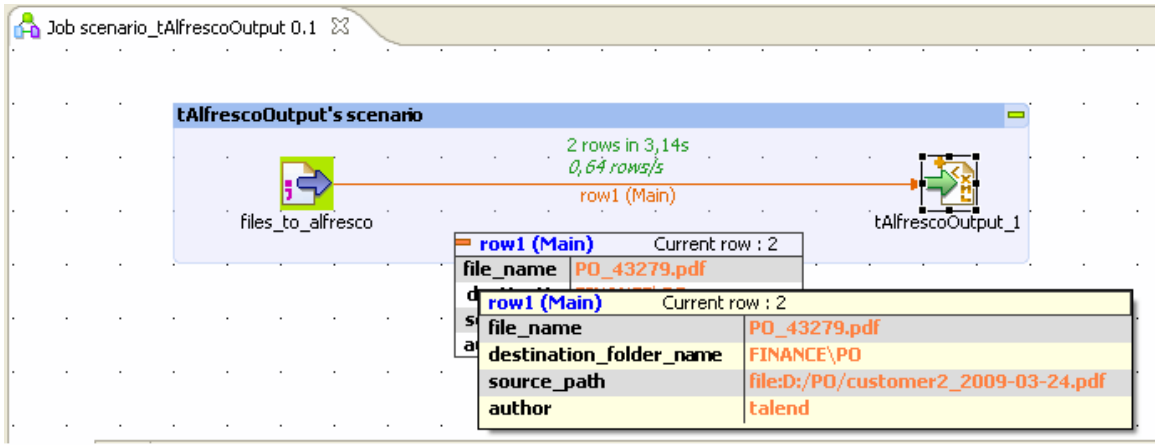
1. Click **Sync columns** to auto propagate all the columns of the delimited file.

   If needed, click **Edit schema** to view the output data structure of **tAlfrescoOutput**.

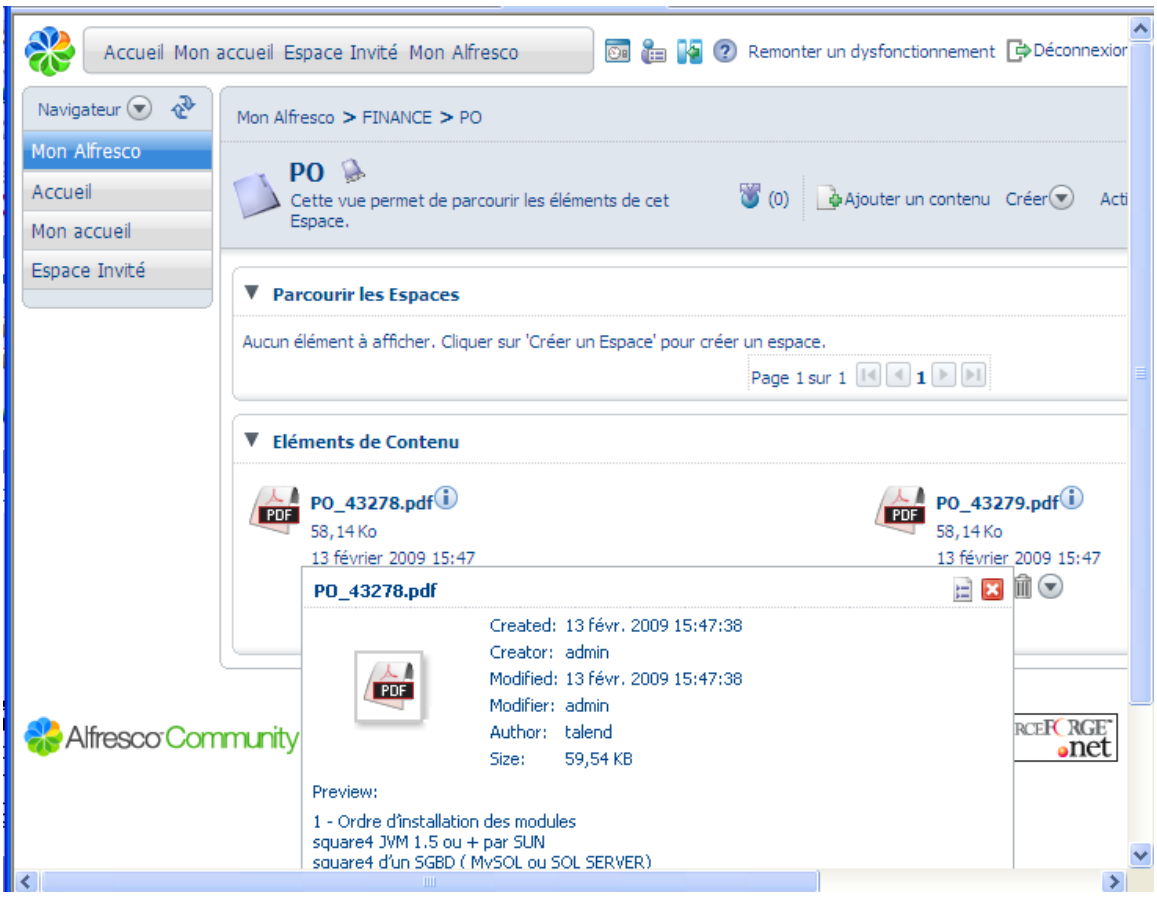


2. Click the three-dot button next to the **Result Log File Name** field and browse to the file where you want to save any logs after Job execution.

3. Save your Job, and press **F6** to execute it.



   The two documents are created in Alfresco using the metadata provided in the input schemas.


Accueil Mon accueil Espace Invité Mon Alfresco
Remonter un dysfonctionnement
Déconnexion
Navigateur
Mon Alfresco
Accueil
Mon accueil
Espace Invité
Mon Alfresco > FINANCE > PO
PO
Cette vue permet de parcourir les éléments de cet Espace.
(0)
Ajouter un contenu
Créer
Parcourir les Espaces
Aucun élément à afficher. Cliquer sur 'Créer un Espace' pour créer un espace.
Page 1 sur 1
1
Eléments de Contenu
PO_43278.pdf
58,14 Ko
13 février 2009 15:47
PO_43279.pdf
58,14 Ko
13 février 2009 15:47
PO_43278.pdf
Created: 13 févr. 2009 15:47:38
Creator: admin
Modified: 13 févr. 2009 15:47:38
Modifier: admin
Author: talend
Size: 59,54 KB
Preview:
1 - Ordre d'installation des modules
square4 JVM 1.5 ou + par SUN
square4 d'un SGBD (MySQL ou SQL SERVER)
Alfresco Community

tMarketoInput

tMarketoInput Properties

Component family	Business/Cloud	
Function	The **tMarketoInput** component retrieves data from a Marketo Web server.	
Purpose	The **tMarketoInput** component allows you to retrieve data from a Marketo DB on a Web server.	
Basic settings	*Endpoint address*	The URL of the Marketo Web server for the SOAP API calls to.
	Secret key	Encrypted authentication code assigned by Marketo. Contact Marketo Support via support@marketo.com to get this information.
	Client Access ID	A user ID for the access to Marketo web service. Contact Marketo Support via support@marketo.com to get this information.
	Operation	Options in this list allow you to retrieve lead data from Marketo to external systems. **getLead**: This operation retrieves basic information of leads and lead activities in Marketo DB. **getMultipleLeads**: This operation retrieves lead records in batch. **getLeadActivities**: This operation retrieves the history of activity records for a single lead identified by the provided key. **getLeadChanges**: This operation checks the changes on Lead data in Marketo DB.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data is stored centrally.
	Columns Mapping	You can set the mapping conditions by making changes in **Edit Schema**. By default, column names in **Column** fields are the same as what they are in the schema. Because some column names in Marketo database may contain blank space, which is not allowed in the component schema, you need to specify the corresponding column fields in the **Columns in Marketo** field. If the defined column names in schema are the same as column names in Marketo database, it is not necessary to set the columns mapping.
	LeadKey type	The data types of **LeadKey** supported by Marketo DB.

	LeadKey value	The value of **LeadKey.**
	Set Include Types	Select this check box to include the types of LeadActivity content to be retrieved. Click the plus button under the **Include Types** area to select in the list types to add. This field is displayed only when you select **getLeadActivity** or **getLeadChanges** from the **Operation** list.
	Set Exclude Types	Select this check box to exclude the types of LeadActivity content to be retrieved. Click the plus button under the **Exclude Types** area to select in the list types to add. This field is displayed only when you select **getLeadActivity** or **getLeadChanges** from the **Operation** list.
	Last Updated At	Type in the time of last update to retrieve only the data since the last specified time. The time format is YYYY-MM-DD HH:MM:SS. This field is displayed only when you select **getMultipleLeads** from the **Operation** list.
	Batch Size	The maximum batch size in retrieving lead data in batch. This field is displayed only when you select **getLeadActivity** or **getLeadChanges** from the **Operation** list.
	Timeout (milliseconds)	Type in the query timeout (in milliseconds) on the Marketo Web service. The Job will stop when Timeout exception error occurs.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** connection.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used as an input component, it requires an output component.	
Limitation	n/a	

Related Scenario

For a related use case, see section *Scenario: Data transmission between Marketo DB and an external system*.

tMarketoListOperation

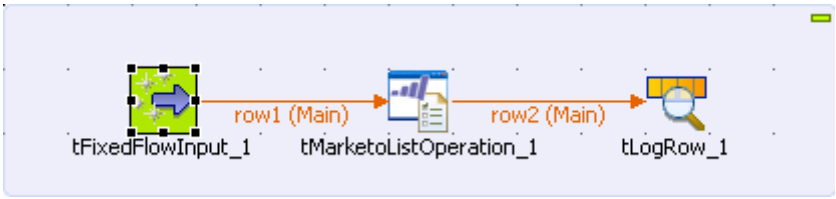
tMarketoListOperation Properties

Component family	Business/Cloud	
Function	The **tMarketoListOperation** component adds/removes one or more leads to/from a list in the Marketo DB; It also verifies if one or more leads exist in a list in Marketo DB.	
Purpose	The **tMarketoListOperation** component allows you to add/remove one or more leads to/from a list in the Marketo DB on a Web server. Also, you can verify the existence of one or more leads in a list in the Marketo DB.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data is stored centrally.
	Endpoint address	The URL of the Marketo Web server for the SOAP API calls to.
	Secret key	Encrypted authentication code assigned by Marketo. Contact Marketo Support via support@marketo.com for further information.
	Client Access ID	A user ID for the access to Marketo web service. Contact Marketo Support via support@marketo.com for further information.
	Operation	Options in this list allow you carry out the adding/deletion one or more leads to/from a list in the Marketo DB; Also you can verify the existence of single or multiple leads in a list in the Marketo DB. **addTo**: This operation adds one or more leads to a list in the Marketo DB. **isMemberOf**: This operation checks the Marketo DB to judge whether the specific leads exist in the list. **removeFrom**: This operation removes one or more leads from a list in the Marketo DB.
	Add or remove multiple leads	Select this check box to add multiple leads to or remove multiple leads from a list in the Marketo DB. This check box appears only when you select **addTo** or **removeFrom** from the **Operation** list.
	Timeout (milliseconds)	Type in the query timeout (in milliseconds) on the Marketo Web service. The Job will stop when Timeout exception error occurs.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If

		needed, you can retrieve the rows on error via a **Row** > **Reject** connection.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used as an intermediate component, it requires an input component and an output component.	
Limitation	n/a	

Scenario: Adding a lead record to a list in the Marketo DB

The following scenario describes a three-component Job that adds a lead record into a list in the Marketo database.

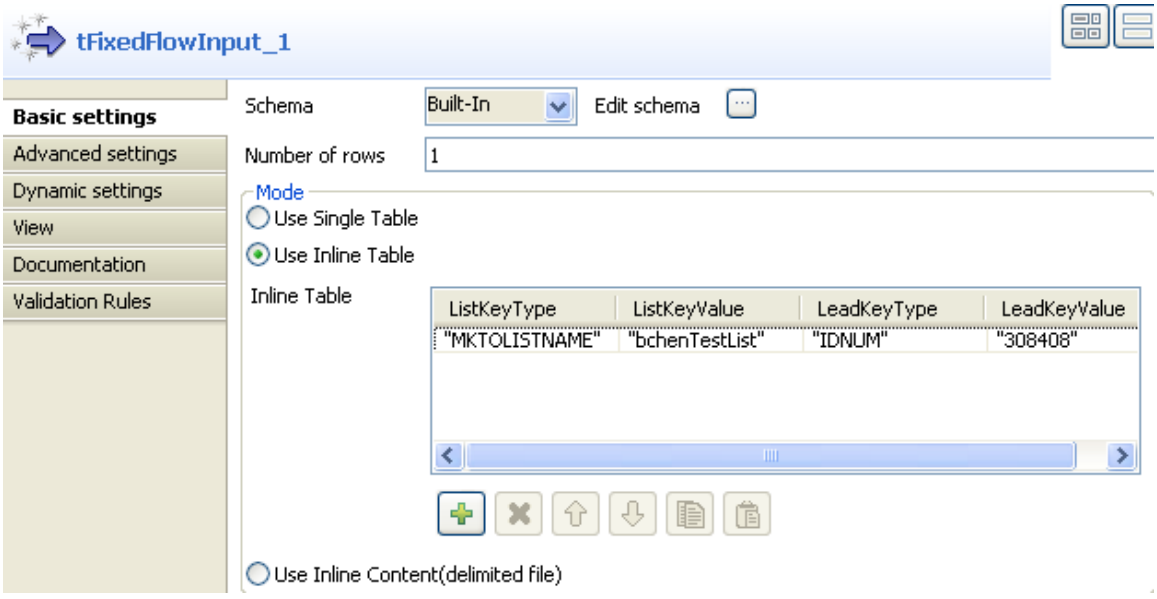


Setting up the Job

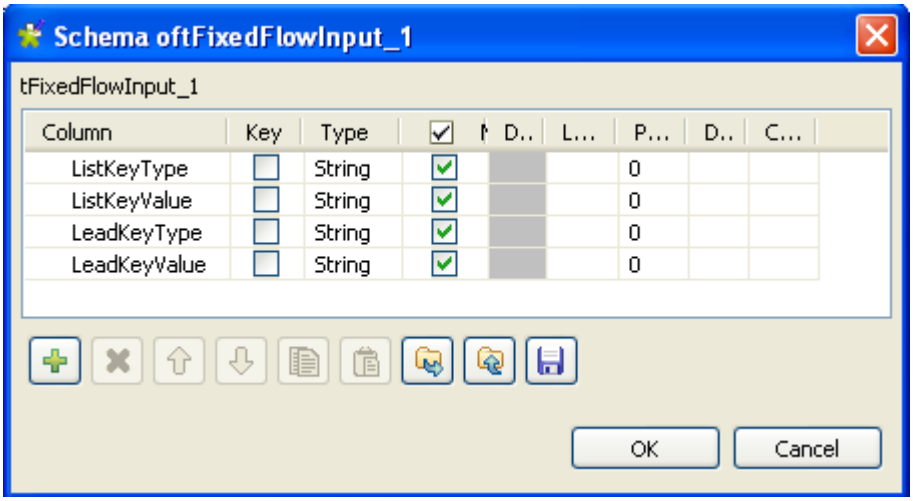
1. Drop **tMarketoListOperation**, **tFixedFlowInput** and **tLogRow** onto the design workspace.
2. Connect **tFixedFlowInput** to **tMarketoListOperation** using a **Row** > **Main** connection.
3. Connect **tMarketoListOperation** to **tLogRow** using a **Row** > **Main** connection.

Configuring the input component

1. Double-click **tFixedFlowInput** to define the component properties in its **Basic settings** view.



2. Click the three-dot button next to **Edit schema** to set the schema manually.



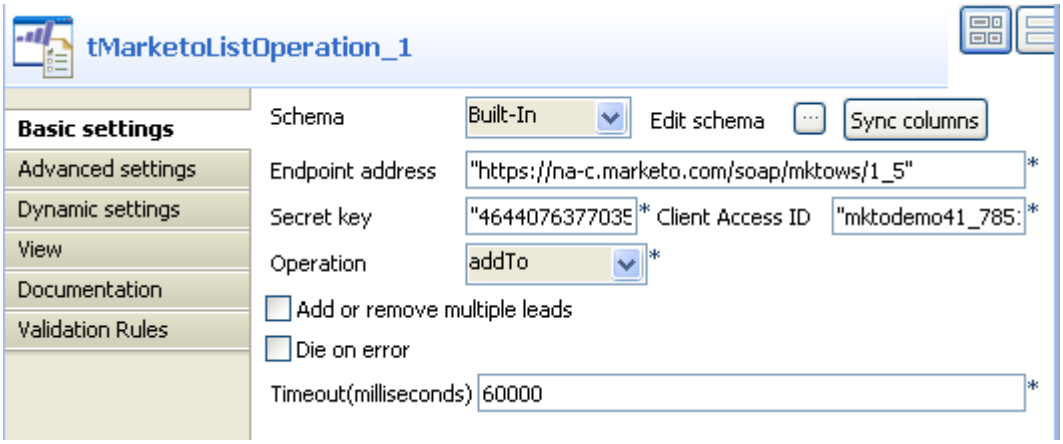
3. Click the plus button to add four columns: *ListKeyType*, *ListKeyValue*, *LeadKeyType* and *LeadKeyValue*. Keep the settings as default. Then click **OK** to save the settings.

4. In the **Mode** area, select **Use Inline Table**.

5. Click the plus button to add a new line and fill the line with respective values. In this example, these values are: *MKTOLISTNAME* for **ListKeyType**, *bchenTestList* for **ListKeyValue**, *IDNUM* for **LeadKeyType** and *308408* for **LeadKeyValue**.

Configuring tMarketoListOperation

1. Double-click **tMarketoListOperation** to define the component properties in its **Basic settings** view.



2. Click the **Sync columns** button to retrieve the schema defined in **tFixedFlowInput**.

3. Type in *1* in the **Number of rows** field.

4. Fill the **Endpoint address** field with the URL of the Marketo Web server. In this example, it is *https://na-c.marketo.com/soap/mktows/1_5*.

   Note that the URL used in this scenario is for demonstration purpose only.

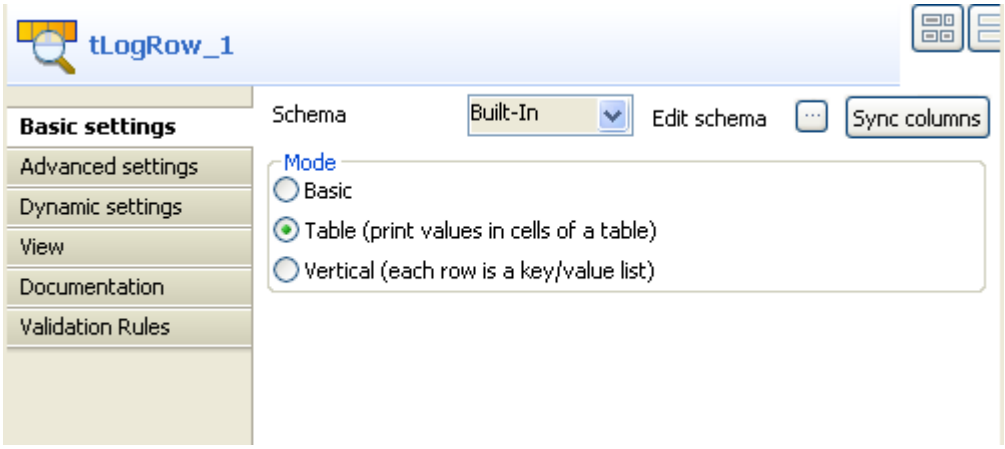
5. Fill the **Secret key** field with encrypted authentication code assigned by Marketo. In this example, it is *464407637703554044DD11AA2211998*.

6. Fill the **Client Access ID** field with the user ID. In this example, it is *mktodemo41_785133934D1A219*.

7. From the **Operation** list, select **addTo**.
8. Type in the limit of query timeout in the **Timeout** field. In this example, use the default number: *60000*.

Job Execution

1. Double-click **tLogRow** to define the component properties in its **Basic settings** view.



2. Click the **Sync columns** button to retrieve the schema defined in **tMarketoListOperation**.
3. In the **Mode** area, select **Table**.
4. Save your Job and press **F6** to execute it.

```
Starting job MarketoListOperation at 17:30 12/05/2011.

[statistics] connecting to socket on port 3497
[statistics] connected
.-----------+-------------+-----------+------------+-------.
|                          tLogRow_1                          |
|=----------+-------------+-----------+------------+------=|
|ListKeyType |ListKeyValue |LeadKeyType|LeadKeyValue|Success|
|=----------+-------------+-----------+------------+------=|
|MKTOLISTNAME|bchenTestList|IDNUM      |308408      |true   |
'-----------+-------------+-----------+------------+-------'

[statistics] disconnected
Job MarketoListOperation ended at 17:30 12/05/2011. [exit
code=0]
```

The result of adding a lead record to a list in Marketo DB is displayed on the **Run** console.

tMarketoOutput

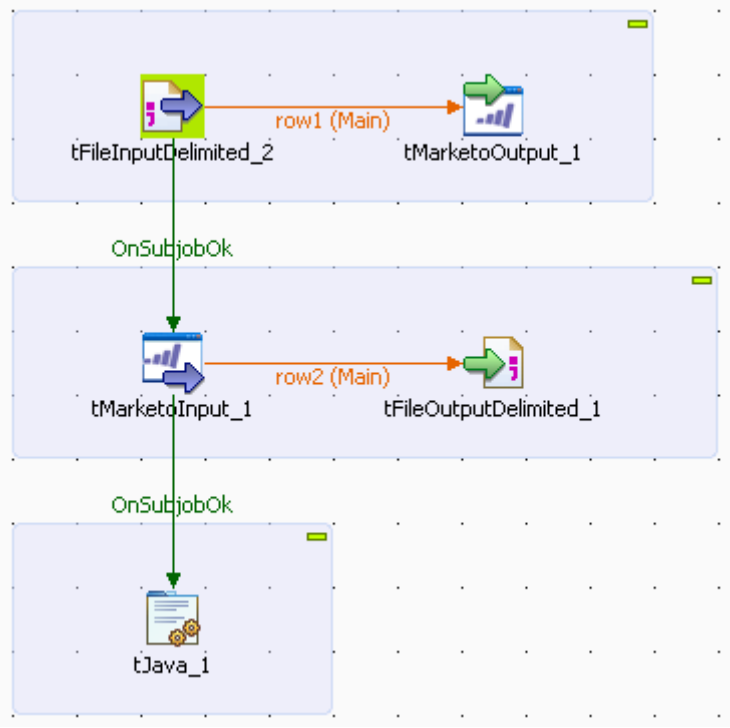
tMarketoOutput Properties

Component family	Business/Cloud	
Function	The **tMarketoOutput** component outputs data to a Marketo Web server.	
Purpose	The **tMarketoOutput** component allows you to write data into a Marketo DB on a Web server.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data is stored centrally.
	Endpoint address	The URL of the Marketo Web server for the SOAP API calls to.
	Secret key	Encrypted authentication code assigned by Marketo. Contact Marketo Support via support@marketo.com to get this information.
	Client Access ID	A user ID for the access to Marketo web service. Contact Marketo Support via support@marketo.com to get this information.
	Operation	Options in this list allow you to synchronize lead data between Marketo and another external system. **syncLead**: This operation requests an insert or update operation for a lead record. **syncMultipleLeads**: This operation requests an insert or update operation for lead records in batch.
	Columns Mapping	You can set the mapping conditions by making changes in **Edit Schema**. By default, column names in **Column** fields are the same as what they are in the schema. Because some column names in Marketo database may contain blank space, which is not allowed in the component schema, you need to specify the corresponding column fields in the **Columns in Marketo** field. If the defined column names in schema are the same as column names in Marketo database, it is not necessary to set the columns mapping.
	De-duplicate lead record on email address	Select this check box to de-duplicate and update lead records using email address. Deselect this check box to create another lead which contains the same email address.

		This check box will be displayed only when you select **syncMultipleLeads** from the **Operation** list.
	Batch Size	The maximum batch size in synchronizing lead data in batch. This field will be displayed only when you select **syncMultipleLeads** from the **Operation** list.
	Timeout (milliseconds)	Type in the query timeout (in milliseconds) on the Marketo Web service. The Job will stop when Timeout exception error occurs.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** connection.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used as an output component, it requires an input component.	
Limitation	n/a	

Scenario: Data transmission between Marketo DB and an external system

The following scenario describes a five-component Job that inserts Lead records into Marketo database and retrieves these records from Marketo database to a local file. Upon completing the data accessing, the Job displays the number of relevant API calls on the **Run** console.



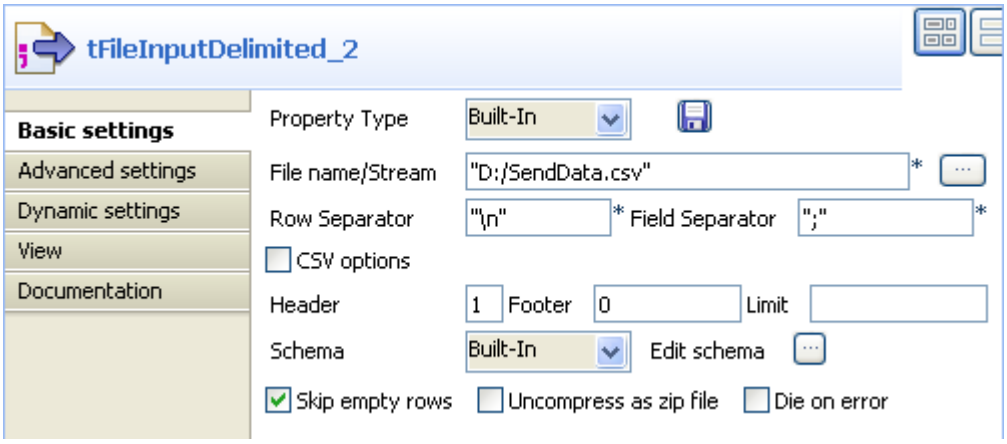
Setting up the Job

1. Drop **tMarketoOutput**, **tMarketoInput**, **tFileInputDelimited**, **tFileOutputDelimited** and **tJava** from the **Palette** onto the design workspace.

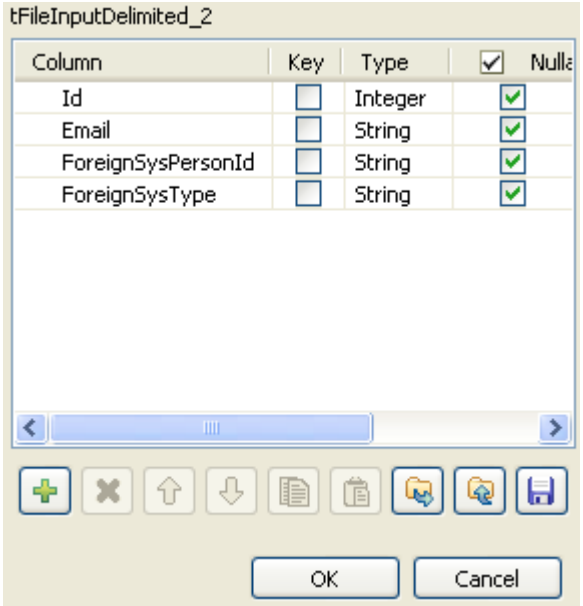
2. Connect **tFileInputDelimited** to **tMarketoOutput** using a **Row** > **Main** connection.
3. Connect **tMarketoInput** to **tFileOutputDelimited** using a **Row** > **Main** connection.
4. Connect **tFileInputDelimited** to **tMarketoInput** using a **Trigger** > **OnSubjectOk** connection.
5. Connect **tMarketoInput** to **tJava** using a **Trigger** > **OnSubjectOk** connection.

Configuring tFileInputDelimited

1. Double-click **tFileInputDelimited** to define the component properties in its **Basic settings** view.



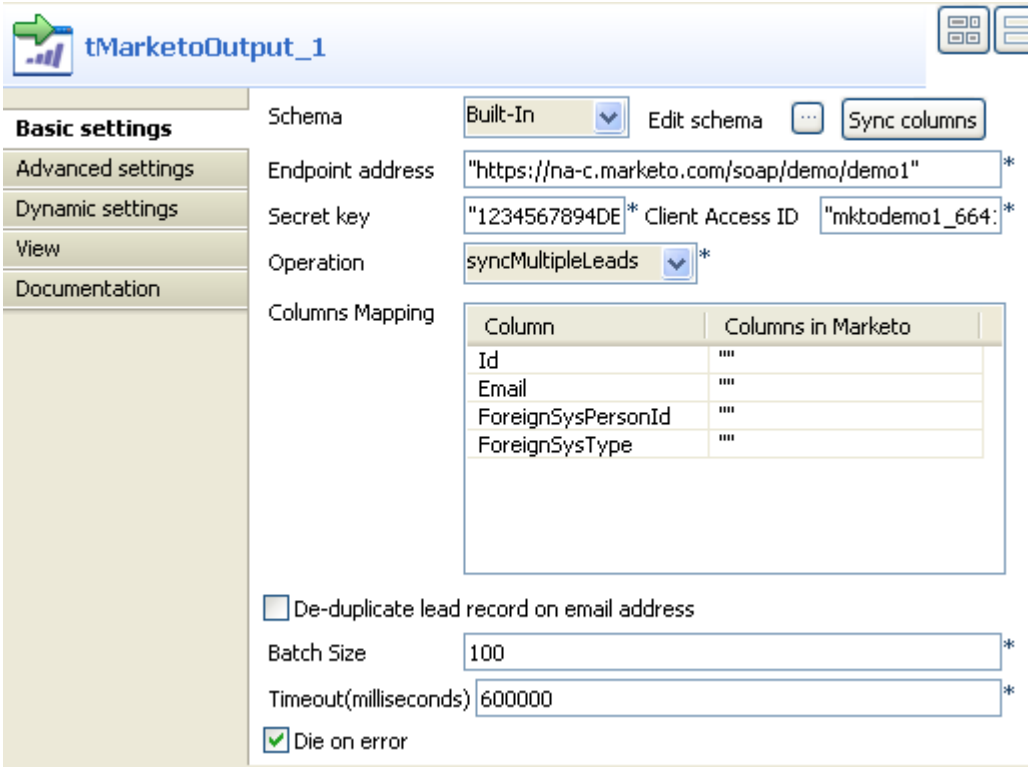
2. Click the three-dot button next to the **File name/Stream** field to select the source file for data insertion. In this example, it is *D:/SendData.csv*.
3. Click the three-dot button next to **Edit schema** to set the schema manually.



4. Click the plus button to add four columns: *Id*, *Email*, *ForeignSysPersonId* and *ForeignSysType*. Set the **Type** of *Id* to Integer and keep the rest as default. Then click **OK** to save the settings.
5. Type in *1* in the **Header** field and keep the other settings as default.

Configuring tMarketoOutput

1. Double-click **tMarketoOutput** to define the component properties in its **Basic settings** view.



2. Click the **Sync columns** button to retrieve the schema defined in **tFileInputDelimited** and fill the **Endpoint address** field with the URL of the Marketo Web server. In this example, it is *https://na-c.marketo.com/soap/demo/demo1*.

   Note that the URL used in this scenario is for demonstration purpose only.

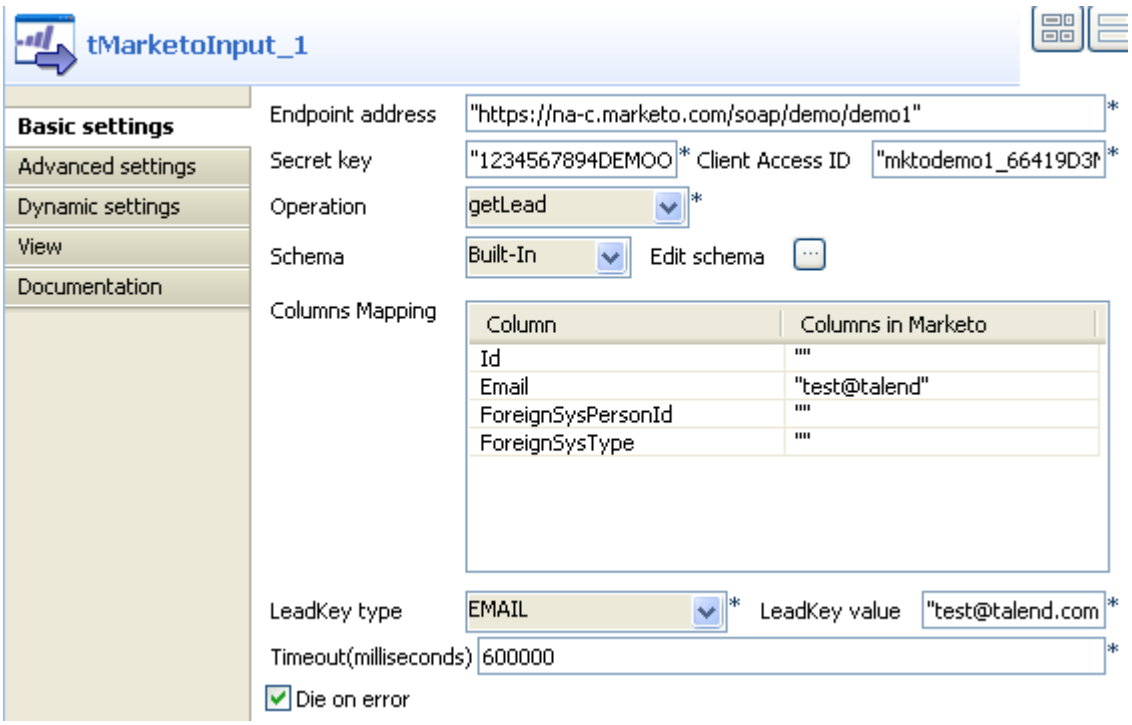
3. Fill the **Secret key** field with encrypted authentication code assigned by Marketo. In this example, it is *1234567894DEMOONLY987654321*.

4. Fill the **Client Access ID** field with the user ID. In this example, it is *mktodemo1_1234567894DEMOONLY987654321*.

5. Select **syncMultipleLeads** from the **Operation** list and type in the limit of query timeout in the **Timeout** field. In this example, use the default number: *600000*.

Configuring tMarketoInput

1. Double-click **tMarketoInput** to define the component properties in its **Basic settings** view.

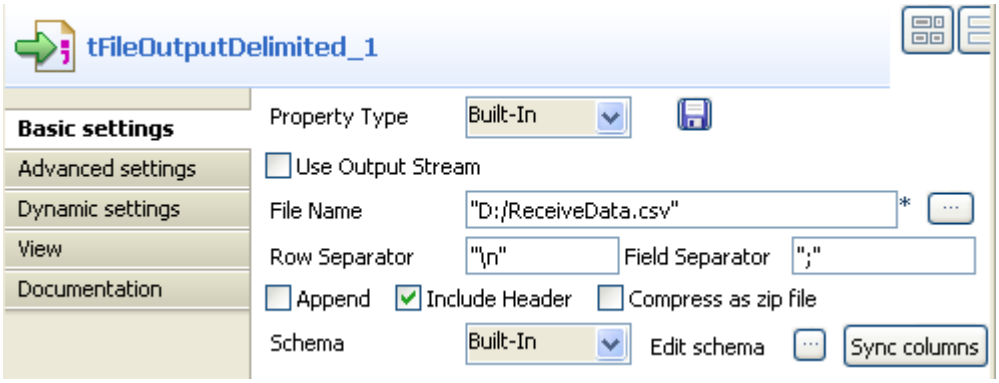


2. From the **Operation** list, select **getLead**.
3. In **Columns Mapping** area, type in *test@talend.com* in **Columns in Marketo** column to set the **Email** column.

   Note that all the data used in this scenario is for demonstration purpose only.
4. From the **LeadKey type** list, select **EMAIL** and fill the **LeadKey value** field with *test@talend.com*.
5. Keep the rest of the settings as the corresponding settings in **tMarketoOutput**.

Configuring tFileOutputDelimited

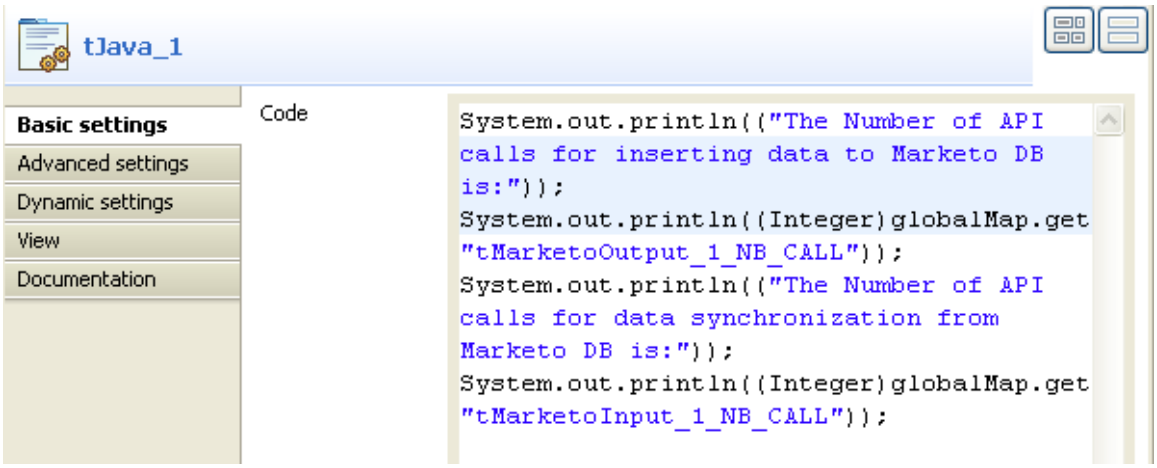
1. Double-click **tFileOutputDelimited** to define the component properties in its **Basic settings** view.



2. Click the three-dot button next to the **File name** field to synchronize data to a local file. In this example, it is *D:/ReceiveData.csv*.
3. Click the **Sync columns** button and keep the rest of the settings as default.

Using Java scripts to count API calls

1. Double-click **tJava** to add code in its **Basic settings** view.



2. In the **Code** field, type in following code to count the number of API calls throughout the data operations:

```
System.out.println(("The Number of API calls for inserting
data to Marketo DB is:"));
System.out.println((Integer)globalMap.get("tMarketoOutput_1_NB_CALL"));
System.out.println(("The Number of API calls for data synchronization
from Marketo DB is:"));
System.out.println((Integer)globalMap.get("tMarketoInput_1_NB_CALL"));
```

Job execution

1. Save your Job.
2. Press **F6** to execute it.

```
Id;Email;ForeignSysPersonId;ForeignSysType
308434;;;
308436;;;
308435;;;
308431;;;
308432;;;
308433;;;
```

The inserted lead records in the Marketo DB are synchronized to *D:/ReceiveData.csv*.

```
Starting job io at 18:03 25/03/2011.

[statistics] connecting to socket on port 3829
[statistics] connected
The Number of API calls for inserting data to
Marketo DB is:
1
The Number of API calls for data synchronization
from Marketo DB is:
1
[statistics] disconnected
Job io ended at 18:03 25/03/2011. [exit code=0]
```

The number of API calls throughout each data operation is displayed on the **Run** console.

tMicrosoftCrmInput

tMicrosoftCrmInput Properties

Component family	Business / Microsoft CRM	
Function	Connects to an entity of Microsoft CRM database via the relevant webservice.	
Purpose	Allows to extract data from a Microsoft CRM DB based on conditions set on specific columns.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Authentication Type	List of authentication types that correspond to specific CRM deployment models. For details, download White paper: Microsoft Dynamics CRM Security Model.
	Microsoft Webservice URL	Type in the webservice URL to connect to the Microsoft CRM DB. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Organizename	Enter the name of the user or organization, set by an administrator, that needs to access the Microsoft CRM database.
	Username and *Password*	Type in the Webservice user authentication data.
	Domain	Type in the domain name of the server on which Microsoft CRM is hosted. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Host	Type in the IP address of Microsoft CRM database server.
	Port	Listening port number of Microsoft CRM database server. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Time out (seconds)	Number of seconds for the port to listen before closing.
	Entity	Select the relevant entity in the list.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. In this component the schema is related to the selected entity.
	Logical operators used to combine conditions	In the case you want to combine the conditions you set on columns, select the combine mode you want to use.
	Conditions	Click the plus button to add as many conditions as needed. The conditions are performed one after the other for each row.

		Input column: Click in the cell and select the column of the input schema the condition is to be set on. **Operator**: Click in the cell and select the operator to bind the input column with the value. **Value**: Type in the column value, between quotes if need be.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

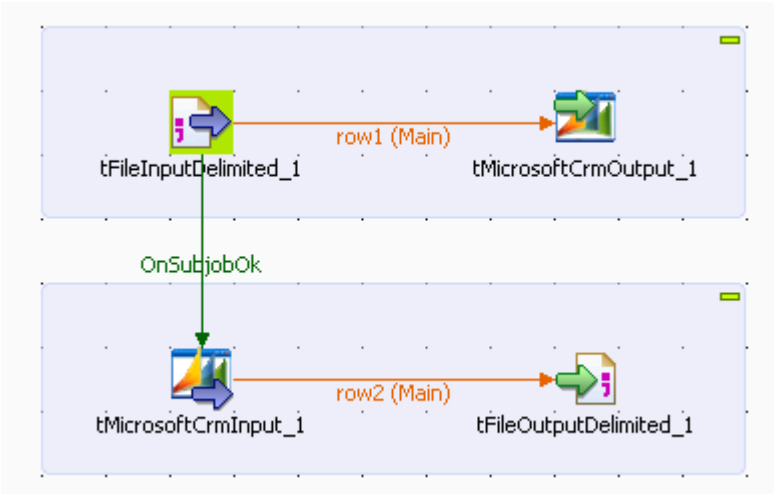
Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows

This scenario describes a four-component Job which aims at writing the data included in a delimited input file in a custom entity in a MicrosoftCRM database. It then extracts specified rows to an output file using the conditions set on certain input columns.

If you want to write in a CustomEntity in Microsoft CRM database, make sure to name the columns in accordance with the naming rule set by Microsoft, that is "name_columnname" all in lower case.

Setting up the Job

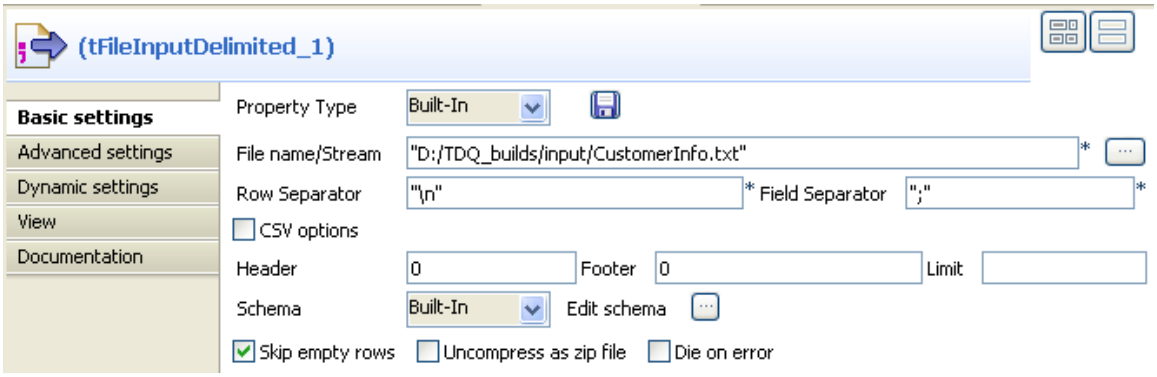
1. Drop the following components from the **Palette** to the design workspace: **tFileInputdelimited**, **tFileOutputDelimited**, **tMicrosoftCrmInput**, and **tMicrosoftCrmOutput**.



2. Connect **tFileInputDelimited** to **tMicrosoftCrmOutput** using a **Row Main** connection.
3. Connect **tMicrosoftCrmIntput** to **tFileOutputDelimited** using a **Row Main** connection.
4. Connect **tFileInputDelimited** to **tMicrosoftCrmInput** using **OnSubjobOk** connection.

Configuring tFileInputDelimited

1. Double-click **tFileInputDelimited** to display its **Basic settings** view and define its properties



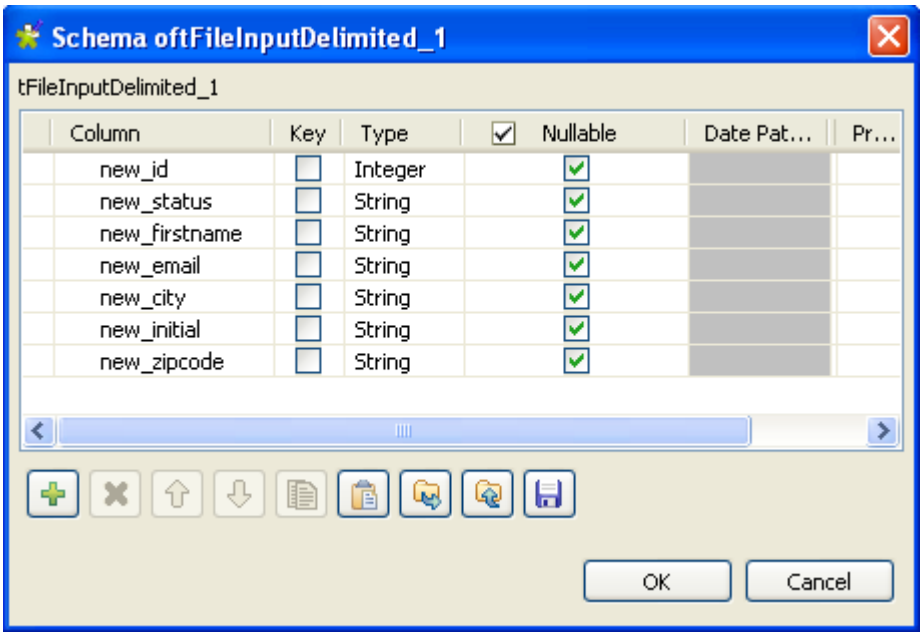
2. Click the three-dot button next to the **File Name/Input Stream** field and browse to the delimited file that holds the input data. The input file in this example contains the following columns: *new_id*, *new_status*, *new_firstname*, *new_email*, *new_city*, *new_initial* and *new_zipcode*.

```
new_id;new_status;new_firstname;new_email;new_city;new_initial;new_zipcode
1;married;Paul;pnewman@comp.com;New York;P.M;55677
2;single;Raul;pnewman@comp.com;New York;R.L;55677
3;single;Mary;mnewman@comp.com;Chicago;M.B;66898
4;married;John;Jnewman@comp.com;Chicago,J.M;66898
5;single;Martin;mnewman@comp.com;Sunnyvale,M.P;33662
6;married;Janet;jnewman@comp.com;Sunnyvale,J.P;33662
7;married;Harry;hnewman@comp.com;New York;H.M;55677
8;married;Jerry;jnewman@comp.com;New York;J.M;55677
9;married;Alice;anewman@comp.com;New York;A.M;55677
10;single;Jack;jnewman@comp.com;New York;J.M;55677
```

3. In the **Basic settings** view, define the **Row Separator** allowing to identify the end of a row. Then define the **Field Separator** used to delimit fields in a row.

4. If needed, define the header, footer and limit number of processed rows in the corresponding fields. In this example, the header, footer and limits are not set.

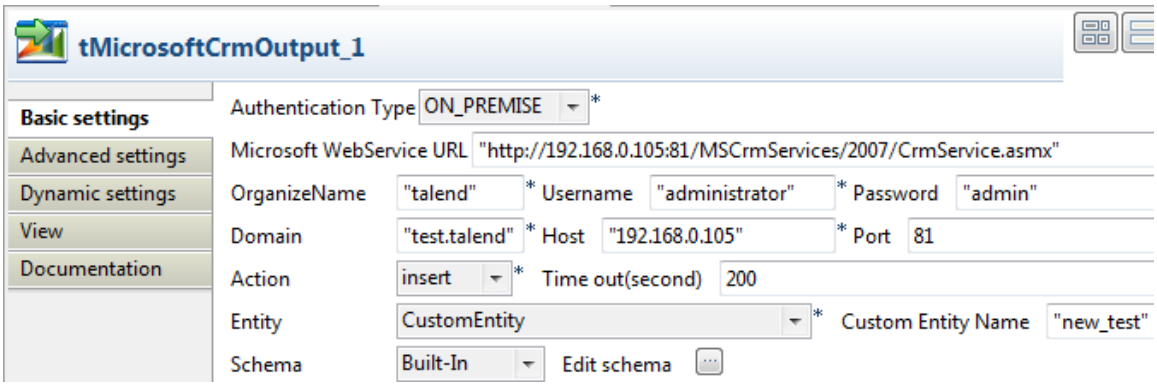
5. Click **Edit schema** to open a dialog box where you can define the input schema you want to write in Microsoft CRM database.



6. Click **OK** to close the dialog box.

Configuring tMicrosoftCrmOutput

1. Double-click **tMicrosoftCrmOutput** to display the component **Basic settings** view and define its properties.



2. Enter the Microsoft Web Service URL as well as the user name and password in the corresponding fields.

3. In the **OrganizeName** field, enter the name that is given the right to access the Microsoft CRM database.

4. In the **Domain field**, enter the domain name of the server on which Microsoft CRM is hosted, and then enter the host IP address and the listening port number in the corresponding fields.

5. In the **Action** list, select the operation you want to carry on. In this example, we want to insert data in a custom entity in Microsoft Crm.

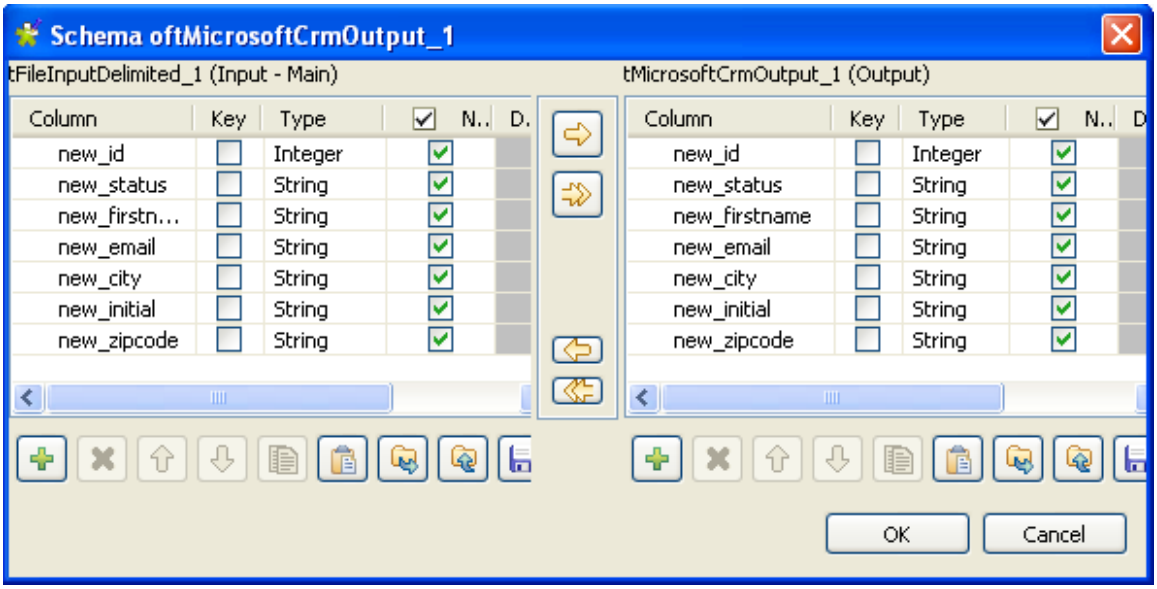
6. In the **Time out** field, set the amount of time (in seconds) after which the Job will time out.

7. In the **Entity** list, select one among those offered. In this example, *CustomEntity* is selected.

   If *CustomEntity* is selected, a **Custom Entity Name** field displays where you need to enter a name for the custom entity.

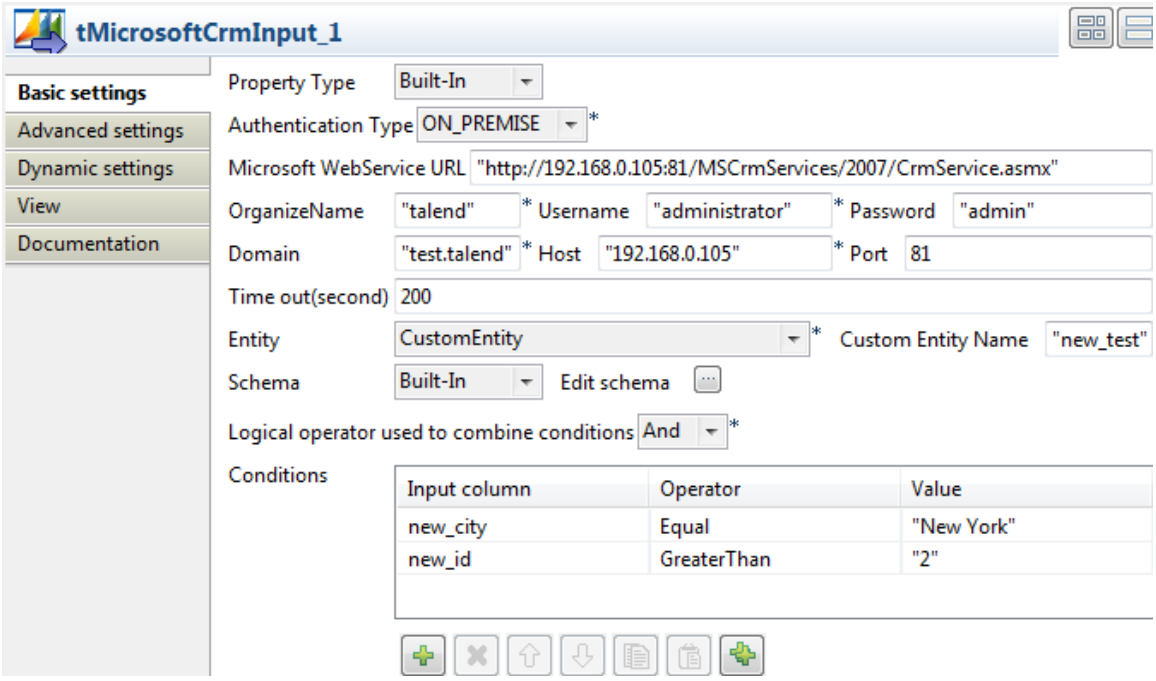
   The **Schema** is then automatically set according to the entity selected. If needed, click **Edit schema** to display a dialog box where you can modify this schema and remove the columns that you do not need in the output.

8. Click **Sync columns** to retrieve the schema from the preceding component.



Configuring tMicrosoftCrmInput

1. Double-click **tMicrosoftCrmInput** to display the component **Basic settings** view and define its properties.



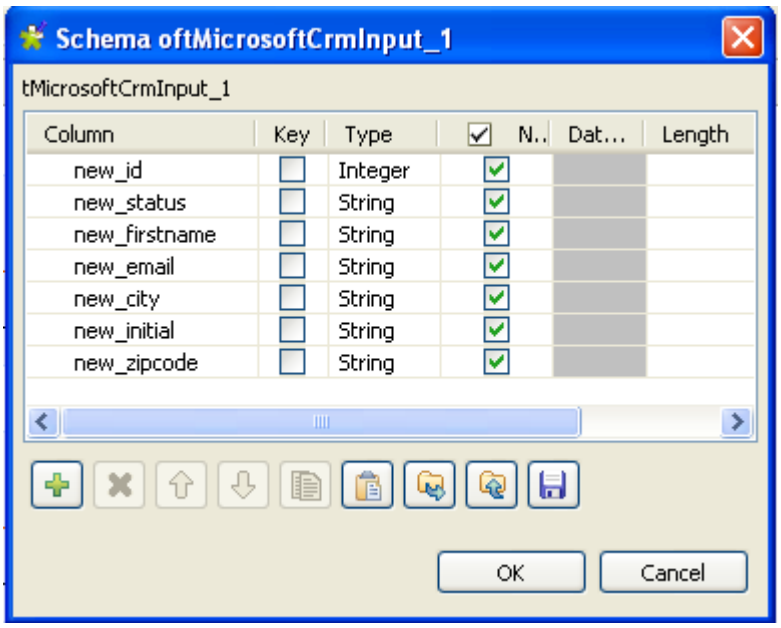
2. Enter the Microsoft Web Service URL as well as the user name and password in the corresponding fields and enter the name that is given the right to access the Microsoft CRM database in the **OrganizeName** field.

3. In the **Domain field**, enter the domain name of the server on which Microsoft CRM is hosted, and then enter the host IP address and the listening port number in the corresponding fields.

4. In the **Time out** field, set the amount of time (in seconds) after which the Job will time out.

5. In the **Entity** list, select the one among those offered you want to connect to. In this example, *CustomEntity* is selected.

6. The **Schema** is then automatically set according to the entity selected. But you can modify it according to your needs. In this example, you should set the schema manually since you want to access a custom entity. Copy the seven-column schema from **tMicrosoftCrmOutput** and paste it in the schema dialog box in **tMicrosoftCrmInput**.



7. Click **OK** to close the dialog box. You will be prompted to propagate changes. Click **Yes** in the popup message.

8. In the **Basic settings** view, select **And** or **Or** as the logical operator you want to use to combine the conditions you set on the input columns. In this example, we want to set two conditions on two different input columns and we use **And** as the logical operator.

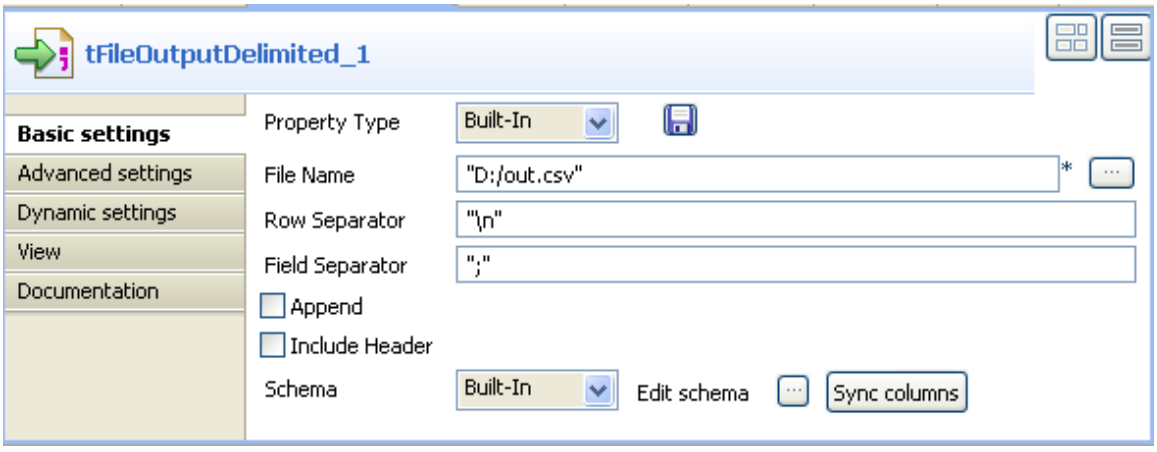
9. In the **Condition** area, click the plus button to add as many lines as needed and then click in each line in the **Input column** list and select the column you want to set condition on. In this example, we want to set conditions on two columns, *new-city* and *new_id*. We want to extract all customer rows whose city is equal to “New York” and whose id is greater than 2.

10. Click in each line in the **Operator** list and select the operator to bind the input column with its value, in this example **Equal** is selected for *new_city* and **Greater Than** for *new_id*.

11. Click in each line in the **Value** list and set the column value, `New York` for *new_city* and `2` for *new_id* in this example. You can use a fixed or a context value in this field.

Configuring tFileOutputDelimited

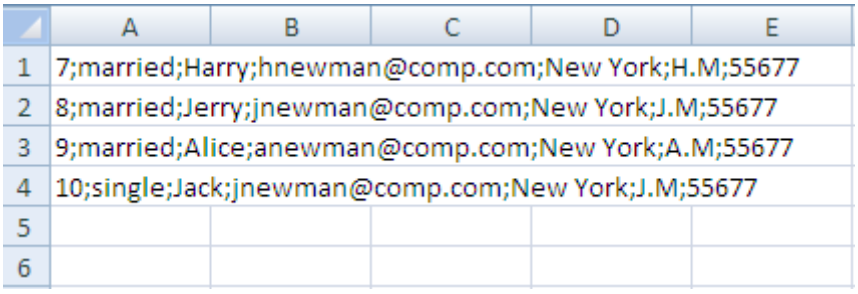
1. Double-click **tFileOutputdelimited** to display the component **Basic settings** view and define its properties.



2. Click the three-dot button next to the **File Name** field and browse to the output file.
3. Set row and field separators in the corresponding fields.
4. Select the **Append** check box if you want to add the new rows at the end of the records.
5. Select the **Include Header** check box if the output file includes a header.
6. Click **Sync columns** to retrieve the schema from the preceding component.

Job execution

Save the Job and press **F6** to execute it.



	A	B	C	D	E
1	7;married;Harry;hnewman@comp.com;New York;H.M;55677				
2	8;married;Jerry;jnewman@comp.com;New York;J.M;55677				
3	9;married;Alice;anewman@comp.com;New York;A.M;55677				
4	10;single;Jack;jnewman@comp.com;New York;J.M;55677				
5					
6					

Only customers who live in New York city and those whose “id” is greater than 2 are listed in the output file you stored locally.

tMicrosoftCrmOutput

tMicrosoftCrmOutput Properties

Component family	Business / Microsoft CRM	
Function	Writes in an entity of a Microsoft CRM database via the relevant webservice.	
Purpose	Allows to write data into a Microsoft CRM DB.	
Basic settings	*Authentication Type*	List of authentication types that correspond to specific CRM deployment models. For details, download White paper: Microsoft Dynamics CRM Security Model.
	Microsoft Webservice URL	Type in the webservice URL to connect to the Microsoft CRM DB. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Organizename	Enter the name of the organization that needs to access the Microsoft CRM database
	Username and *Password*	Type in the Webservice user authentication data.
	Domain	Type in the domain name of the server that installs Microsoft CRM server. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Host	Type in the IP address of Microsoft CRM database server.
	Port	Listening port number of Microsoft CRM database server. (Available when **On_Premise** is selected from the **Authentication Type** list.)
	Action	Select in the list the action you want to do on the CRM data. Available actions are: **insert**, **update**, and **delete**.
	Time out (seconds)	Number of seconds for the port to listen before closing.
	Entity	Select the relevant entity in the list.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Lookup Type Mapping	Add lines as needed to establish mappings between the source and target tables. Select a lookup object from the **Input column** drop down list and enter the keyword of the source tables in the **Type** field.
Advanced settings	*Reuse Http Client*	Select this check box to retain the current connection or deselect it to release the connection.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Used as an output component. An Input component is required.	

Limitation	n/a

Related Scenario

For a related use case, see section *Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows*.

tOpenbravoERPInput

tOpenbravoERPInput properties

Component Family	Business	
Function	**tOpenbravoERPInput** connects to an OpenbravoERP database entity via the appropriate Web service.	
Purpose	This component allows you to extract data from OpenBravoERP database according to the conditions defined in specific columns.	
Basic settings	*Openbravo REST WebService URL*	Enter the URL of the Web service that allows you to connect to the OpenbravoERP database.
	Username et *Password*	User authentication information.
	Entity	Select the appropriate entity from the drop-down list.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. For this component, the schema corresponds to a selected entity.
	WHERE Clause	Enter your WHERE clause.
	Order by	Select this check bow to define how to order the results (the elements in the drop-down list depend on the entity selected) **Sort**: Choose whether to organise the results in either **Ascending** or **Descending** order.
	First result	Enter the row number you want to retrieve first.
	Max result	Enter the maximum number of results you want to retrieve.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators to be used for the numbers. Either: **Thousands separator** or **Decimal separator**
	tStatCatcher Statistics	Select this check box to collect the log data at a component level.
Utilisation	This component is generally used as an input component. An output component is required.	
Limitation	n/a	

Related Scenario

For a scenario in which **tOpenbravoERPInput** might be used, see section *Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows*

tOpenbravoERPOutput

tOpenbravoERPOutput properties

Component Family	Business	
Function	**tOpenbravoERPOutput** writes an object in an OpenbravoERP database via the appropriate Web service.	
Purpose	This component writes data in an OpenbravoERP database.	
Basic settings	*Openbravo REST Webservice URL*	Enter the URL of the Web service that allows you to connect to the OpenbravoERP database.
	Username et *Password*	User authentication information.
	Action on data	From the list, select the one of the following actions: **Update/Create** or **Remove**
	Use existing data file	Select this check box if desired and then select the file by browsing your directory.
	Entity	Select the appropriate entity from the drop-down list.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. Click **Edit Schema** to make changes to the schema. . Click **Sync columns** to retrieve the schema from the previous component connected in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is used as an output component. It requires an input component.	
Limitation	n/a	

Related scenario

For a scenario in which **tOpenbravoERPOutput** may be used, see section *Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows*.

tSageX3Input

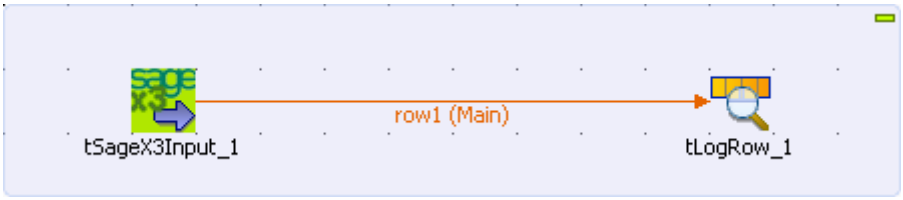
tSageX3Input Properties

Component family	Business/Sage X3	
Function	This component leverages the Web service provided by a given Sage X3 Web server to extract data from the Sage X3 system (the X3 server).	
Purpose	This component extracts data from a given Sage X3 system.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Edit Schema	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
	Endpoint address	Type in the address of the Web service provided by the given Sage X3 Web server.
	Username and *Password*	Type in the Web service user authentication data that you have defined for configuring the Sage X3 Web server.
	Language	Type in the name of the X3 language code used to start a connection group.
	Pool alias	Type in the name of the connection pool that distributes the received requests to available connections. This name was given from the Sage X3 configuration console.
	Request config	Type in the configuration string if you want to retrieve the debug or trace information. For example, the string could be: `RequestConfigDebug="adxwss .trace.on=on";` If you need use several strings, separate them with a `&`, for example, `RequestConfigDebug="adxwss.trace .on=on&adxwss.trace.size=16384";` *A third party tool is needed to retrieve this kind of information.*
	Publication name	Type in the publication name of the published object, list or sub-program you want your Studio to access.
	Mapping	Complete this table to map the variable elements of the object, the sub-program or the list set in the given Sage X3 Web server. The columns to be completed include: **Column**: the columns defined in the schema editor for this component.**Group ID**: the identifier of each variable element group. For example, a variable element group could represent one of attributes of an object.**Field name**: the field name of each variable element.
	Query condition	Select this check box to set up the query condition(s). The columns to be completed include: **Key**: the names of the variable elements used as the key for data extraction.

		Value: the value of the given key field used to extract the corresponding data.
	Limit	Type in a number to indicate the maximum row count of the data to be extracted.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario: Using query key to extract data from a given Sage X3 system

This scenario describes a two-component Job used to extract one row of data from a given Sage X3 system. The object method is to be called, that means the variable elements of this object thus are attributes. The data used in this scenario can be found in the example provided by Sage X3.

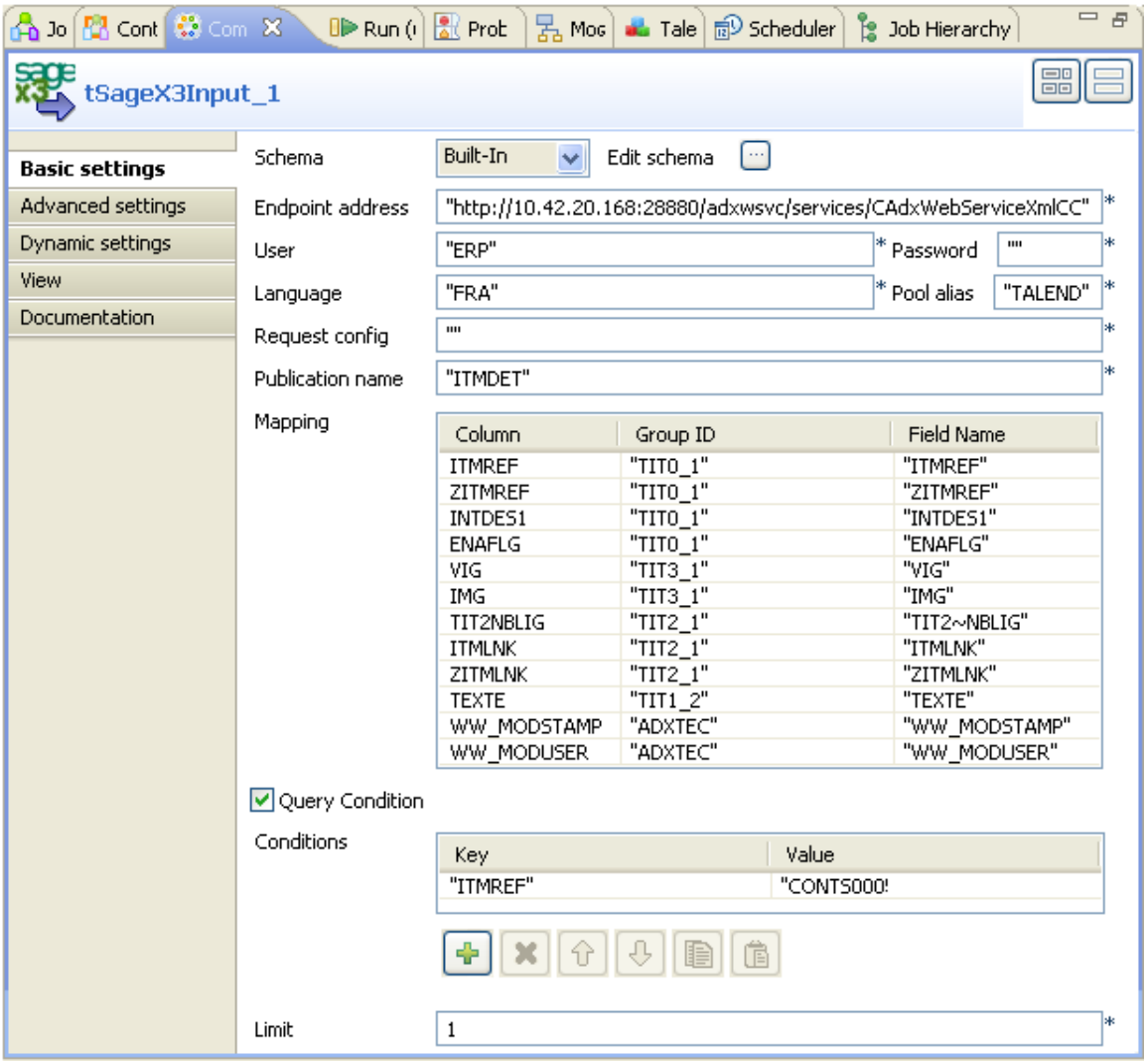


Setting up the Job

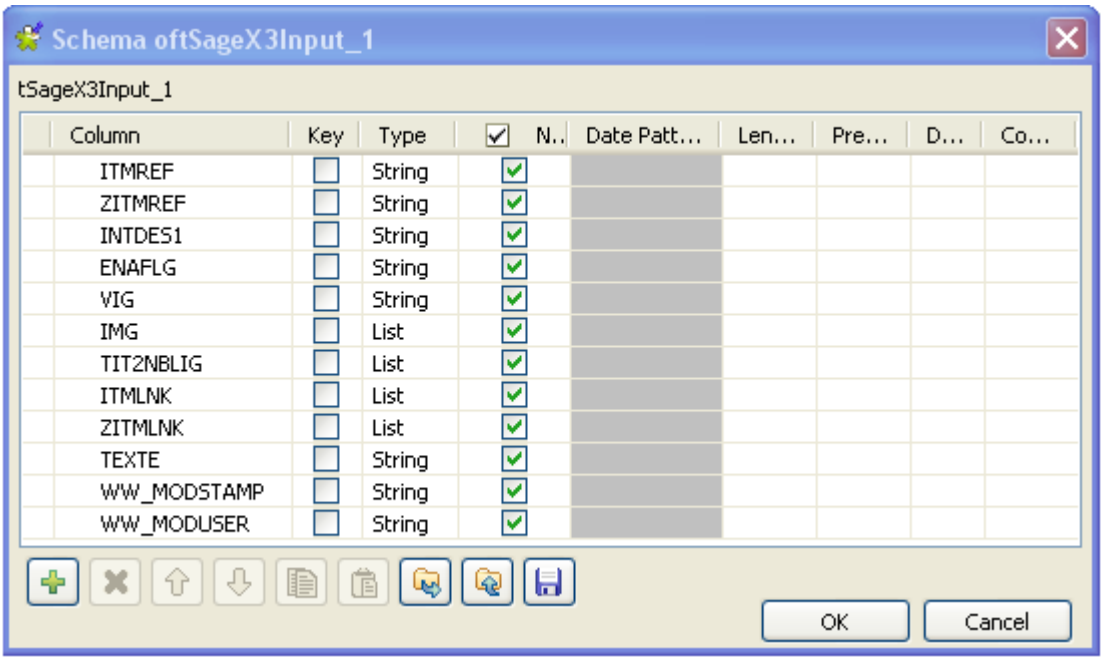
1. Drop the **tSageX3Input** component and the **tLogRow** components onto the workspace from **Palette**.
2. Connect the **tSageX3Input** component to the **tLogRow** component using a **Row** > **Main** link.

Configuring the schema of tSageX3Input

1. Double-click **tSageX3Input** to set its properties in the **Basic Settings** view.



2. Click the three-dot button next to **Edit schema** to open the schema editor.



3. In this editor, click the plus button 12 times beneath the schema table to add 12 rows into this table.

4. Type in the names you want to use for each row. In this example, these rows are named after the publication names of the object attributes set in the Sage X3 Web server. These columns are used to map the corresponding attribute fields in the Sage X3 system.

5. In the **Type** column, click the *IMG* row to display its drop-down list.

6. From the drop-down list, select **List** as this attribute appears twice or even more and do the same to switch the types of the *TIT2NBLIG* row, the *ITMLNK* row and the *ZITMLNK* row to **List** as well for the same reason.

7. Click **OK** to validate this change and accept the propagation prompted by a pop-up dialog box.

Configuring the connection to the Sage X3 Web server

1. In the **Endpoint address** field, type in the URL address of the Web service provided by the Sage X3 Web server. In this example, it is *http://10.42.20.168:28880/adxwsvc/services/CAdxWebServiceXmlCC*

2. In the **User** field, type in the user name of the given Sage X3. In this example, it is *ERP*.

3. In the **Language** field, type in the name of the X3 language code used to start a connection group. In this example, it is *FRA*.

4. In the **Pool alias** field, type in the name of connection pool to be used. In this example, this connection pool is called *TALEND*.

5. In the **Publication name** field, type in the publication name of the object to be called. In this scenario, the publication name is *ITMDET*.

Setting up the mapping and configuring the query condition

1. In the **Group ID** column and the **Field name** column of the **Mapping** table, type in values corresponding to the attribute group IDs and the attribute publication names defined in the Sage X3 Web server. In this example, the values are presented in the figure below.

Column	Group ID	Field Name
ITMREF	"TIT0_1"	"ITMREF"
ZITMREF	"TIT0_1"	"ZITMREF"
INTDES1	"TIT0_1"	"INTDES1"
ENAFLG	"TIT0_1"	"ENAFLG"
VIG	"TIT3_1"	"VIG"
IMG	"TIT3_1"	"IMG"
TIT2NBLIG	"TIT2_1"	"TIT2~NBLIG"
ITMLNK	"TIT2_1"	"ITMLNK"
ZITMLNK	"TIT2_1"	"ZITMLNK"
TEXTE	"TIT1_2"	"TEXTE"
WW_MODSTAMP	"ADXTEC"	"WW_MODSTA...
WW_MODUSER	"ADXTEC"	"WW_MODUSER"

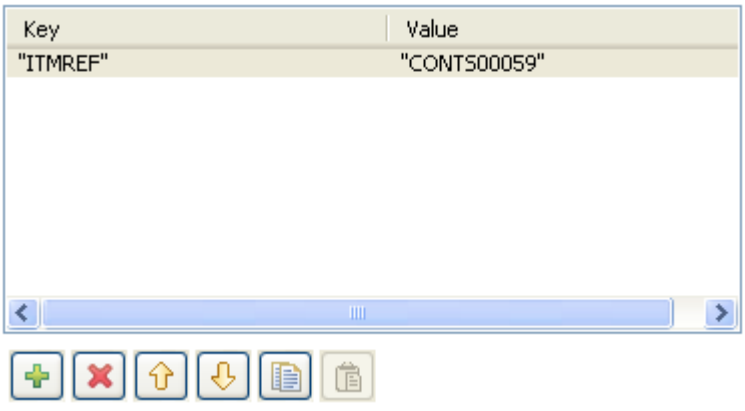
In the **Mapping** table, the **Column** column has been filled automatically with the columns you created in the schema editor.

2. Select the **Query condition** check box to activate the **Conditions** table.

3. Under the **Conditions** table, click the plus button to add one row into the table.

4. In the **Key** column, type in the publication name associated with the object attribute you need to extract data from.

5. In the **Value** column, type in the value of the attribute you have selected as the key of the data extraction. In this scenario, it is *CONTS00059*, one of the product references.



Job execution

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The results are displayed on the **Run** console:

```
+----------+----------------------+----------------------+------+----------------+--------------------+---------+------------+
|                                                                                  tLogRow_1                                  |
|=---------+----------------------+----------------------+------+----------------+--------------------+---------+------------=|
|ITMREF    |ZITMREF               |INTDES1               |ENAFLG|VIG             |IMG                 |TIT2NBLIG|ITMLNK      |
|=---------+----------------------+----------------------+------+----------------+--------------------+---------+------------=|
|CONTS00060|Ecran 24" premium 16/10|Ecran 24" standard 28/10|2     |vig_CONTS00060.jpg|[img1_CONTS00060.jpg]|[1]      |[PFSER00097]|
'----------+----------------------+----------------------+------+----------------+--------------------+---------+------------'
```

tSageX3Output

tSageX3Output Properties

Component family	Business/Sage X3	
Function	This component connects to the Web service provided by a given Sage X3 Web server and therefrom insert, update or delete data in the Sage X3 system (the X3 server).	
Purpose	This component writes data into a given Sage X3 system.	
Basic settings	*Property type*	If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Edit Schema	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Endpoint address	Type in the address of the Web service provided by the given Sage X3 Web server.
	Username and *Password*	Type in the Web service user authentication data that you have defined for configuring the Sage X3 Web server.
	Language	Type in the name of the X3 language code used to start a connection group.
	Pool alias	Type in the name of the connection pool that distributes the received requests to available connections. This name was given from the Sage X3 configuration console.

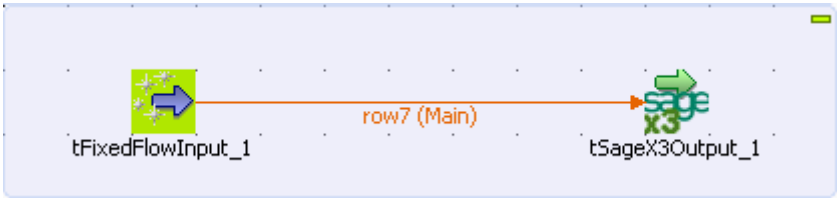
	Request config	Type in the configuration string if you want to retrieve the debug or trace information.
		
		For example, the string could be: `"RequestConfigDebug=“adxwss.trace.on=on";`
		
		If you need use several strings, separate them with a `&`, for example,
		
		`RequestConfigDebug="adxwss.trace.on`
		
		`=on&adxwss.trace.size=16384";`
		
		A third party tool is needed to retrieve this kind of information.

	Publication name	Type in the publication name of the published object, list or sub-program you want your Studio to access.
	Action	You can do any of the following operations on the data in a Sage X3 system: **Insert**: insert data **Update**: update data **Delete**: delete data
	Mapping	Complete this table to map the variable elements of the object, the list or the sub-program your Studio access. Only the elements you

		need to conduct the data action of your interest on are selected and typed in for the purpose of mapping. The columns to be completed include: **Column**: the columns defined in the schema editor for this component. **Key**: the variable element used as key for data insertion, update or deletion. Select the corresponding check box if a variable element is the key. **Group ID**: the identifier of each variable element group. For example, a variable element group could represent one of attributes of an object.**Field name**: the field name of each selected variable element.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as an output component. An input component is required.	
Limitation	n/a	

Scenario: Using a Sage X3 Web service to insert data into a given Sage X3 system

This scenario describes a two-component Job used to generate one row of data and insert the data into a given Sage X3 system. You can find the data used in this scenario in the example provided by Sage X3. The Sage X3 Web service is used to access an object.

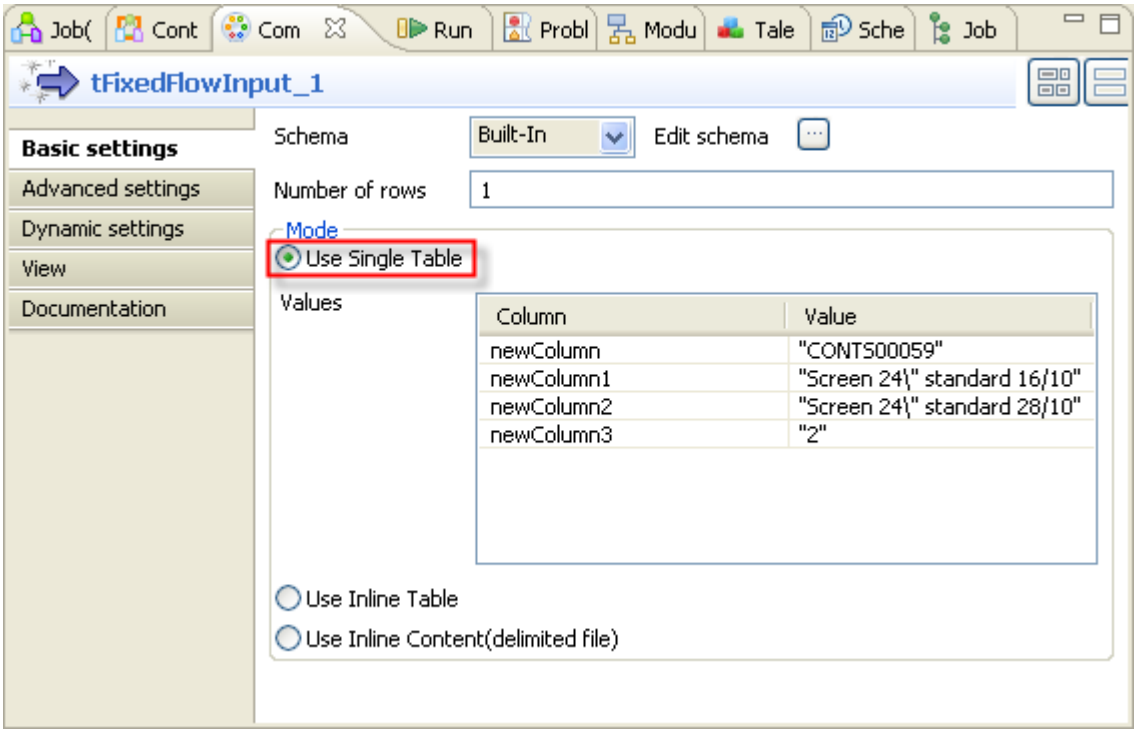


Setting up the Job

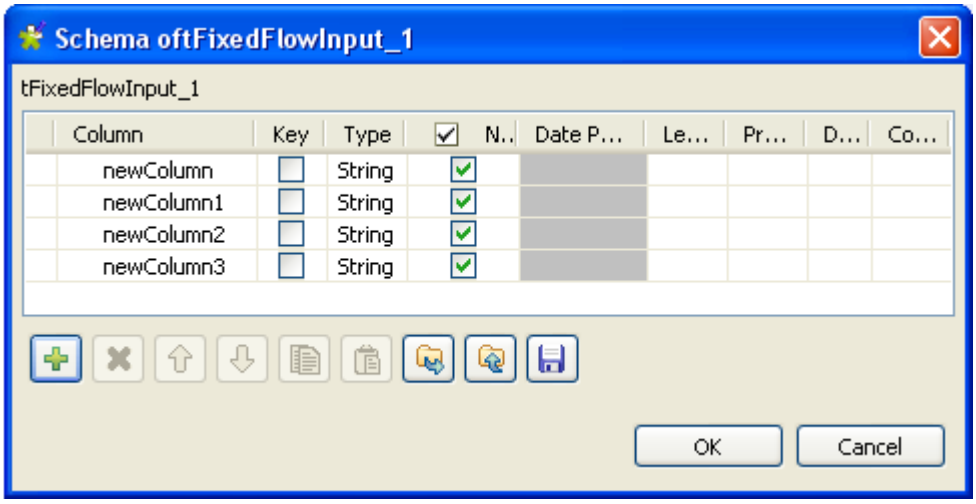
1. Drop the **tFixedFlowInput** and the **tSageX3Output** components onto the workspace from **Palette**.
2. Connect the **tFixedFlowInput** component to the **tSageX3Output** component using a **Row** > **Main** connection.

Configuring the schema for the input data

1. Double-click the **tFixedFlowInput** component to set its **Basic Settings** in the **Component** view



2. Click the three-dot button next to **Edit schema** to open the schema editor.



3. In the schema editor and then under the schema table, click the plus button four times to add four rows.

4. Click **OK** to validate this changes and then accept the propagation prompted by the pop-up dialog box. The four rows appear automatically in the **Values** table of the **Component** view.

5. In the **Values** table within the **Mode** area, type in the values for each of the four rows in the **Value** column. In this scenario, the values downward are:

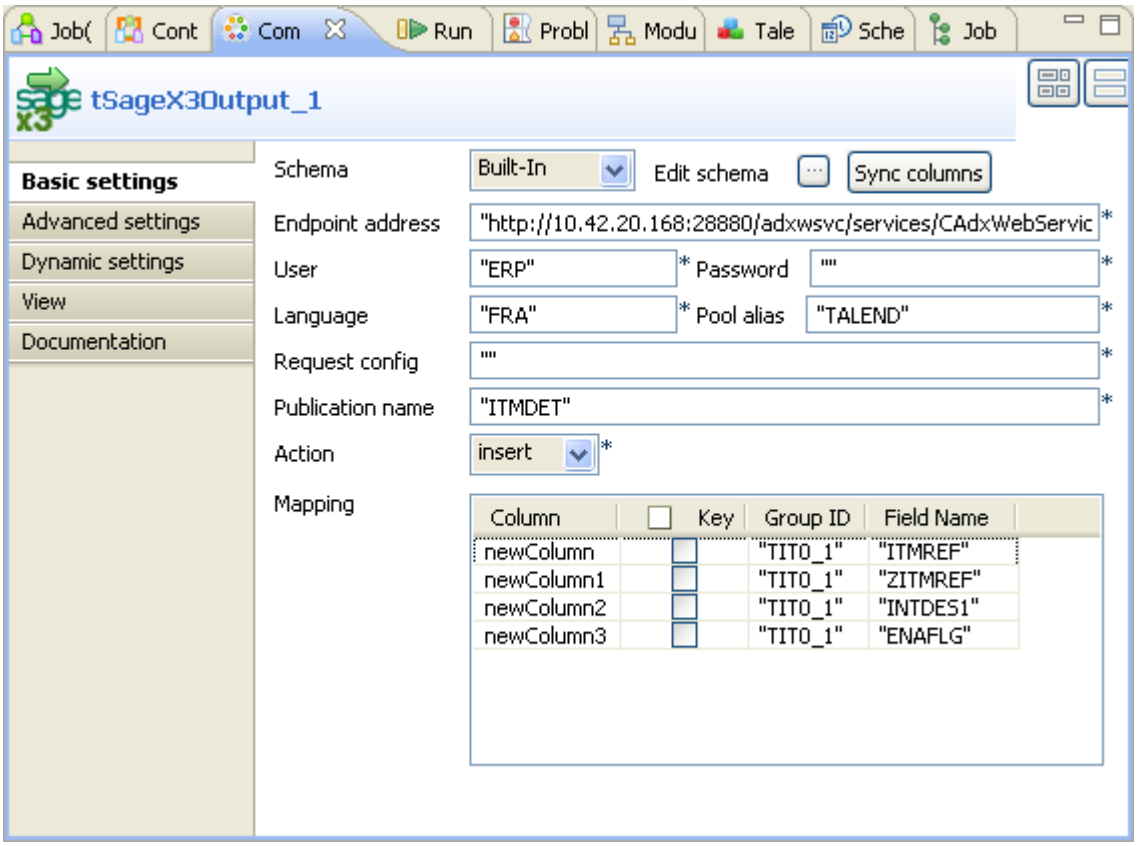
```
CONTS00059, Screen 24\" standard 16/10, Screen 24\" standard 28/10, 2
```

.

These values in the Value column must be put between quotation marks.

Setting up the connection to the Sage X3 Web server

1. Double-click **tSageX3Output** to set its properties from the **Basic Settings** view.



2. In the **Endpoint address** field, type in the URL address of the Web service provided by the Sage X3 Web server. In this example, it is *http://10.42.20.168:28880/adxwsvc/services/CAdxWebServiceXmlCC*

3. In the **User** field, type in the user name of the given Sage X3. In this example, it is *ERP*.

4. In the **Language** field, type in the name of the X3 language code used to start a connection group. In this example, it is *FRA*.

5. In the **Pool alias** field, type in the name of connection pool to be used. In this example, this connection pool is called *TALEND*.

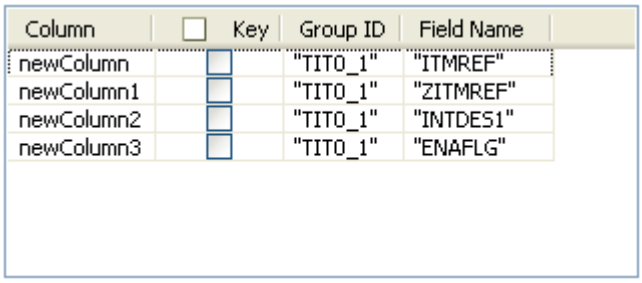
6. In the **Publication name** field, type in the publication name of the object to be called. In this scenario, the publication name is *ITMDET*.

7. In the **Action** field, select **insert** from the drop-down list.

Setting up the mapping

1. In the **Field name** column of the **Mapping** table, type in the field names of the attributes the selected data action is exercised on.

2. In the **Group ID** column of the **Mapping** table, type in values corresponding to group IDs of the selected attributes. These IDs are defined in the Sage X3 Web server



Column	Key	Group ID	Field Name
newColumn		"TIT0_1"	"ITMREF"
newColumn1		"TIT0_1"	"ZITMREF"
newColumn2		"TIT0_1"	"INTDES1"
newColumn3		"TIT0_1"	"ENAFLG"

In the **Mapping** table, the **Column** column has been filled automatically with the columns retrieved from the schema of the preceding component.

Job execution

Press CTRL+S to save your Job and press **F6** to execute it.

To verify the data that you inserted in this scenario, you can use the **tSageX3Input** component to read the concerned data from the Sage X3 server.

For further information about how to use the **tSageX3Input** component to read data, see section *Scenario: Using query key to extract data from a given Sage X3 system*.

tSalesforceBulkExec

tSalesforceBulkExec Properties

tSalesforceOutputBulk and **tSalesforceBulkExec** components are used together to output the needed file and then execute intended actions on the file for your Salesforce.com. These two steps compose the **tSalesforceOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Business/Cloud	
Function	**tSalesforceBulkExec** executes the intended actions on the prepared bulk data.	
Purpose	As a dedicated component, **tSalesforceBulkExec** gains performance while carrying out the intended data operations into your Salesforce.com.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appears allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Salesforce Version	Type in the version of the Salesforce you are using.
	Bulk file path	Directory where are stored the bulk data you need to process.
	Action	You can do any of the following operations on the data of the Salesforce object: **Insert**: insert data. **Update**: update data. **Upsert**: update and insert data.
	Upsert Key Column	Specify the key column for the upsert operation. Available when **Upsert** is selected from the **Action** list.
	Module	Select the relevant module in the list. if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Rows to commit*	Specify the number of lines per data batch to be processed.
	Bytes to commit	Specify the number of bytes per data batch to be processed.
	Use Socks Proxy	Select this check box if you want to use a proxy server. Once selected, you need provide the connection parameters that are host, port, username and password.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Used as an output component. An Input component is required.	
Limitation	The bulk data to be processed should be *.csv* format. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related Scenario:

For a related scenario, see section *Scenario: Inserting transformed bulk data into your Salesforce.com*.

tSalesforceConnection

tSalesforceConnection properties

Component family	Business/Cloud	
Function	**tSalesforceConnection** opens a connection to a Salesforce system in order to carry out a transaction.	
Putpose	The component enables connection to a Salesforce.	
Basic settings	*Salesforce Webservice URL*	Enter the Webservice URL required to connect to the Salesforce database.
	Username et *Password*	Enter your Web service authentication details.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com.
	For salesforce bulk component	Select this check box if you use bulk data processing components from the salesforce family. Once selected; the **Salesforce Version** field appears and therein you need to enter the Salesforce version you are using. For more information on these bulk data processing components, see section *tSalesforceOutputBulk*, section *tSalesforceBulkExec* and section *tSalesforceOutputBulkExec*.
	Use Soap Compression	Select this check box if you want to activate SOAP compression. The compression of SOAP messages results in increased performance levels.
	Use Socks Proxy	Select this check box if you want to use a proxy. Once selected, you need type in the connection parameters in the fields which appear. These parameters are the host, the port, the username and the password of the Proxy you need to use.
Advanced settings	*Client ID*	Set the ID of the real user to differentiate between those who use the same account and password to access the salesforce website.
	tStatCatcher Statistics	Select this check box to collect the log data at a component level.
Usage	This component is normally used with Salesforce components..	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For further information regarding the usage of **tSalesforceConnection**, see section *tMysqlConnection*.

tSalesforceGetDeleted

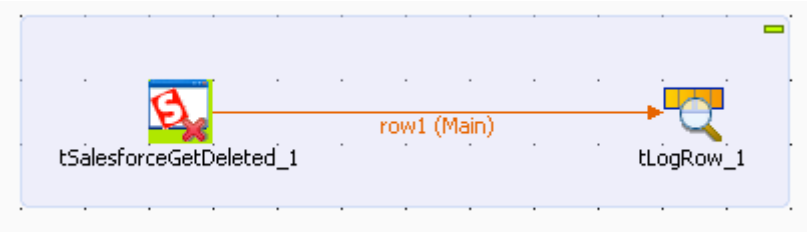
tSalesforceGetDeleted properties

Component family	Business/Cloud	
Function	**tSalesforceGetDeleted** recovers deleted data from a Salesforce object over a given period of time.	
Purpose	This component can collect the deleted data from a Salesforce object during a specific period of time.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com.
	Module	Select the relevant module in the list. If you select the **Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Start Date	Type in between double quotes the date at which you want to start the search. Use the following date format: “yyy-MM-dd HH:mm:ss”. You can do the search only on the past 30 days.
	End Date	Type in between double quotes the date at which you want to end the search. Use the following date format:“yyy-MM-dd HH:mm:ss”.
Advanced settings	*Use Soap Compression*	Select this check box to activate the SOAP compression. The compression of SOAP messages optimizes system performance.

	Client ID	Set the ID of the real user to differentiate between those who use the same account and password to access the Salesforce website.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetDeleted** requires an input component.	
Limitation	n/a	

Scenario: Recovering deleted data from the Salesforce server

This scenario describes a two-component Job that collects the deleted data over the past 5 days from the Salesforce server.

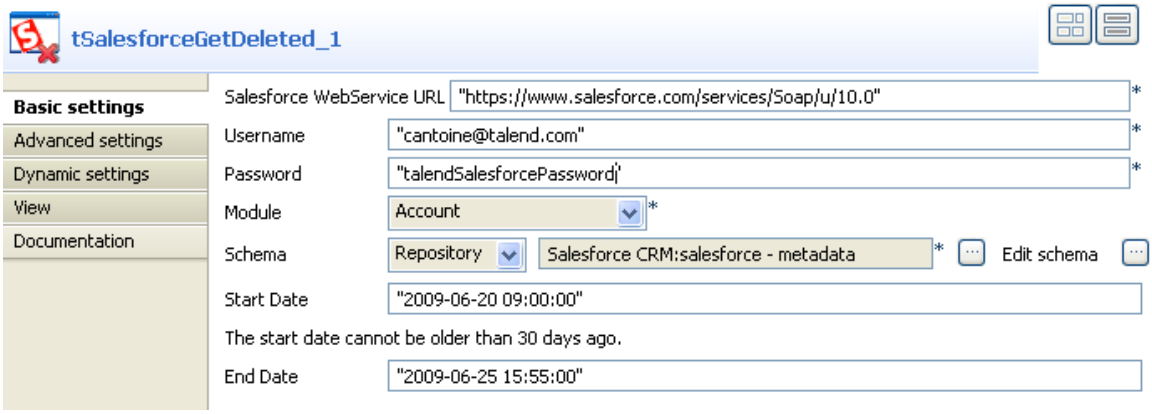


Setting up the Job

1. Drop **tSalesforceGetDeleted** and **tLogRow** from the **Palette** onto the design workspace.
2. Connect the two components together using a **Row** > **Main** connection.

Setting up the connection to the Salesforce server

1. Double-click **tSalesforceGetDeleted** to display its **Basic settings** view and define the component properties.



2. In the **Salesforce WebService URL** filed, use the by-default URL of the Salesforce Web service or enter the URL you want to access.
3. In the **Username** and **Password** fields, enter your login and password for the Web service.

4. From the **Module** list, select the object you want to access, **Account** in this example.

Setting the search condition

1. Click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually.
2. In the **Start Date** and **End Date** fields, enter respectively the start and end dates for collecting the deleted data using the following date format: "*yyyy-MM-dd HH:mm:ss*". You can collect deleted data over the past 30 days. In this example, we want to recover deleted data over the past 5 days.

Job execution

1. Double-click **tLogRow** to display its **Basic settings** view and define the component properties.
2. Click **Sync columns** to retrieve the schema from the preceding component.
3. In the **Mode** area, select **Vertical** to display the results in a tabular form on the console.
4. Press **Ctrl+S** to save your Job and press **F6** to execute it.

```
Starting job tSalesforceGetDeleted_scenario at 17:17 25/06/2009.

.-------------------------------------------------.
|                    #1. tLogRow_1                |
+----------------------+--------------------------+
| key                  | value                    |
+----------------------+--------------------------+
| Id                   | 00170000003smNAAR        |
| IsDeleted            | true                     |
| MasterRecordId       | null                     |
| Name                 | sForce                   |
| Type                 | null                     |
| ParentId             | null                     |
| BillingStreet        | The Landmark at One Market |
| BillingCity          | San Francisco            |
| BillingState         | CA                       |
| BillingPostalCode    | 94087                    |
| BillingCountry       | US                       |
| ShippingStreet       | null                     |
| ShippingCity         | null                     |
| ShippingState        | null                     |
| ShippingPostalCode   | null                     |
| ShippingCountry      | null                     |
| Phone                | (415) 901-7000           |
| Fax                  | (415) 901-7002           |
| AccountNumber        | CX355105                 |
| Website              | www.sforce.com           |
| Sic                  | null                     |
| Industry             | null                     |
| AnnualRevenue        | 0.0                      |
| NumberOfEmployees    | null                     |
| Ownership            | null                     |
| TickerSymbol         | null                     |
| Description          | null                     |
| Rating               | null                     |
```

Deleted data collected by the **tSalesforceGetDeleted** component is displayed in a tabular form on the console.

tSalesforceGetServerTimestamp

tSalesforceGetServerTimestamp properties

Component family	Business/Cloud	
Function	**tSalesforceGetServerTimestamp** retrieves the current date of the Salesforce server.	
Purpose	This component retrieves the current date of the Salesforce server presented in a timestamp format.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Use Socks Proxy*	Select this check box if you want to use a proxy server Once selected, you need enter the connection parameters that are the host, the port, the username and the passerword of the Proxy you need to use.
	Use Soap Compression	Select this check box to activate the SOAP compression. The compression of the SOAP messages optimizes system performance.
	Client ID	Set the ID of the real user to differentiate between those who use the same account and password to access the salesforce website.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetServerTimestamp** requires an input component.	
Limitation	n/a	

Related scenarios

No scenario is available for this component yet.

tSalesforceGetUpdated

tSalesforceGetUpdated properties

Component family	Business/Cloud	
Function	**tSalesforceGetUpdated** recovers updated data from a Salesforce object over a given period of time.	
Purpose	This component can collect all updated data from a given Salesforce object during a specific period of time.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the Web service URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Web service user authentication data.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com.
	Module	Select the relevant module in the list. if you select the **Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Start Date	Type in between double quotes the date at which you want to start the search. Use the following date format: “yyy-MM-dd HH:mm:ss”. You can do the search only on the past 30 days.
	End Date	Type in between double quotes the date at which you want to end the search. Use the following date format:“yyy-MM-dd HH:mm:ss”.
Advanced settings	*Use Soap Compression*	Select this check box to activate the SOAP compression. The compression of SOAP messages optimizes system performance.

	Client ID	Set the ID of the real user to differentiate between those who use the same account and password to access the Salesforce website.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetUpdate** requires an input component.	
Limitation	n/a	

Related scenarios

No scenario is available for this component yet.

tSalesforceInput

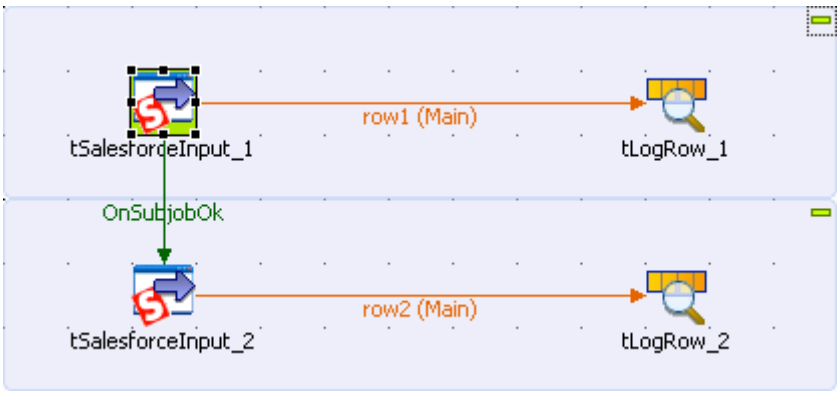
tSalesforceInput Properties

Component family	Business/Cloud	
Function	**tSalesforceInput** connects to an object of a Salesforce database via the relevant Web service.	
Purpose	Allows to extract data from a Salesforce DB based on a query.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Query mode	Two options are available: **Query** and **Bulk Query**.
	Salesforce Webservice URL	Type in the Web service URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Web service user authentication data.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com. Available when **Query** is selected from the **Query mode** list.
	Module	Select the relevant module in the list. If you select the **Custom Module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. In this component the schema is related to the Module selected. *To retrieve a column from a linked module it is necessary to define the column in a particular manner in the **Edit schema** view, otherwise the relationship query will not work. The correct syntax is:* `NameofCurrentModule_Nameof-` `LinkedModule_NameofColumnof-`

		`Interest`
	Query condition	Type in the query to select the data to be extracted. Example: `account_name= 'Talend'`
	Maunal input of SOQL query	Select this check box to display the **Query** field where you can manually enter the desired query.
	Query all records (include deleted records)	Select this check box to query all the records, including the deletions. Available when **Query** is selected from the **Query mode** list.
Advanced settings	*Batch Size*	Number of registrations in each processed batch. Available when **Query** is selected from the **Query mode** list.
	Use Socks Proxy	Select this check box if you want to use a proxy server. Once selected, you need enter the connection parameters that are the host, the port, the username and the password of the Proxy you need to use.
	Normalize delimiter (for child relationship)	Characters, strings or regular expressions used to normalize the data that is collected by queries set on different hierarchical Salesforce objects. Available when **Query** is selected from the **Query mode** list.
	Column name delimiter (for child relationship)	Characters, strings or regular expressions used to separate the name of the parent object from the name of the child object when you use a query on the hierarchical relations among the different Salesforce objects. Available when **Query** is selected from the **Query mode** list.
	Use Soap Compression	Select this check box to activate the SOAP compression. The compression of SOAP messages optimizes system performance, in particular for the batch operations.
	Output Http Trace Message	Select this check box to output the HTTP trace message. Available when **Bulk Query** is selected from the **Query mode** list.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
	Client ID	Set the ID of the real user to differentiate between those who use the same account and password to access the Salesforce website. Available when **Query** is selected from the **Query mode** list.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario: Using queries to extract data from a Salesforce database

This scenario describes a four-component Job used to extract specific sets of data from parent and child objects in a Salesforce database.

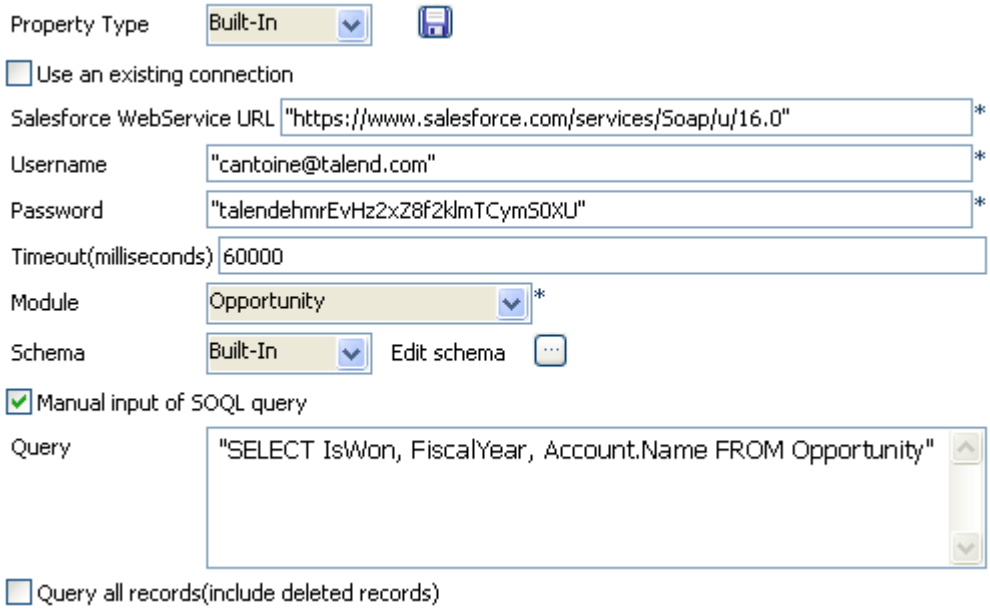


Setting up the Job

1. Drop two **tSalesforceInput** components and two **tLogRow** components onto the workspace.
2. Connect each **tSalesforceInput** component to a **tLogRow** component using a **Row** > **Main** connection for each pair.
3. Connect **tSalesforceInput_1** to **tSalesforceInput_2** using an **OnSubjobOk** connection.

Setting up the connection to the Salesforce server for the parent object

1. Double-click **tSalesforceInput_1** to set its **Basic Settings** in the **Component** tab.

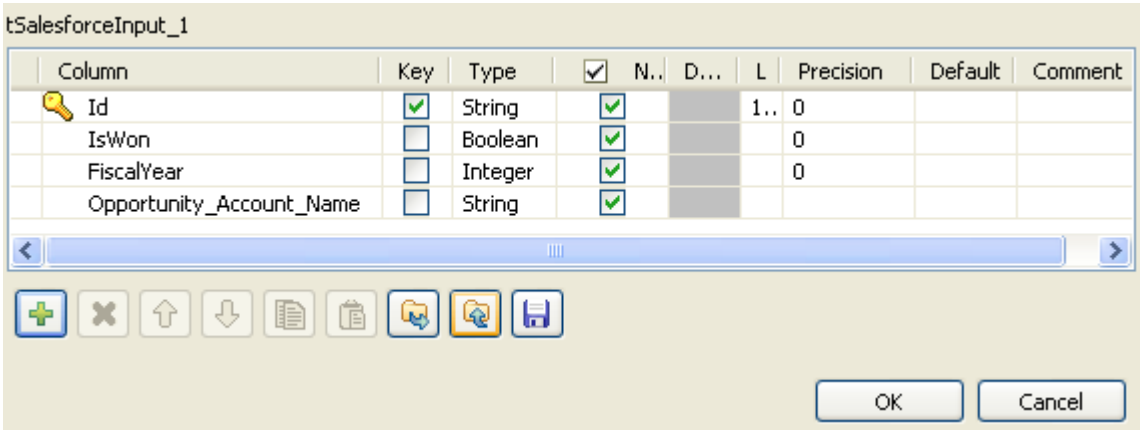


2. Enter the **Salesforce WebService URL** of the database you want to connect to in the corresponding field.
3. Enter your authentication information in the corresponding **Username** and **Password** fields.
4. Enter the desired query **Timeout (milliseconds)** limit.

Setting the query and the schema for the parent object

1. Select the **Module** (salesforce object) you want to query.
2. Select the **Manual input of SOQL Query** check box and enter your query scripts in the enabled **Query** field.

   The query scripts you enter should follow the SOQL syntax.
3. Click **[...]** next to **Edit schema** to open the schema editor.



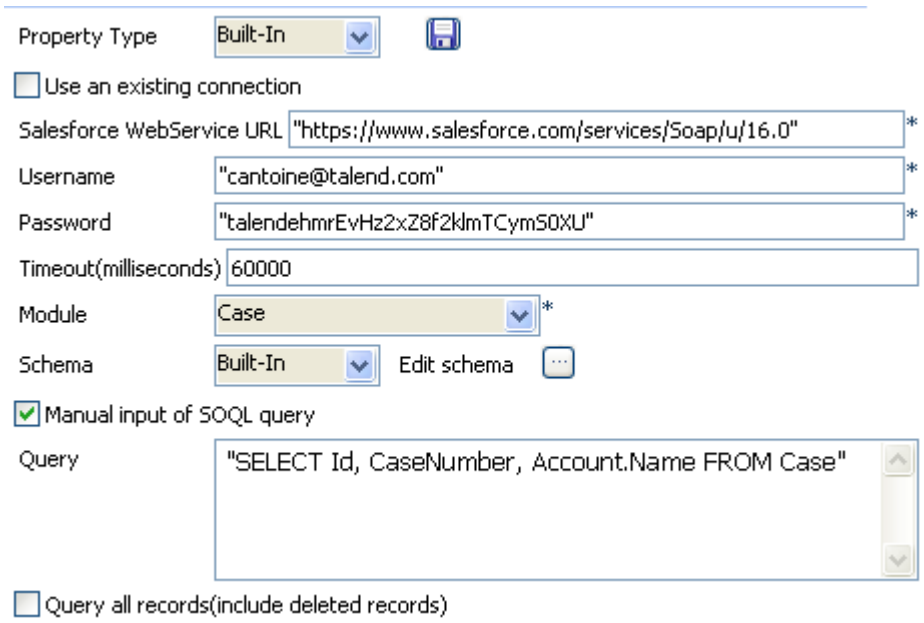
   In this example, the *IsWon* and *FiscalYear* columns in the query are located in the **Opportunity** module specified. The *Name* column is in a linked module called *Account*. To return a column from a linked module the correct syntax is to enter the name of the linked module, followed by the period character, then the name of the column of interest. Hence, the query required in this example is:

   ```
   "SELECT IsWon, FiscalYear, Account.Name FROM Opportunity".
   ```
4. Click the plus button to add a new column for the fields taken from the *Name* column in the *Account* module.
5. Name this column *Opportunity_Account_Name* and click **OK** to save the changes.

   *To retrieve a column from a linked module, it is necessary to define the column in a particular manner in the **Edit schema** view. The correct syntax is: `NameofCurrentModule_NameofLinkedModule_NameofColumnofInterest`. Hence, in this example, the column must be named: Opportunity_Account_Name. If this syntax is not respected then the data from the linked table will not be returned.*

Setting up the connection to the Salesforce server for the child object

1. Double-click **tSalesforceInput_2** to set its **Basic settings** in the **Component** tab.



2. Enter the **Salesforce WebService URL** of the database you want to connect to in the corresponding field.

   The query scripts you enter must follow the SOQL syntax.

3. Enter your authentication information in the corresponding **Username** and **Password** fields.

4. Enter the desired query **Timeout (milliseconds)** limit.

Setting the query and the schema for the child object

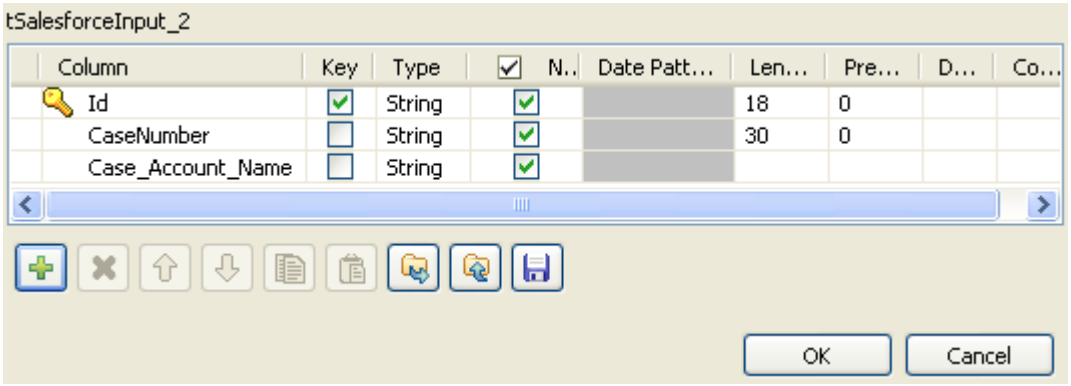
1. Select the **Module** (salesforce object) you want to query.

2. Select the **Manual input of SOQL Query** check box and enter your query scripts in the enabled **Query** field.

   In this example we want to extract the *Id* and *CaseNumber* fields from the *Case* module as well as the *Name* fields from the *Account* module. The query is therefore: .

   ```
   “SELECT Id, CaseNumber, Account.Name FROM Case”
   ```

3. Click **[...]** next to **Edit schema** to open the schema editor.



4. Click the plus button to add a new column for the fields taken from the *Name* column in the *Account* module.

5. Name this column *Case_Account_Name* and click **OK** to save the changes.

Job execution

1. Click each **tLogRow** component and set their component properties in the **Basic settings** view as desired.

   In this example, there is no need to modify the **tLogRow** settings.

2. Press **Ctrl+S** to save your Job and press **F6** to execute it.

   The results are displayed in the **Run** tab:

```
|true|2007|GenePoint
|true|2007|Burlington Textiles Corp of America
|true|2007|United Oil & Gas Corp.
|false|2007|United Oil & Gas Corp.
|true|2007|United Oil & Gas Corp.
|false|2007|United Oil & Gas Corp.
|true|2007|United Oil & Gas Corp.
|true|2007|United Oil & Gas Corp.
|false|2007|GenePoint
|true|2007|GenePoint
|true|2007|Edge Communications
|false|2007|Edge Communications
|true|2007|Edge Communications
|false|2007|Pyramid Construction Inc.
|false|2007|Dickenson plc
|true|2007|Grand Hotels & Resorts Ltd
|true|2007|Grand Hotels & Resorts Ltd
|false|2007|Grand Hotels & Resorts Ltd
500700000005DBELAA4|00001000|Edge Communications
500700000005DBEMAA4|00001001|United Oil & Gas Corp.
500700000005DBENAA4|00001002|United Oil & Gas Corp.
500700000005DBEOAA4|00001003|Express Logistics and Transport
500700000005DBEPAA4|00001004|Express Logistics and Transport
500700000005DBEQAA4|00001005|Express Logistics and Transport
500700000005DBERAA4|00001006|GenePoint
500700000005DBESAA4|00001007|Grand Hotels & Resorts Ltd
500700000005DBETAA4|00001008|Grand Hotels & Resorts Ltd
500700000005DBEUAA4|00001009|United Oil & Gas, UK
500700000005DBEVAA4|00001010|United Oil & Gas, Singapore
```

tSalesforceOutput

tSalesforceOutput Properties

Component family	Business/Cloud	
Function	**tSalesforceoutput** writes in an object of a Salesforce database via the relevant Web service.	
Purpose	Allows to write data into a Salesforce DB.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data is stored centrally.
	Use an existing connection	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the Web service URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Web service user authentication data.
	Timeout (milliseconds)	Type in the intended number of query timeout in Salesforce.com.
	Action	You can do any of the following operations on the data of the Salesforce object: **Insert**: insert data. **Update**: update data. **Delete**: delete data. **Upsert**: update and insert data.
	Upsert Key Column	Specify the key column for the upsert operation. Available when **Upsert** is selected from the **Action** list.
	Module	Select the relevant module in the list. if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.

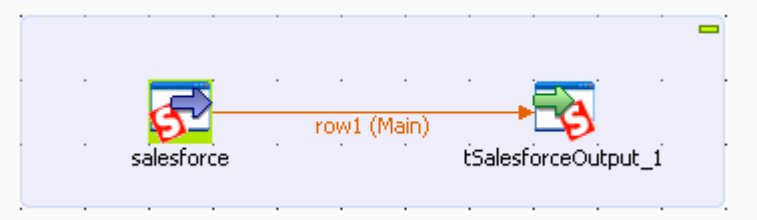
		Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Extended Output*	This check box is selected by default. It allows to transfer output data in batches. You can specify the number of lines per batch in the **Rows to commit** field.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link. The **Reject** link is available only when you have deselected the **Extended Output** and **Die on error** check boxes.
	Error logging file	If you want to create a file that holds all error logs, click the three-dot button next to this field and browse to the specified file to set its access path and its name.
	Use Socks Proxy	Select this check box if you want to use a proxy server. Once selected, you need enter the connection parameters that are the host, the port, the username and the passerword of the Proxy you need to use.
	Ignore NULL fields values	Select this check box to ignore NULL values in **Update** or **Upsert** mode.
	Use Soap Compression	Select this check box to activate the SOAP compression. The compression of SOAP messages optimizes system performance.
	Retrieve inserted ID	Select this check box to allow Salesforce.com to return the salesforce ID produced for a new row that is to be inserted. The ID column is added to the processed data schema in Salesforce.com. This option is available only when you have chosen **insert** action yet not in batch mode, i.e. not in the **Extended Output** option.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
	Client ID	Set the ID of the real user to differentiate between those who use the same account and password to access the salesforce website.
	Relationship mapping for upsert	Click the **[+]** button to add lines as needed and specify the external ID fields in the input flow, the upsert module and the lookup module as well as the lookup module. **Column name of Talend schema**: external ID field in the input flow. **Lookup field name**: external ID field in the upsert module. **Module name**: name of the lookup module. **External id name**: external ID field in the lookup module.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Scenario 1: Deleting data from the Account object

This scenario describes a two-component Job that removes an entry from the Account object.

Dragging and dropping as well as connecting the components

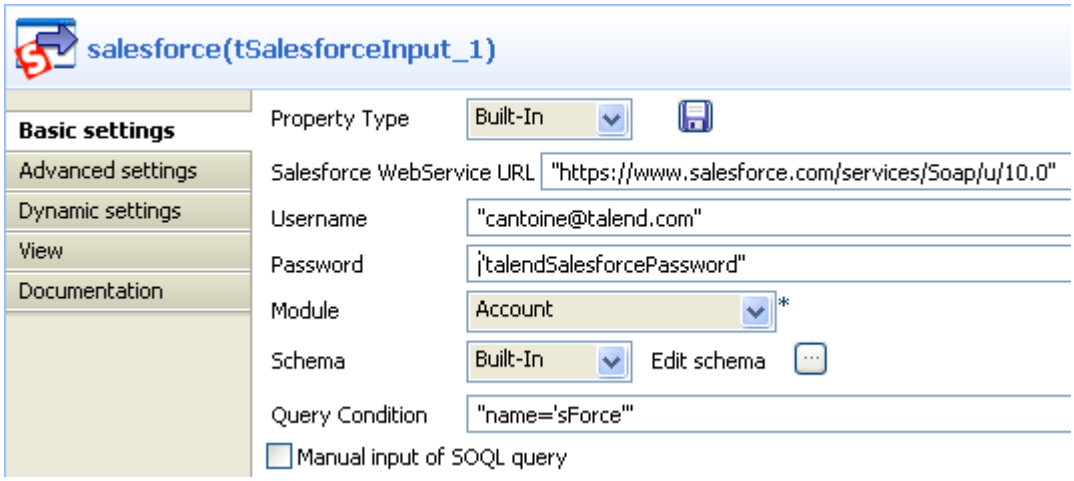
1. Drop **tSalesforceInput** and **tSalesforceOutput** from the **Palette** onto the design workspace.
2. Connect the two components together using a **Row** > **Main** link.



Configuring the components

Querying the content to be deleted

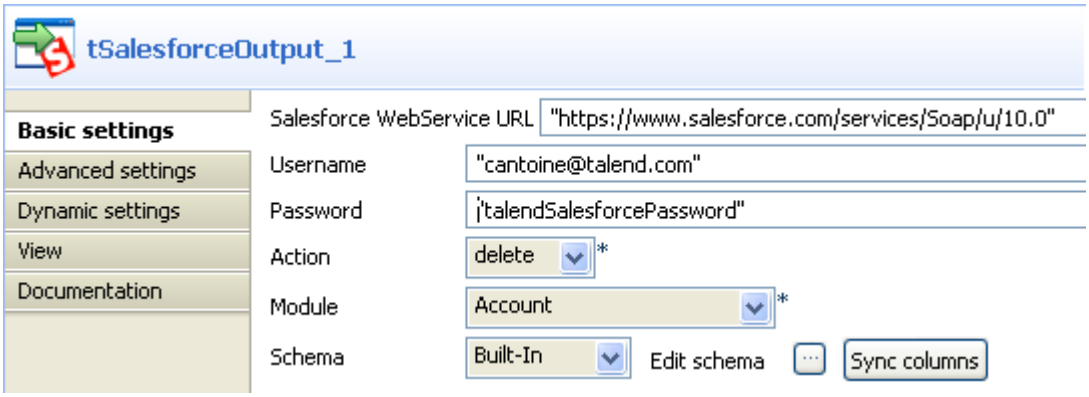
1. Double-click **tSalesforceInput** to display its **Basic settings** view and define the component properties.



2. In the **Salesforce WebService URL** field, use the default URL of the Salesforce Web service or enter the URL you want to access or select the **Use an existing connection** check box to use an established connection.
3. In the **Username** and **Password** fields, enter your login and password for the Web service.
4. Type in your intended query timeout in the **Timeout (milliseconds)** field. In this example, use the default number.
5. From the **Module** list, select the object you want to access, **Account** in this example.
6. Click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually.
7. In the **Query Condition** field, enter the query you want to apply. In this example, we want to retrieve the clients whose names are *sForce*. To do this, we use the query: `"name='sForce'"`.
8. For a more advanced query, select the Manual input of SOQL query and enter the query manually.

Deleting the queried contents

1. Double-click **tSalesforceOutput** to display its **Basic settings** view and define the component properties.



2. In the **Salesforce WebService URL** field, use the default URL of the Salesforce Web service or enter the URL you want to access.
3. In the **Username** and **Password** fields, enter your login and password for the Web service.
4. Type in your intended query timeout in the **Timeout (milliseconds)** field. In this example, use the default number.
5. From the **Action** list, select the operation you want to carry out. In this example we select **Delete** to delete the *sForce* account selected in the previous component.
6. From the **Module** list, select the object you want to access, **Account** in this example.
7. Click **Sync columns** to retrieve the schema of the preceding component.
8. Press **Ctrl+S** to save your Job.

Executing the Job

- Press **F6** to execute the Job.

  Check the content of the **Account** object and verify that the *sForce* account(s) is/are deleted from the server.

Scenario 2: Gathering erroneous data while inserting data to a module at Salesforce.com

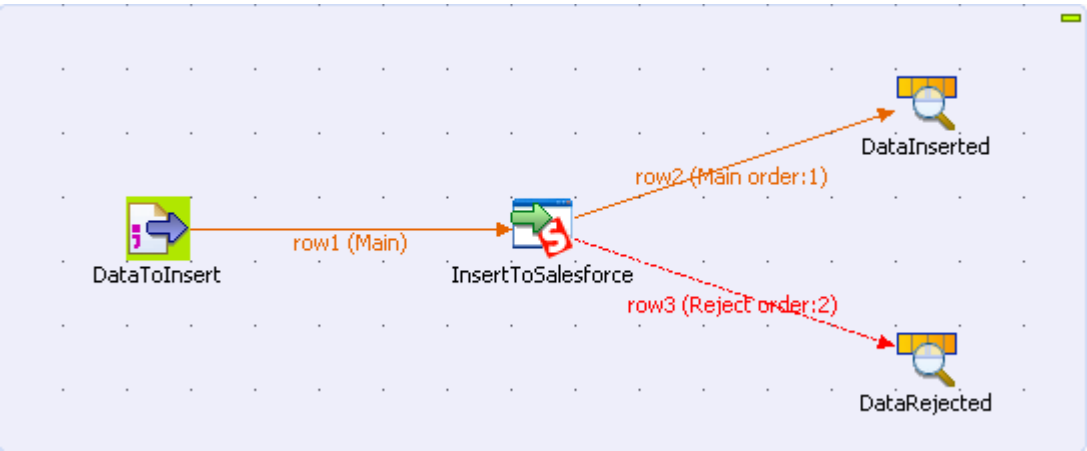
In this scenario, data in a local file is inserted to the **AdditionalNumber** module. Meanwhile, erroneous data in that file is collected via a **Row** > **Reject** link.

Dragging and dropping components and linking them together

1. Drag and drop the following components from the **Palette** onto the workspace: **tFileInputDelimited**, **tSalesforceOutput** and two **tLogRow** components.
2. Rename **tFileInputDelimited** as **DataToInsert**, **tSalesforceOutput** as **InsertToSalesforce**, and the two **tLogRow** components as **DataInserted** as well as **DataRejected** respectively.

3. Link **DataToInsert** to **InsertToSalesforce** using a **Row** > **Main** connection.
4. Link **InsertToSalesforce** to **DataInserted** using a **Row** > **Main** connection.
5. Link **InsertToSalesforce** to **DataRejected** using a **Row** > **Reject** connection.

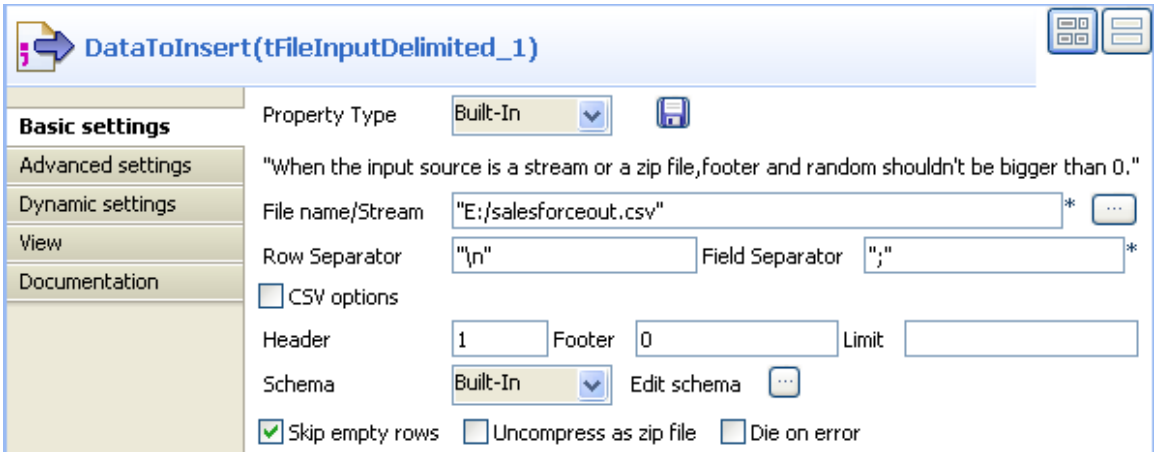
Deselect the **Extended Output** and **Die on error** check boxes in the **Advanced settings** view of the **tSalesforceOutput** component so that the **Reject** link is available .



Configuring the components

Configuring the data source

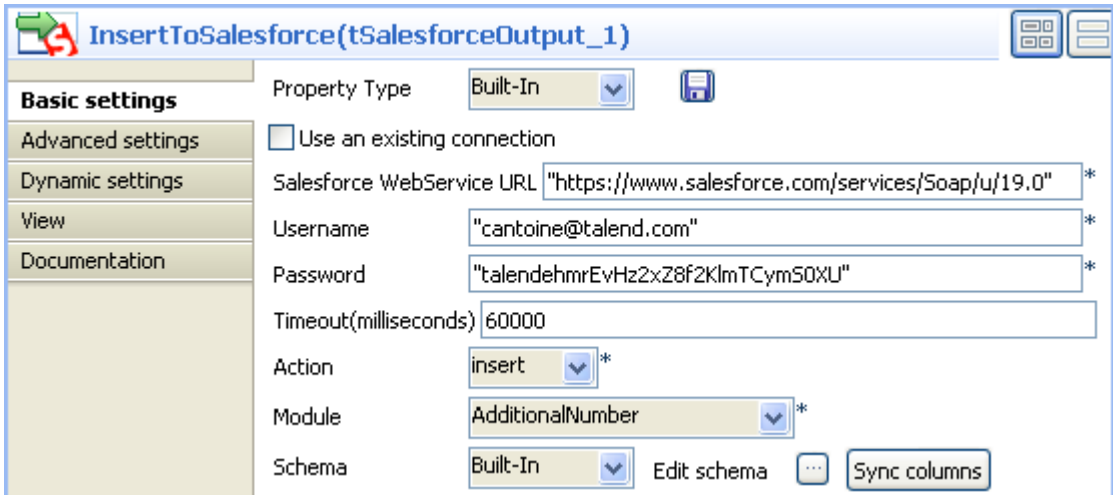
1. Double-click **DataToInsert** to open its **Basic settings** view in the **Component** tab.



2. In the **File name/Stream** field, type in the path of the source file, for example, *E:/salesforceout.csv*.
3. In the **Header** field, type in *1* to retrieve the column names. Keep the default settings for other fields.

Configuring the module for data insertion

1. Double-click **InsertToSalesforce** to open its **Basic settings** view in the **Component** tab.



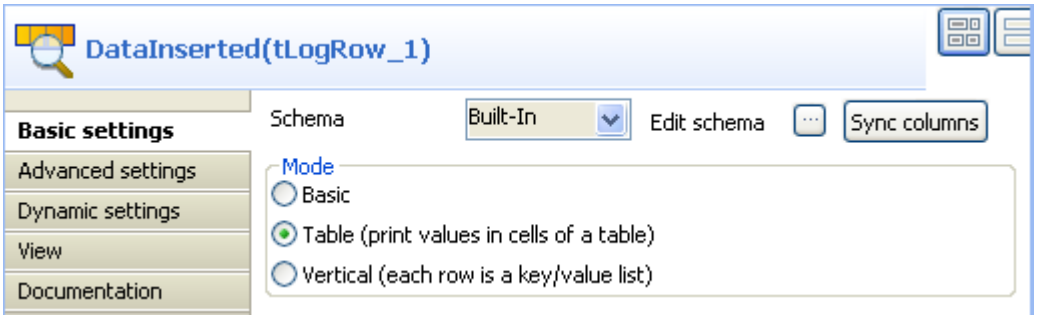
2. In the **Username** field, enter your username, for example, *cantoine@talend.com*.
3. In the **Password** field, enter your password, for example, *talendehmrEvHz2xZ8f2KlmTCymS0XU*.
4. In the **Action** drop-down list, select **insert**.
5. In the **Module** drop-down list, select **AdditionalNumber**.

> When linking the components earlier, the **Extended Output** and **Die on error** check boxes have been deselected in the **Advanced settings** view so that the **Reject** link can appear.

6. Keep the default settings for other fields.

Configuring the console display

1. Double-click **DataInserted** to open its **Basic settings** view in the **Component** tab.



2. In the **Mode** area, select **Table (print values in cells of a table)** for a better view.
3. Perform the same operation for **DataRejected**.
4. Press **Ctrl+S** to save your Job.

Executing the Job

- Press **F6** to run the Job and you can find the erroneous data (if any) is displayed in the **Run** view.

```
|
tLogRow_2
|
|=-----------------+-----------------+---------------------+-------------
---+-------------------------+-----------+--------------------------------
-----------------------------=|
|CallCenterId       |Name               |Description          |Phone
|errorCode                |errorFields |errorMessage
|
|=-----------------+-----------------+---------------------+-------------
---+-------------------------+-----------+--------------------------------
-----------------------------=|
|03m70000000CaZBAA0|Talend Testing V19|Conduct testing on
TOS|0086-010-88888888|FIELD_INTEGRITY_EXCEPTION|CallCenterId|Call Center ID:
id value of incorrect type: 03m70000000CaZBAA0|
|04m70000000CaZBAA0|Talend Testing V19|Conduct testing on
TOS|0086-010-88888888|FIELD_INTEGRITY_EXCEPTION|CallCenterId|Call Center ID:
id value of incorrect type: 04m70000000CaZBAA0|
'-----------------+-----------------+---------------------+-------------
---+-------------------------+-----------+--------------------------------
-----------------------------'
```

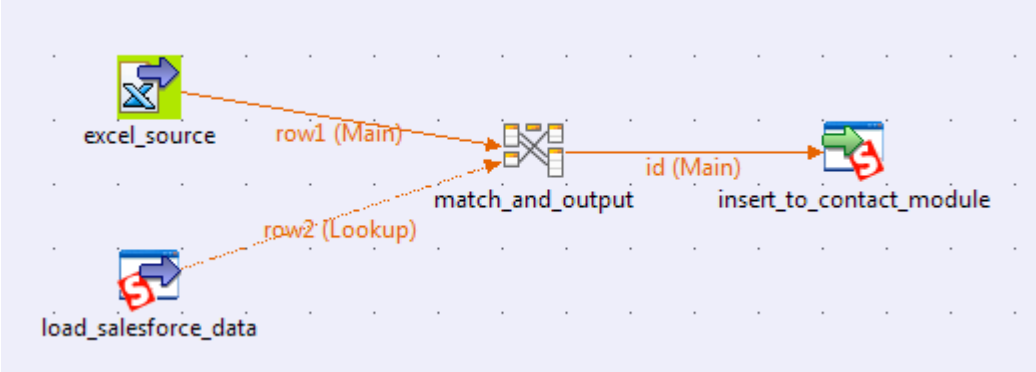
As shown above, there are two **Call Center ID** fields that have incorrect data.

Scenario 3: Inserting AccountIDs from an Excel File to the Contact Module

In this scenario, the AccountIDs from an excel file are inserted to the **Contact** module at the `www.salesforce.com` based on the matching of *LastName* and *Name* fields.

Dragging and dropping components

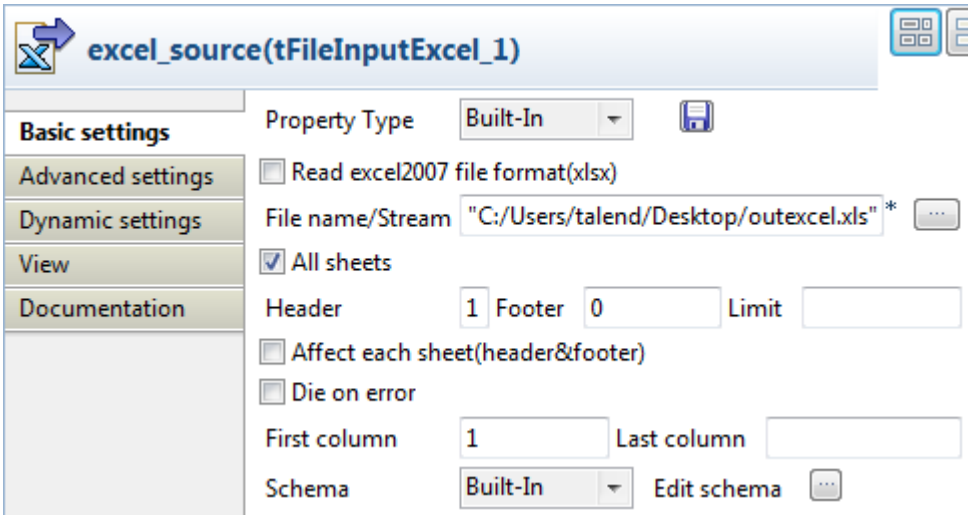
1. Drag and drop the following components from the **Palette** onto the workspace: **tFileInputExcel**, **tSalesforceIntput**, **tMap** and **tSalesforceOutput**.
2. Rename **tFileInputExcel** as **excel_source**, **tSalesforceIntput** as **load_salesforce_data**, **tMap** as **match_and_output** and **tSalesforceOutput** as **insert_to_contact_module**.
3. Link the components using a **Row** > **Main** connection.



Configuring the components

Configuring the source excel input

1. Double-click **excel_source** to open its **Basic settings** view in the **Component** tab.



2. Click the **[...]** button next to the **File name/Stream** field to select the source file.

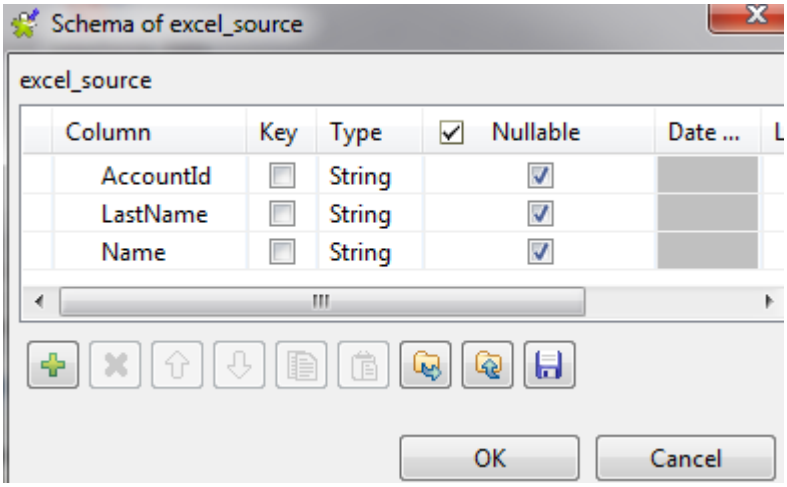
   The content looks like:

AccountId	LastName	Name
001A000000Ho08zIAB	Gonzalez	Gonzalez Rose
001A000000Ho090IAB	Forbes	Forbes Sean
001A000000Ho096IAB	Rogers	Rogers Jack

3. Select the **All sheets** check box to retrieve the data of the entire excel file.

4. Enter *1* in the **Header** field as the first line lists the column names.

5. Click the **[...]** button next to the **Edit schema** field to open the schema editor.

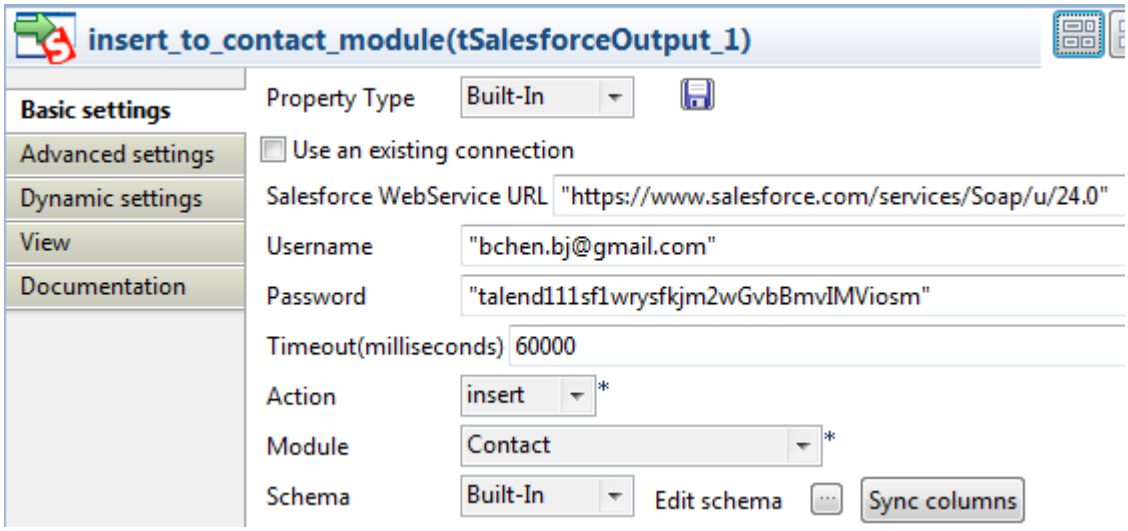


6. Click the **[+]** button to add three columns, i.e. *AccountId*, *LastName* and *Name*.

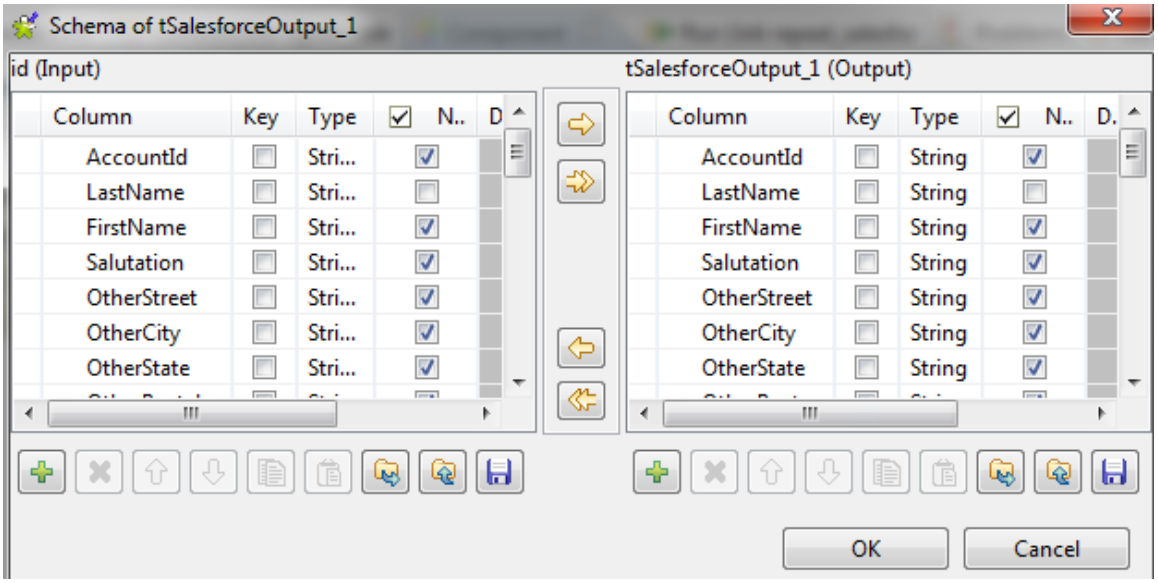
7. Click **OK** to close the editor. Keep other default settings as they are.

Configuring the destination module and the desired operation

1. Double-click **insert_to_contact_module** to open its **Basic settings** view in the **Component** tab.



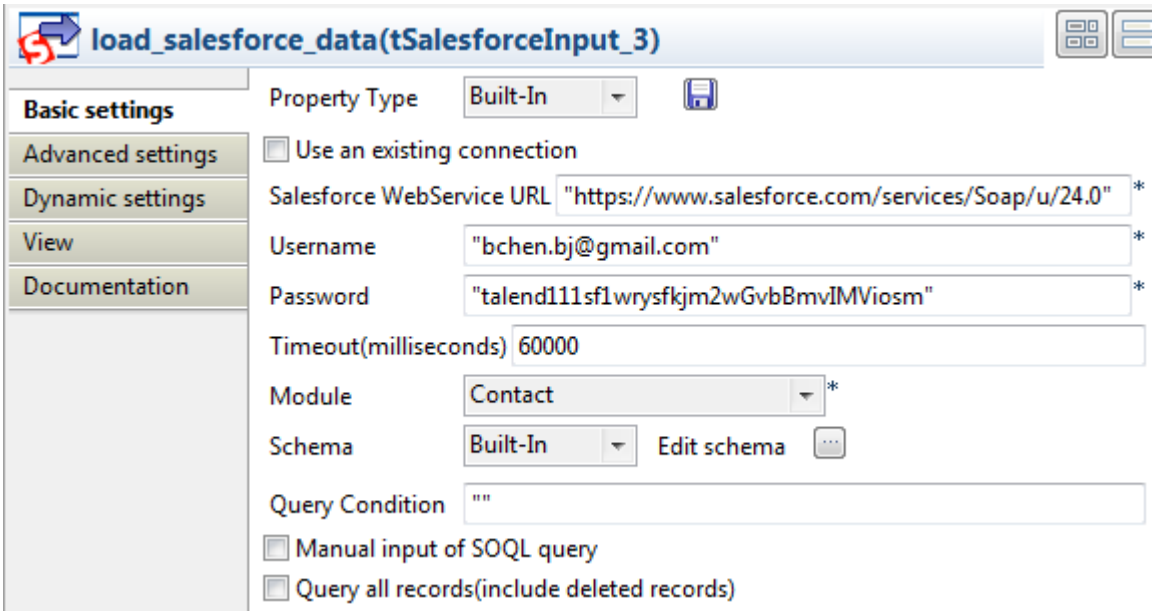
2. In the **Username** and **Password** fields, enter your authentication credentials.
3. Select **insert** in the **Action** list and **Contact** in the **Module** list.
4. Click the **[...]** button next to **Edit schema** to open the schema editor.



5. Click to copy all the columns from the output table to the input table.
6. Click **OK** to close the editor.

Configuring the lookup source and establishing the mapping relations

1. Double-click **load_salesforce_data** to open its **Basic settings** view in the **Component** tab.

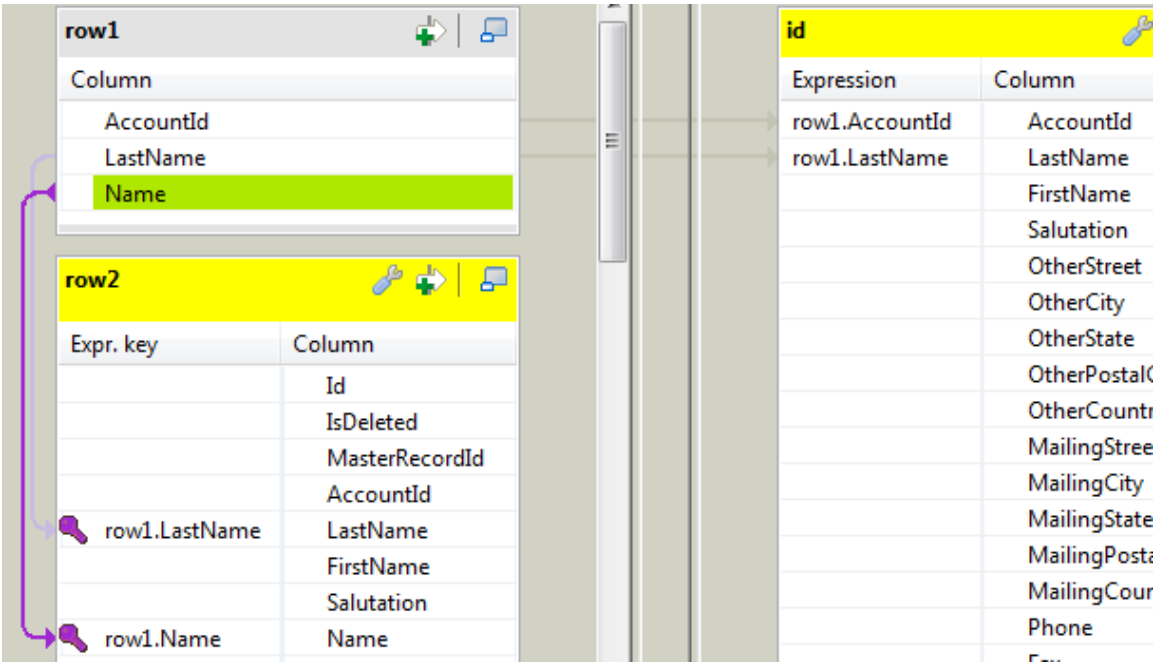


2. In the **Username** and **Password** fields, enter your authentication credentials.

3. In the **Module** list, select the **Contact** module to retrieve data.

4. Clear the **Query Condition** field.

5. Double-click **match_and_output** to open its map editor.



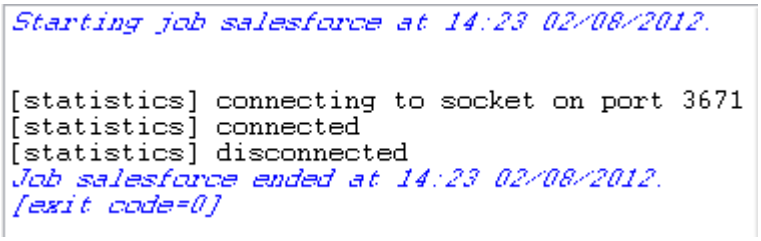
6. Select fields **LastName** and **Name** from the table *row1* and drop them next to their counterparts in the table *row2*. This way, data from the excel file will be checked against their counterparts in the **Contact** module.

7. Select fields **LastName** and **AccountID** from the table *row1* and drop them next to their counterparts in the table *id*. This way, qualified data from the excel file will be passed to their counterpart fields in the *id* table.

8. Click **OK** to close the map editor.

Executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** to run the Job.



tSalesforceOutputBulk

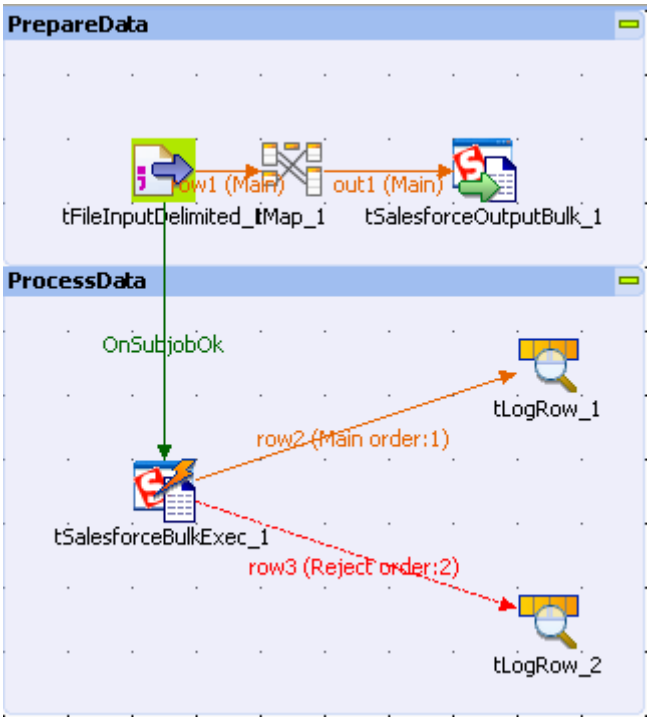
tSalesforceOutputBulk Properties

tSalesforceOutputBulk and **tSalesforceBulkExec** components are used together to output the needed file and then execute intended actions on the file for your Salesforce.com. These two steps compose the **tSalesforceOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Business/Cloud	
Function	**tSalesforceOutputBulk** generates files in suitable format for bulk processing.	
Purpose	Prepares the file to be processed by **tSalesforceBulkExec** for executions in Salesforce.com.	
Basic settings	*File Name*	Type in the directory where you store the generated file.
	Append	Select the check box to write new data at the end of the existing data. Or the existing data will be overwritten.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Ignore NULL fields values	Select this check box to ignore NULL values in **Update** or **Upsert** mode.
Advanced settings	*Relationship mapping for upsert*	Click the **[+]** button to add lines as needed and specify the external ID fields in the input flow, the upsert module and the lookup module. **Column name of Talend schema**: external ID field in the input flow. **Lookup field name**: external ID field in the upsert module. **External id name**: external ID field in the lookup module.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is intended for the use along with **tSalesforceBulkExec** component. Used together they gain performance while feeding or modifying information in Salesforce.com.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Inserting transformed bulk data into your Salesforce.com

This scenario describes a six-component Job that transforms *.csv* data suitable for bulk processing, load them in Salesforce.com and then displays the Job execution results in the console.



This Job is composed of two steps: preparing data by transformation and processing the transformed data.

Before starting this scenario, you need to prepare the input file which offers the data to be processed by the Job. In this use case, this file is *sforcebulk.txt*, containing some customer information.

Then to create and execute this Job, operate as follows:

Setting up the Job

1. Drop **tFileInputDelimited**, **tMap**, **tSalesforceOutputBulk**, **tSalesforceBulkExec** and **tLogRow** from the **Palette** onto the workspace of your studio.

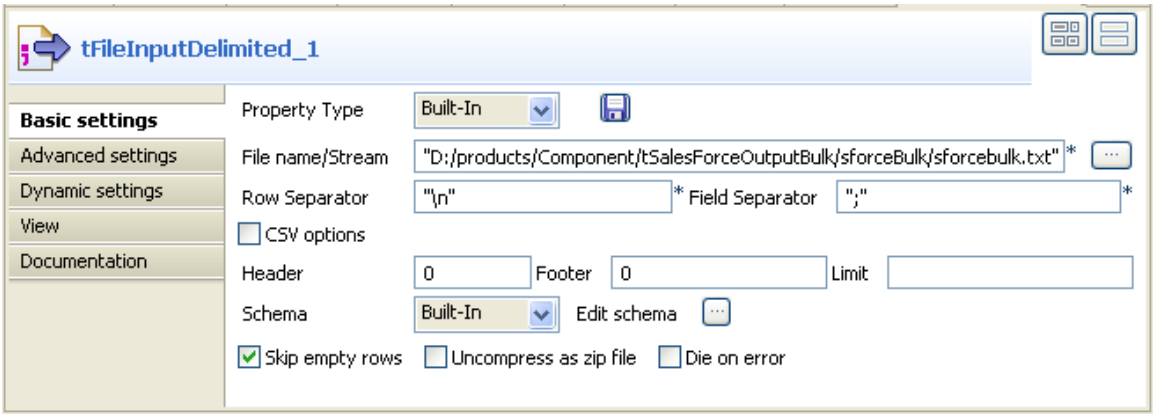
2. Use a **Row** > **Main** connection to connect **tFileInputDelimited** to **tMap**, and **Row** > **out1** from **tMap** to **tSalesforceOutputBulk**.

3. Use a **Row** > **Main** connection and a **Row** > **Reject** connection to connect **tSalesforceBulkExec** respectively to the two **tLogRow** components.

4. Use a **Trigger** > **OnSubjobOk** connection to connect **tFileInputDelimited** and **tSalesforceBulkExec**.

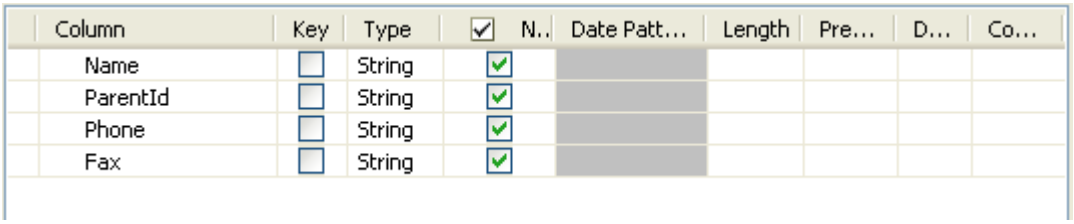
Configuring the input component

1. Double-click **tFileInputDelimited** to display its **Basic settings** view and define the component properties.



2. Next to the **File name/Stream** field, click the **[...]** button to browse to the input file you prepared for the scenario, for example, *sforcebulk.txt*.

3. Click the three-dot button next to the **Edit schema** field to open the dialog box to set the schema. In this scenario, the schema is made of four columns: *Name*, *ParentId*, *Phone* and *Fax*.

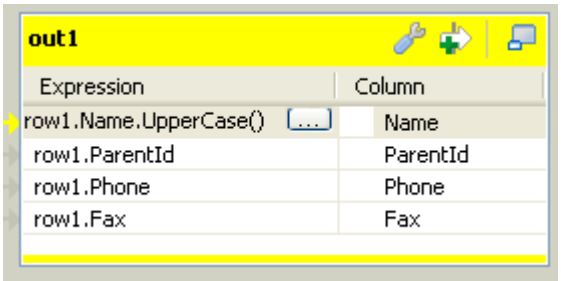


4. According to your input file to be used by the Job, set the other fields like **Row Separator**, **Field Separator**...

Setting up the mapping

1. Double-click the **tMap** component to open its editor and set the transformation.

2. Drop all columns from the input table to the output table.

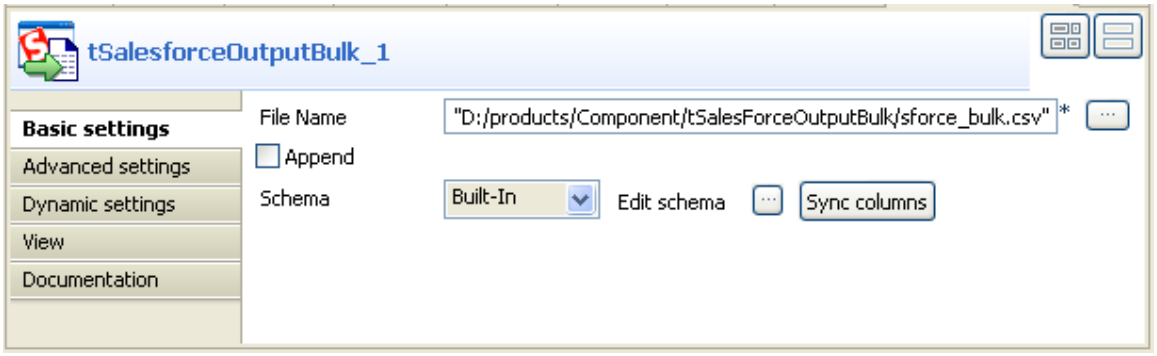


3. Add `.toUpperCase()` behind the *Name* column.

4. Click **OK** to validate the transformation.

Defining the output path

1. Double-click **tSalesforceOutputBulk** to display its **Basic settings** view and define the component properties.

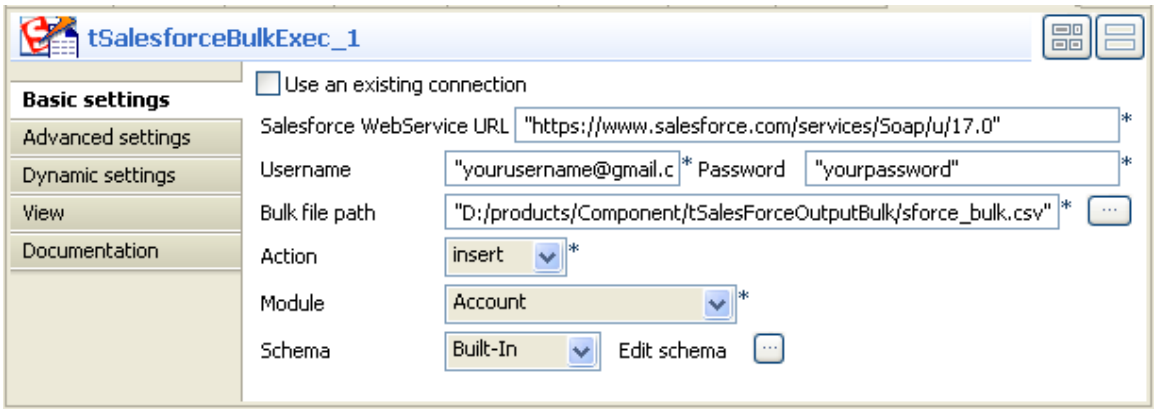


2. In the **File Name** field, type in or browse to the directory where you want to store the generated *.csv* data for bulk processing.

3. Click **Sync columns** to import the schema from its preceding component.

Setting up the connection to the Salesforce server

1. Double-click **tSalesforceBulkExect** to display its **Basic settings** view and define the component properties.



2. Use the by-default URL of the Salesforce Web service or enter the URL you want to access.

3. In the **Username** and **Password** fields, enter your username and password for the Web service.

4. In the **Bulk file path** field, browse to the directory where is stored the generated *.csv* file by **tSalesforceOutputBulk**.

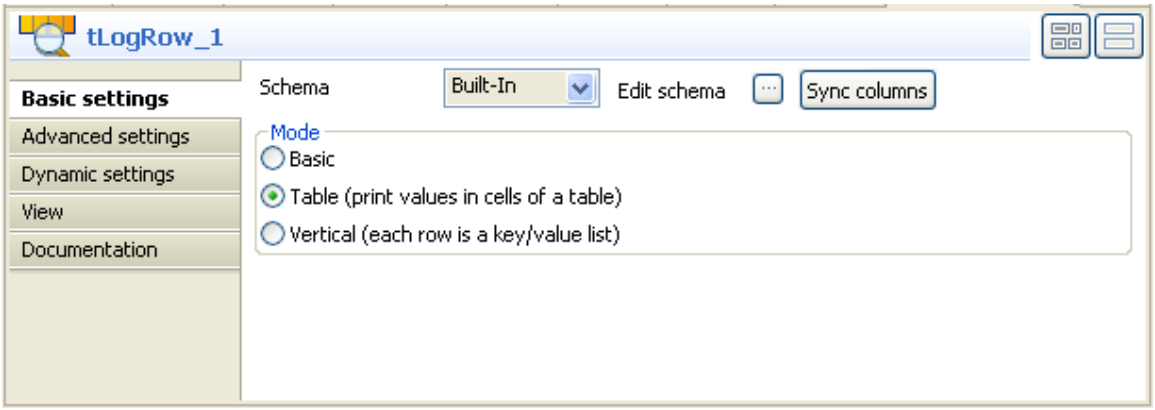
5. From the **Action** list, select the action you want to carry out on the prepared bulk data. In this use case, **insert**.

6. From the **Module** list, select the object you want to access, **Account** in this example.

7. Click the three-dot button next to the **Edit schema** field to open the dialog box to set the schema. In this example, edit it conforming to the schema defined previously.

Configuring the output component

1. Double-click **tLogRow_1** to display its **Basic settings** view and define the component properties.



2. Click **Sync columns** to retrieve the schema from the preceding component.
3. Select **Table** mode to display the execution result.
4. Do the same with **tLogRow_2**.

Job execution

1. Press **CTRL+S** to save your Job.
2. Press **F6** to execute it.

   You can check the execution result on the **Run** console.

```
[statistics] connected
+-----------------------------------+--------+--------------+--------------+------------------+-----
|                                                tLogRow_1
|=----------------------------------+--------+--------------+--------------+------------------+-----
|Name                               |ParentId|Phone         |Fax           |salesforce_id     |sales
|=----------------------------------+--------+--------------+--------------+------------------+-----
|BURLINGTON TEXTILES CORP OF AMERICA|null    |(336) 222-7000|(336) 222-8000|0018000000gfDyDAAU|true
|DICKENSON PLC                      |null    |(785) 241-6200|(785) 241-6201|0018000000gfDyEAAU|true
|GENEPOINT                          |null    |(650) 867-3450|(650) 867-9895|0018000000gfDyFAAU|true
+-----------------------------------+--------+--------------+--------------+------------------+-----

+-------------------------+--------+--------------+--------------+-------------------------------
|                                                tLogRow_2
|=------------------------+--------+--------------+--------------+-------------------------------
|Name                     |ParentId|Phone         |Fax           |error
|=------------------------+--------+--------------+--------------+-------------------------------
|EDGE COMMUNICATIONS      |wrong id|(512) 757-6000|(512) 757-9000|MALFORMED_ID:Parent Account ID:
|GRAND HOTELS & RESORTS LTD|wrong id|(312) 596-1000|(312) 596-1500|MALFORMED_ID:Parent Account ID:
+-------------------------+--------+--------------+--------------+-------------------------------

[statistics] disconnected
Job tSalesforcebulk ended at 18:17 13/04/2010. [exit code=0]
```

   In the **tLogRow_1** table, you can read the data inserted into your Salesforce.com.

   In the **tLogRow_2** table, you can read the rejected data due to the incompatibility with the **Account** objects you have accessed.

   All the customer names are written in upper case.

tSalesforceOutputBulkExec

tSalesforceOutputBulkExec Properties

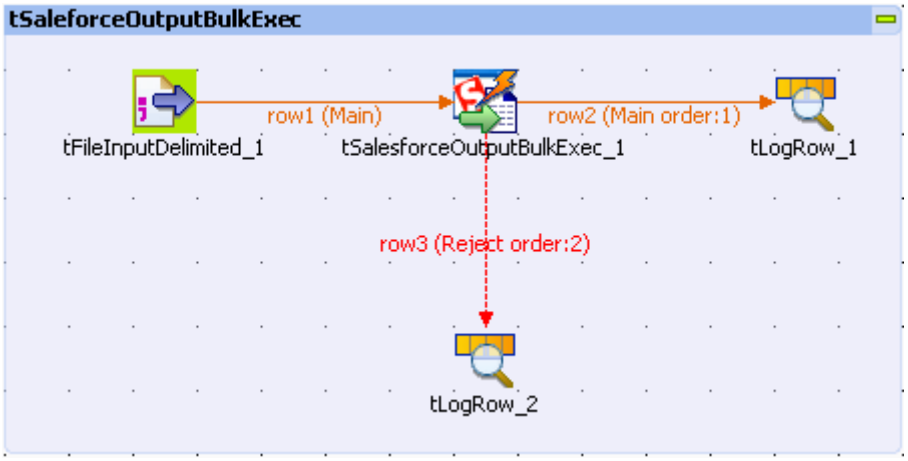
tSalesforceOutputBulk and **tSalesforceBulkExec** components are used together to output the needed file and then execute intended actions on the file for your Salesforce.com. These two steps compose the **tSalesforceOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Business/Cloud	
Function	**tSalesforceOutputBulkExec** executes the intended actions on the *.csv* bulk data for Salesforce.com.	
Purpose	As a dedicated component, **tSalesforceOutputBulkExec** gains performance while carrying out the intended data operations into your Salesforce.com.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tSalesforceConnection**. Once you select it, the **Component list** field appear allowing you to choose the **tSalesforceConnection** component to be used. For more information on **tSalesforceConnection**, see section *tSalesforceConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Salesforce Webservice URL	Type in the Web service URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Web service user authentication data.
	Salesforce Version	Type in the version of the Salesforce you are using.
	Bulk file path	Directory where are stored the bulk data you need to process.
	Action	You can do any of the following operations on the data of the Salesforce object: **Insert**: insert data. **Update**: update data. **Upsert**: update and insert data. **Delete**: delete data.
	Upsert Key Column	Specify the key column for the upsert operation. Available when **Upsert** is selected from the **Action** list.
	Module	Select the relevant module in the list. If you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.

		Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Rows to commit*	Specify the number of lines per data batch to be processed.
	Bytes to commit	Specify the number of bytes per data batch to be processed.
	Concurrency mode	The concurrency mode for the job. **Parallel**: process batches in parallel mode. **Serial**: process batches in serial mode.
	Use Socks Proxy	Select this check box if you want to use a proxy server. In this case, you should fill in the proxy parameters in the **Proxy host**, **Proxy port**, **Proxy username** and **Proxy password** fields which appear beneath.
	Ignore NULL fields values	Select this check box to ignore NULL values in **Update** or **Upsert** mode.
	Relationship mapping for upsert	Click the **[+]** button to add lines as needed and specify the external ID fields in the input flow, the upsert module and the lookup module. **Column name of Talend schema**: external ID field in the input flow. **Lookup field name**: external ID field in the upsert module. **External id name**: external ID field in the lookup module.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded into Salesforce.com.	
Limitation	The bulk data to be processed in Salesforce.com should be *.csv* format.	

Scenario: Inserting bulk data into your Salesforce.com

This scenario describes a four-component Job that submits bulk data into Salesforce.com, executes your intended actions on the data, and ends up with displaying the Job execution results for your reference.



Before starting this scenario, you need to prepare the input file which offers the data to be processed by the Job. In this use case, this file is *sforcebulk.txt*, containing some customer information.

Then to create and execute this Job, operate as follows:

Setting up the Job

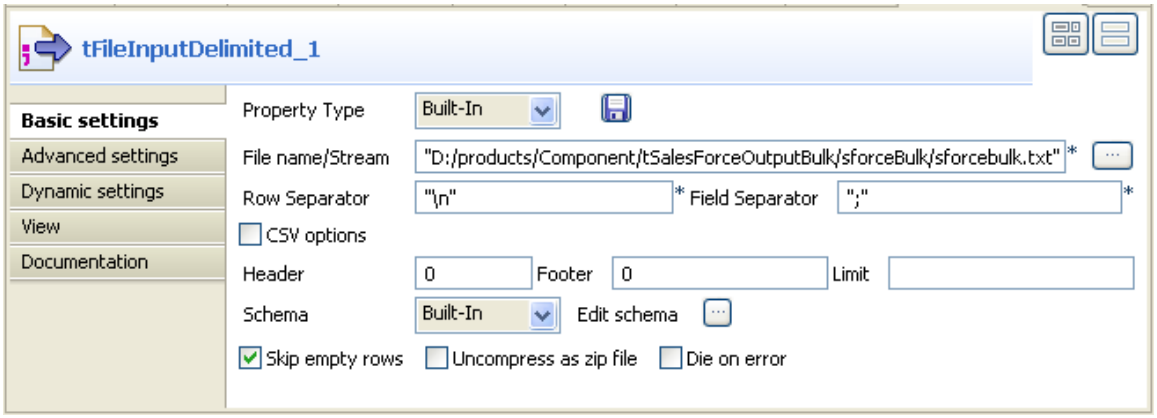
1. Drop **tFileInputDelimited**, **tSalesforceOutputBulkExec**, and **tLogRow** from the **Palette** onto the workspace of your studio.

2. Use **Row** > **Main** connection to connect **tFileInputDelimited** to **tSalesforceOutputBulkExec**.

3. Use **Row** > **Main** and **Row** > **Reject** to connect **tSalesforceOutputBulkExec** respectively to the two **tLogRow** components.

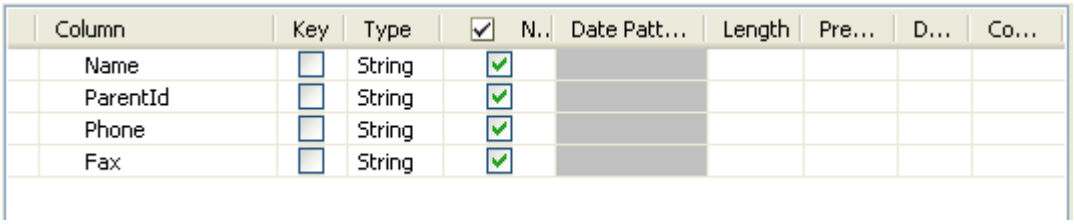
Setting the input data

1. Double-click **tFileInputDelimited** to display its **Basic settings** view and define the component properties.



2. Next to the **File name/Stream** field, click the **[...]** button to browse to the input file you prepared for the scenario, for example, *sforcebulk.txt*.

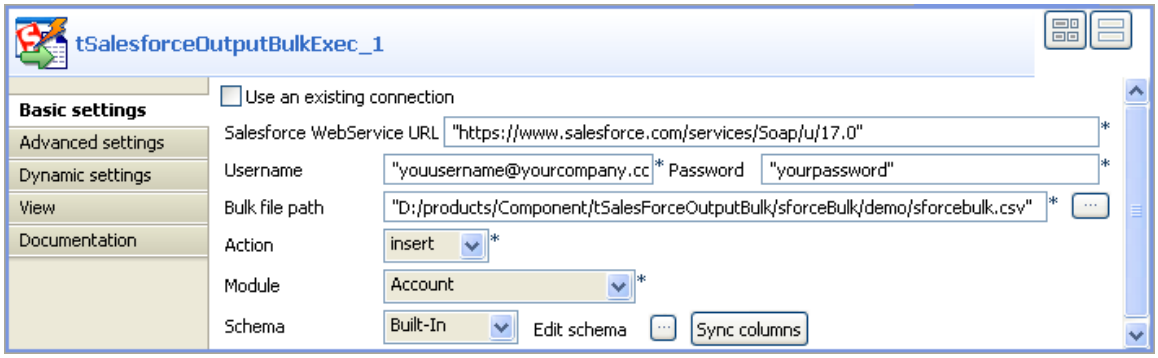
3. Click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually. In this scenario, the schema is made of four columns: *Name*, *ParentId*, *Phone* and *Fax*.



4. According to your input file to be used by the Job, set the other fields like **Row Separator**, **Field Separator**...

Setting up the connection to the Salesforce server

1. Double-click **tSalesforceOutputBulkExec** to display its **Basic settings** view and define the component properties.



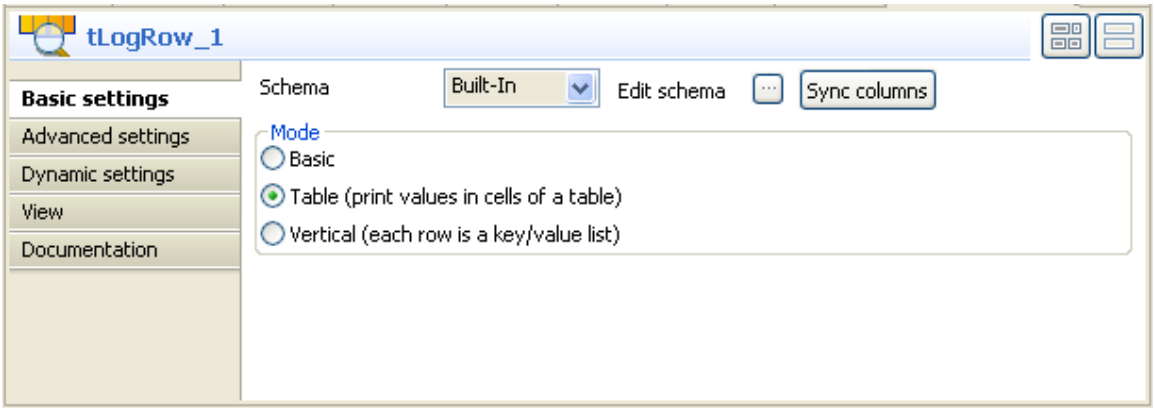
2. In **Salesforce WebService URL** field, use the by-default URL of the Salesforce Web service or enter the URL you want to access.
3. In the **Username** and **Password** fields, enter your username and password for the Web service.
4. In the **Bulk file path** field, browse to the directory where you store the bulk *.csv* data to be processed.

   The bulk file here to be processed must be in *.csv* format.

5. From the **Action** list, select the action you want to carry out on the prepared bulk data. In this use case, **insert**.
6. From the **Module** list, select the object you want to access, **Account** in this example.
7. Click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually. In this example, edit it conforming to the schema defined previously.

Job execution

1. Double-click **tLogRow_1** to display its **Basic settings** view and define the component properties.



2. Click **Sync columns** to retrieve the schema from the preceding component.
3. Select **Table** mode to display the execution result.
4. Do the same with **tLogRow_2**.
5. Press **CTRL+S** to save your Job and press **F6** to execute it.

   On the console of the **Run** view, you can check the execution result.

```
[statistics] connected
+-----------------------------------+--------+--------------+--------------+------------------+-----
|                                                tLogRow_1                                            |
|=----------------------------------+--------+--------------+--------------+------------------+-----
|Name                               |ParentId|Phone         |Fax           |salesforce_id     |sales
|=----------------------------------+--------+--------------+--------------+------------------+-----
|Burlington Textiles Corp of America|null    |(336) 222-7000|(336) 222-8000|0018000000gfDyDAAU|true
|Dickenson plc                      |null    |(785) 241-6200|(785) 241-6201|0018000000gfDyEAAU|true
|GenePoint                          |null    |(650) 867-3450|(650) 867-9895|0018000000gfDyFAAU|true
+-----------------------------------+--------+--------------+--------------+------------------+-----

+-------------------------+--------+--------------+--------------+-------------------------------
|                                                tLogRow_2                                          |
|=------------------------+--------+--------------+--------------+-------------------------------
|Name                     |ParentId|Phone         |Fax           |error
|=------------------------+--------+--------------+--------------+-------------------------------
|Edge Communications      |wrong id|(512) 757-6000|(512) 757-9000|MALFORMED_ID:Parent Account ID: :
|Grand Hotels & Resorts Ltd|wrong id|(312) 596-1000|(312) 596-1500|MALFORMED_ID:Parent Account ID: :
+-------------------------+--------+--------------+--------------+-------------------------------
```

In the **tLogRow_1** table, you can read the data inserted into your Salesforce.com.

In the **tLogRow_2** table, you can read the rejected data due to the incompatibility with the **Account** objects you have accessed.

If you want to transform the input data before submitting them, you need to use **tSalesforceOutputBulk** and **tSalesforceBulkExec** in cooperation to achieve this purpose. For further information on the use of the two components, see section *Scenario: Inserting transformed bulk data into your Salesforce.com*.

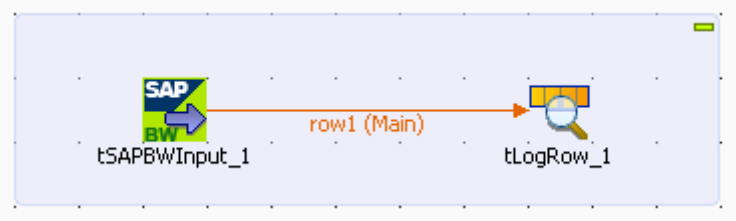
tSAPBWInput

tSAPBWInput Properties

Component family	Business	
Function	**tSAPBWInput** reads data from an SAP BW database using a JDBC API connection and extracts fields based on an SQL query.	
Purpose	This component executes an SQL query with a strictly defined order which must correspond to your schema definition. Then it passes on the field list to the next component via a **Row** > **Main** connection.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
	JDBC URL	Enter the JDBC URL of the database you want to connect to. For example, enter: *jdbc:jdbc4olap://server_address/database_name* to connect to an SAP BW database.
	Username	Enter the username for DB access authentication.
	Password	Enter the password for DB access authentication.
	Table Name	Type in the name of the DB table.
	Query Type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**:
		Built-in: No property data stored centrally.
	Guess Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns. Clear **Trim all the String/Char columns** to enable Trim columns in this field.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component supports SQL queries for SAP BW database using a JDBC connection.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Reading data from SAP BW database

This scenario describes a two-component Job that reads data from an SAP BW database. The data is fetched and displayed on the console.



Prior to setting up the Job, make sure the following prerequisites are met:

1. Copy the following .jar files which compose the jdbc4olap driver to your class path:

   -activation.jar

   -commons-codec.jar

   -jdbc4olap.jar

   -saaj-api.jar

   -saaj-impl.jar

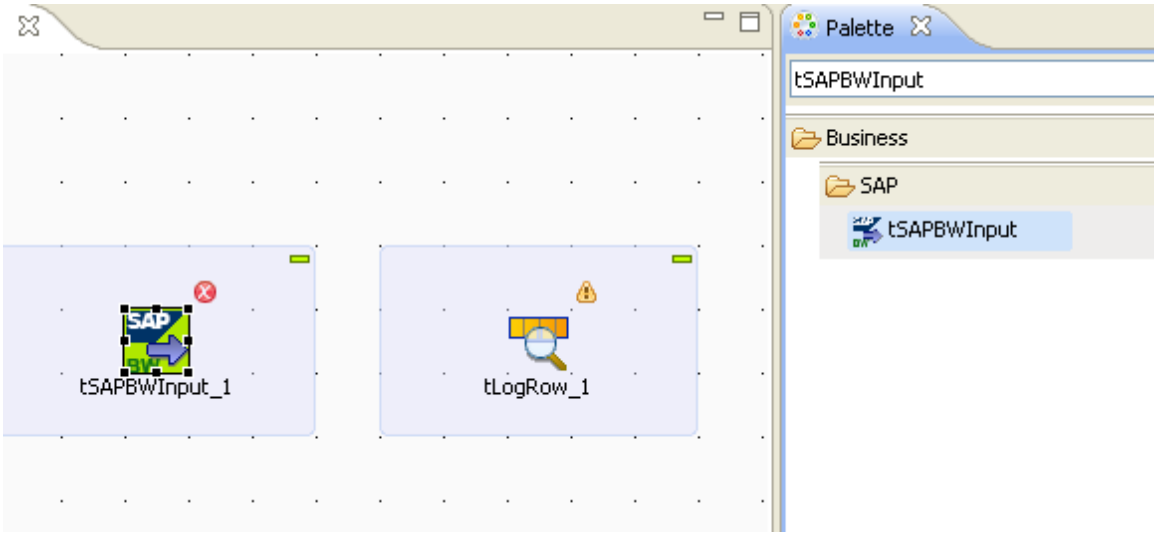
2. Make sure that you have the latest version of jdbc4olap driver. You can download the latest version of jdbc4olap driver from jdbc4olap download section. For further information about the usage of jdbc4olap driver, see jdbc4olap User Guide.

The procedure of this scenario requires 4 main steps detailed hereafter:

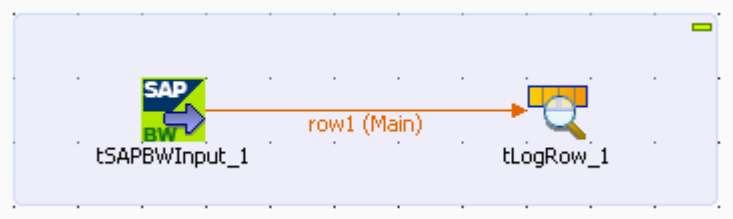
1. Set up the Job.
2. Set up the jdbc connection to the SAP BW server.
3. Set up a query.
4. Display the fetched data on the console.

Set up the Job

1. Drop a **tSAPBWInput** component and a **tLogRow** component from the **Palette** onto the workspace.

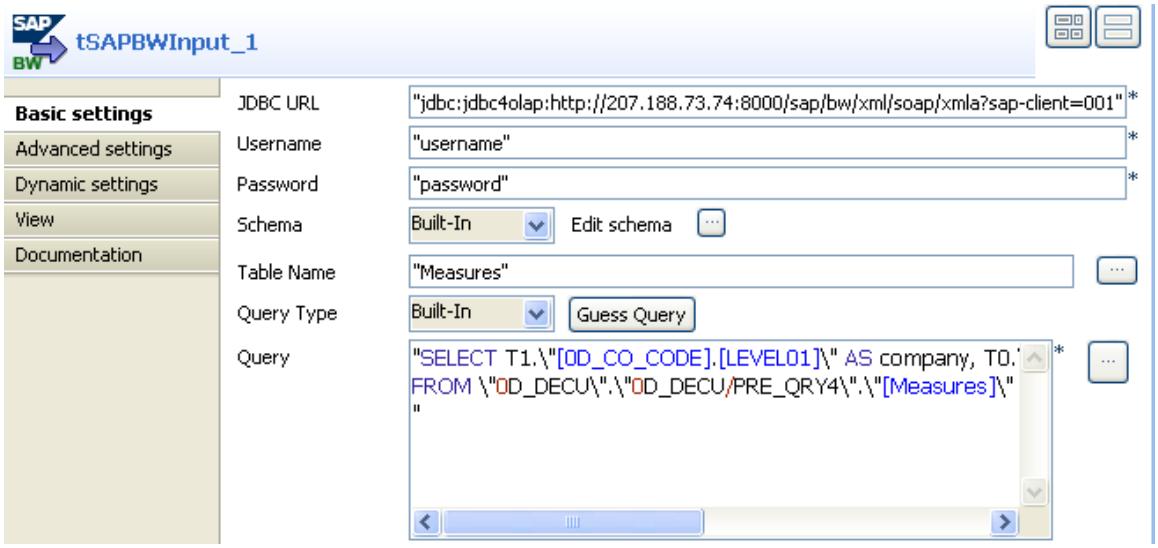


2. Connect the **tSAPBWInput** component and the **tLogRow** component using a **Row** > **Main** connection.



Set up the jdbc connection to the SAP BW server

1. Double-click the **tSAPBWInput** component to open its **Basic settings** view and define the component properties.

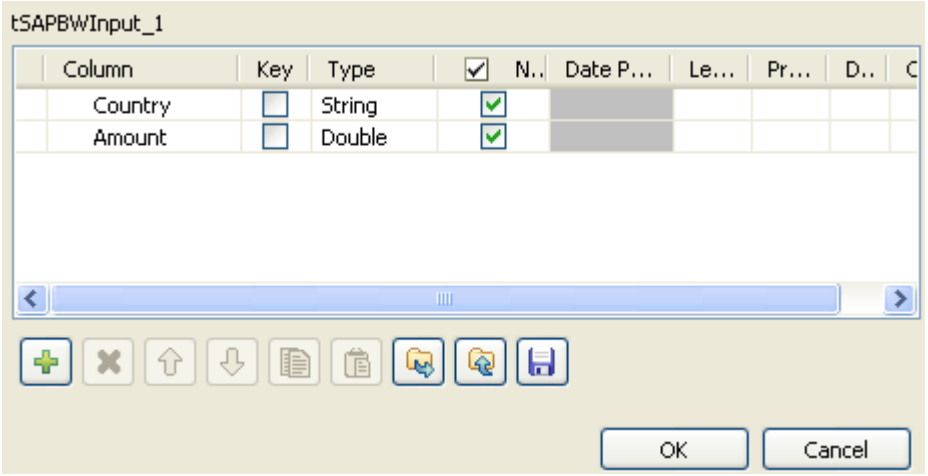


2. Fill the **JDBC URL** field with the URL of your jdbc4olap server.

   Note that the URL displayed above is for demonstration only.

3. Fill the **Username** and **Password** fields with your username and password for the DB access authentication.

4. Click the three-dot button next to **Edit schema** to define the schema to be used.



5. Click the plus button to add new columns to the schema and set the data type for each column and click **OK** to save the schema settings.

Set up a query

1. From the **Basic settings** view of **tSAPBWInput**, fill the **Table Name** field with the table name. In this scenario, table name *"Measures"* is for demonstration only.

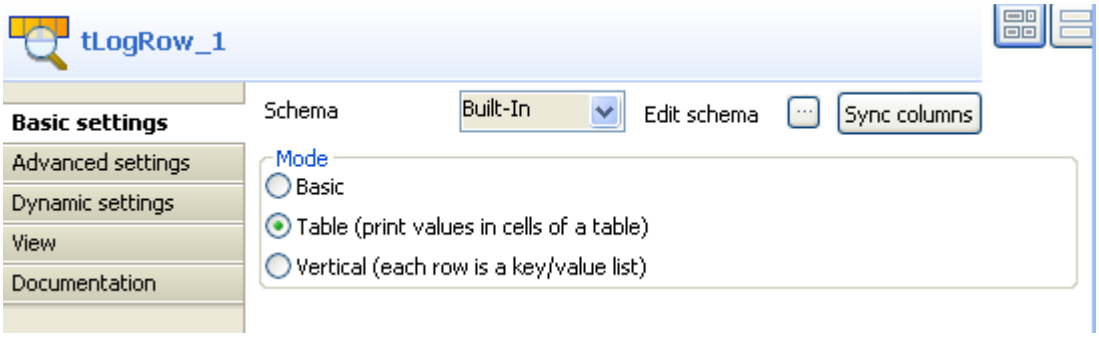
2. Fill the **Query** area with the query script. In this example, we use:

```
"SELECT
T1.\"[0D_CO_CODE].[LEVEL01]\" AS company,
T0.\"[Measures].[D68EEPGGHUMSZ92PIJARDZ0KA]\" AS amount
FROM
\"0D_DECU\".\"0D_DECU/PRE_QRY4\".\"[Measures]\" T0,
\"0D_DECU\".\"0D_DECU/PRE_QRY4\".\"[0D_CO_CODE]\" T1 "
```

Due to the limitations of the supported SQL queries, the query scripts you use must be based on the grammar defined in the jdbc4olap driver. For further information about this grammar, see jdbc4olap User Guide.

Display the fetched data on the console

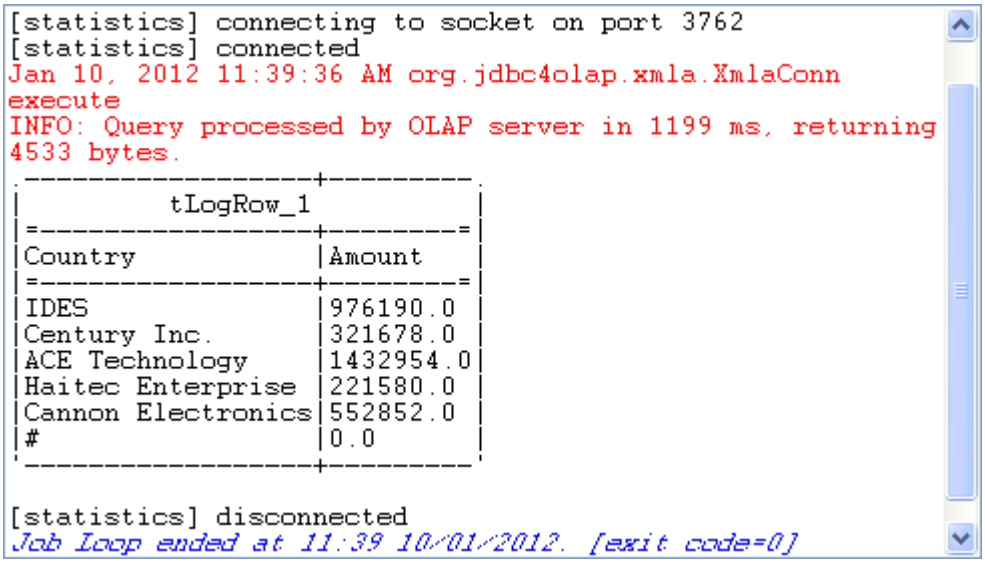
1. Double-click the **tLogRow** component to open its **Basic settings** view and define the component properties.



2. Click **Sync columns** to retrieve the schema defined in the preceding component.

3. Select **Table** in the **Mode** area.

4. Press **Ctrl**+**S** to save your Job and press **F6** to execute it.



The data in the table *"Measure"* is fetched and displayed on the console.

tSAPCommit

tSAPCommit Properties

This component is closely related to **tSAPConnection** and **tSAPRollback**. It usually does not make much sense to use these components separately in a transaction.

Component family	Business/SAP	
Function	Validates the data processed through the Job into the connected server.	
Purpose	Using a unique connection, this component commits a global transaction in one go instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*SAPConnection Component list*	Select the **tSAPConnection** component in the list if more than one connection are planned for the current Job.
	Release Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row >Main** connection to link **tSAPCommit** to your Job, your data will be commited row by row. In this case, do not select the **Release connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with SAP components, especially with **tSAPConnection** and **tSAPRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tSAPConnection** and **tSAPRollback**. It usually does not make much sense to use one of these without using a **tSAPConnection** component to open a connection for the current transaction.

For **tSAPCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tSAPConnection

tSAPConnection properties

Component family	Business	
Function	**tSAPConnection** opens a connection to the SAP system for the current transaction.	
Purpose	**tSAPConnection** allows to commit a whole Job data in one go to the SAP system as one transaction.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data is stored centrally.
	Connection configuration	**Client type**: enter your usual SAP connection. **Userid :** enter user login. **Password**: enter password. **Language**: specify the language. **Host name**: enter the IP address of the SAP system. **System number**: enter the system number.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with other SAP components.	
Limitation	n/a	

Related scenarios

For a related scenarios, see section *Scenario 1: Retrieving metadata from the SAP system* and section *Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function*.

tSAPInput

tSAPInput Properties

Component family	Business	
Function	**tSAPInput** connects to the SAP system using the system IP address.	
Purpose	**tSAPInput** allows to extract data from an SAP system at any level through calling RFC or BAPI functions.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**:
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant connection component on the **Component** list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Connection configuration	**Client type**: Enter your SAP usual connection code **Userid:** Enter the user connection Id. **Password**: Enter the password. **Language**: Specify a language. **Host name** Enter the SAP system IP address. **System number** Enter the system number.
	FunName	Enter the name of the function you want to use to retrieve data.
	Initialize input	Set input parameters. **Parameter Value**: Enter between inverted commas the value that corresponds to the parameter you set in the **Parameter Name** column. **Type**: Select the type of the input entity to retrieve. **Table Name (Structure Name)**: Enter between inverted commas the table name. **Parameter Name**: Enter between in,verted commas the name of the field that corresponds to the table set in the **Table Name** column. When you need different parameter values using the same parameter name, you should enter these values in one row and delimit them with comma.
	Outputs	Configure the parameters of the output schema to select the data to be extracted:

		Schema: Enter the output schema name. **Type (for iterate)**: Select the type of the output entity you want to have. **Table Name (Structure Name)**: Enter between inverted commas the table name. **Mapping**: Enter between inverted commas the name of the field you want to retrieve data from. You can set as many outgoing **Main** links used to output data as schemas you added to this **Outputs** table. This way, data can be grouped into different files.
Connections		Outgoing links (from one component to another): **Row**: Main, Iterate. **Trigger**: Run if; On Component Ok; On Component Error, On Subjob Ok, On Subjob Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if, On Component Ok, On Component Error, On Subjob Ok, On Subjob Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Release Connection*	Clear this check box to continue to use the selected connection once the component has performed its task.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario 1: Retrieving metadata from the SAP system

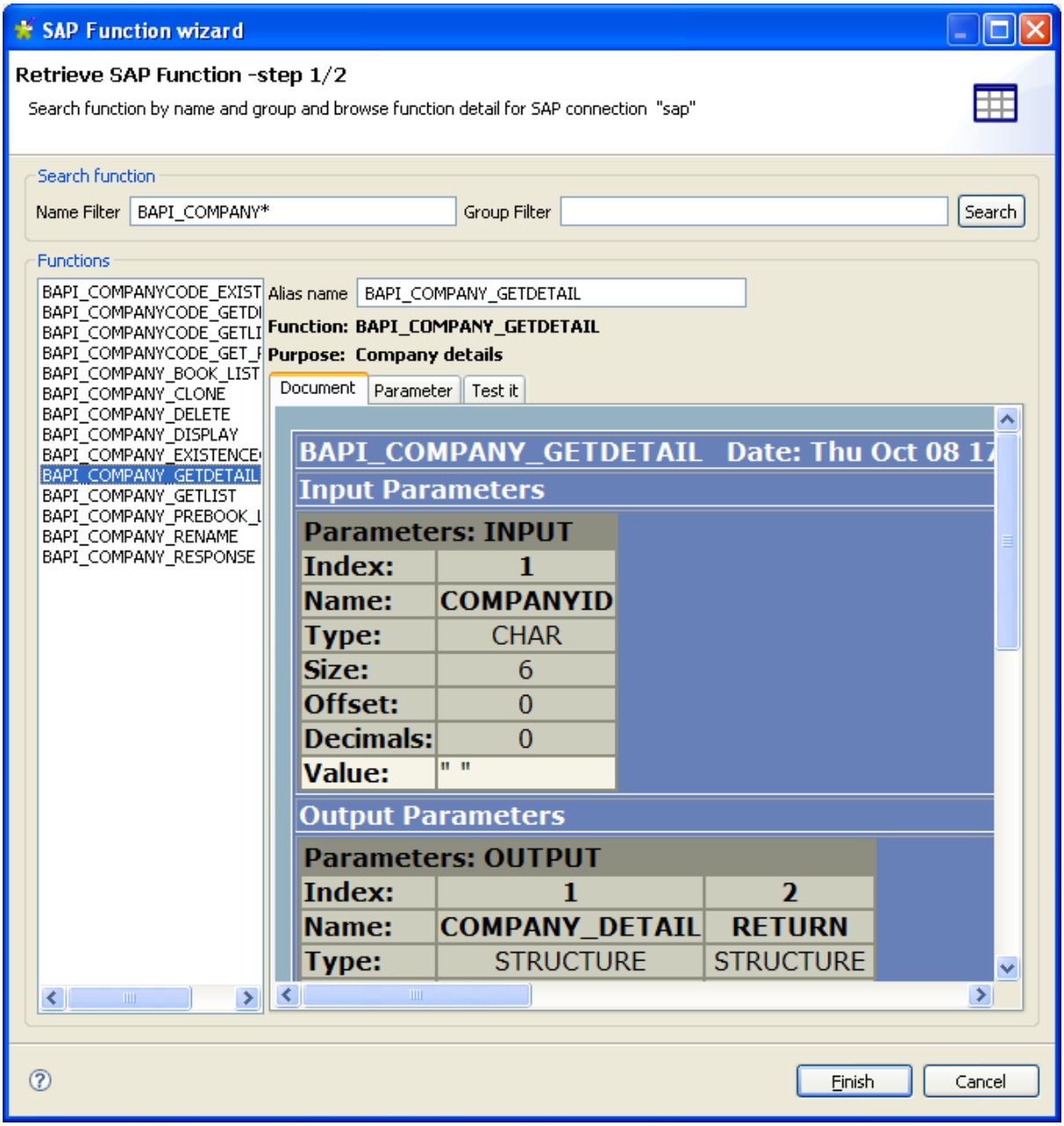
This scenario is for reference only. If you are using Talend Open Studio for Big Data, the property type and schema of a component are always built-in, which means you have to configure the connection details and schemas manually.

Talend SAP components (**tSAPInput** and **tSAPOutput**) as well as the SAP wizard are based on a library validated and provided by SAP (JCO) that allows the user to call functions and retrieve data from the SAP system at Table, RFC or BAPI, levels.

This scenario uses the SAP wizard that leads a user through dialog steps to create SAP connection and call RFC and BAPI functions. This SAP wizard is available only for users who have subscribed to one of the *Talend* solutions. Otherwise, you need to drop the **tSAPInput** component from the **Palette** and set its basic settings manually.

This scenario uses the SAP wizard to first create a connection to the SAP system, and then call a BAPI function to retrieve the details of a company from the SAP system. It finally displays in *Talend Open Studio for Big Data* the company details stored in the SAP system.

The following figure shows the company detail parameters stored in the SAP system and that we want to read in *Talend Open Studio for Big Data* using the **tSAPInput** component.



Setting and configuring the SAP connection using wizard

Setting up the connection to the SAP system

1. Create a connection to the SAP system using the SAP connection wizard, in this scenario the SAP connection is called *sap* and is saved in the **Metadata** node.
2. Call the BAPI function *BAPI_COMPANY_GETDETAIL* using the SAP wizard to access the BAPI HTML document stored in the SAP system and see the company details.
3. In the **Name filter** field, type in *BAPI** and click the **Search** button to display all available BAPI functions.
4. Select *BAPI_COMPANY_GETDETAIL* to display the schema that describes the company details.

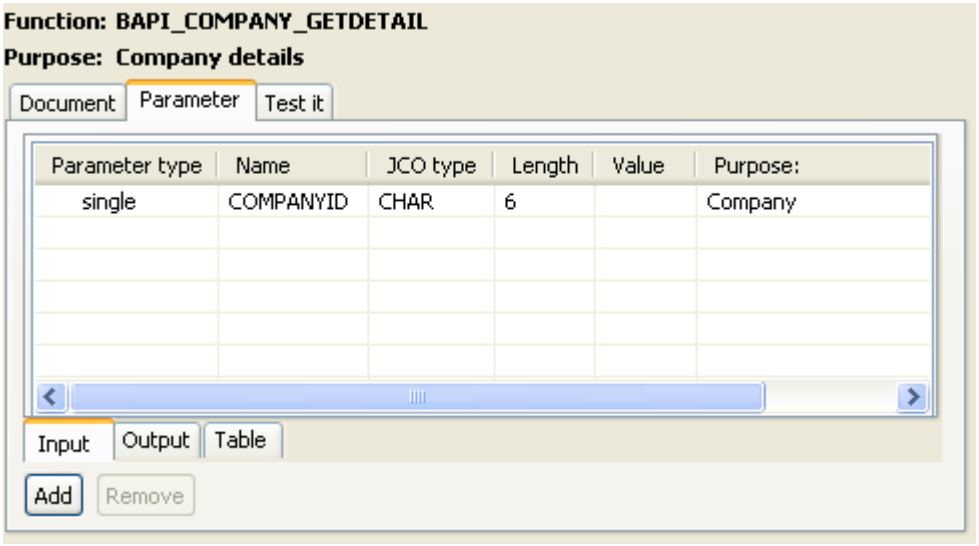
The three-tab view to the right of the wizard displays the metadata of the *BAPI_COMPANY_GETDETAIL* function and allows you to set the necessary parameters.

The **Document** view displays the SAP html document about the *BAPI_COMPANY_GETDETAIL* function.

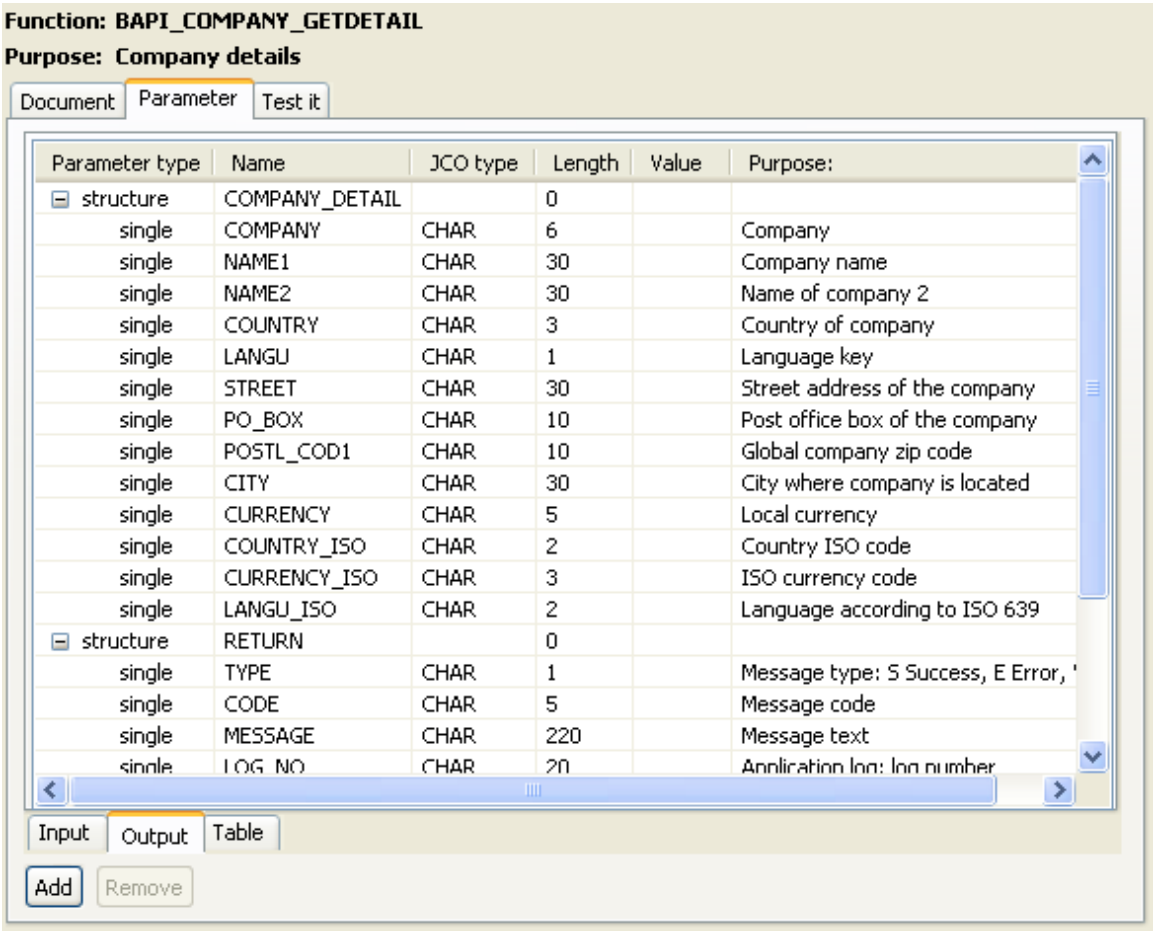
The **Parameter** view provides information about the input and output parameters required by the *BAPI_COMPANY_GETDETAIL* function to return values.

Setting the input and output parameters using the wizard

1. In the **Parameter** view, click the **Input** tab to list the input parameter(s). In this scenario, there is only one input parameter required by *BAPI_COMPANY_GETDETAIL* and it is called *COMPANYID*.

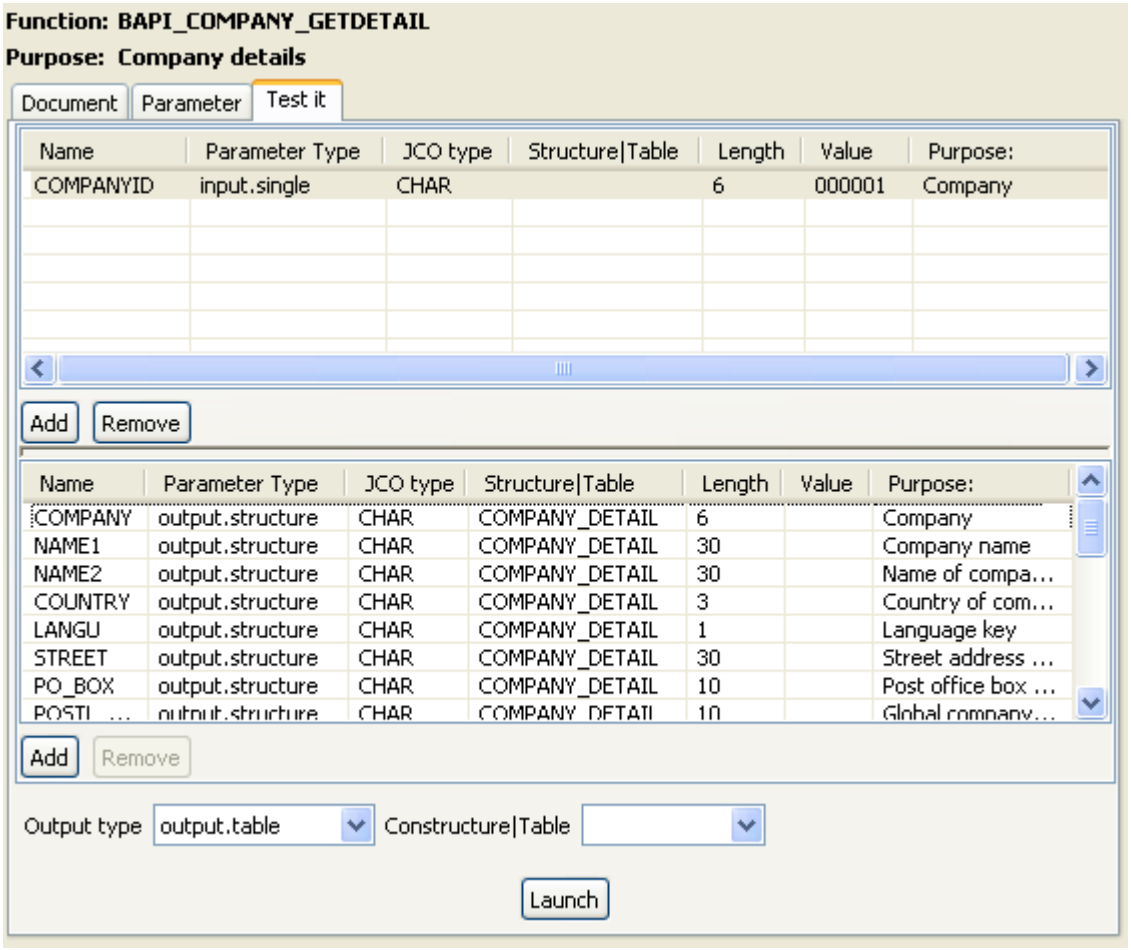


2. In the **Parameter** view, click the **Output** tab to list the output parameters returned by *BAPI_COMPANY_GETDETAIL*. In this scenario, there are two output parameters: *COMPANY_DETAIL* and *RETURN*.



Each of these two “structure” parameters consists of numerous “single” parameters.

The **Test it** view allows you to add or delete input parameters according to the called function. In this scenario, we want to retrieve the metadata of the *COMPANY_DETAIL* “structure” parameter that consists of 14 “single” parameters.



3. In the **Value** column of the **COMPANYID** line in the first table, enter *"000001"* to send back company data corresponding to the value 000001.

4. In the **Output type** list at the bottom of the wizard, select **output.table**.

5. Click **Launch** at the bottom of the view to display the value of each "single" parameter returned by the *BAPI_COMPANY_GETDETAIL* function.

6. Click **Finish** to close the wizard and create the connection.

The *sap* connection and the new schema **BAI_COMPANY_GETDETAIL** display under the **SAP Connections** node in the **Repository** tree view.

Retrieving different schemas of the SAP functions

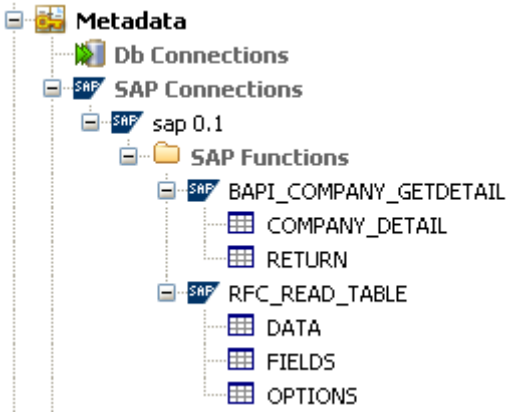
To retrieve the different schemas of the **BAPI_COMPANY_GETDETAIL** function, do the following:

1. Right-click **BAPI_COMPANY_GETDETAIL** in the **Repository** tree view and select **Retrieve schema** in the contextual menu.

2. In the open dialog box, select the schemas you want to retrieve, **COMPANY_DETAIL** and **RETURN** in this scenario.

3. Click **Next** to display the two selected schemas and then **Finish** to close the dialog box.

The two schemas display under the **BAPI_COMPANY_GETDETAIL** function in the **Repository** tree view.



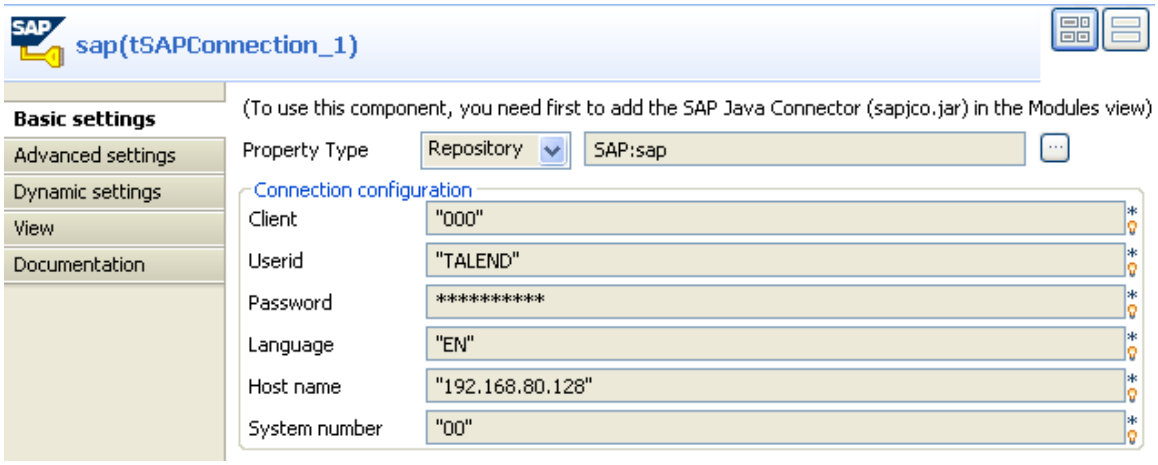
Retrieving the company metadata

To retrieve the company metadata that corresponds to the 000001 value and display it in *Talend Open Studio for Big Data*, do the following:

Setting up the Job

1. In the **Repository** tree view, drop the SAP connection you already created to the design workspace to open a dialog box where you can select **tSAPConnection** from the component list and finally click **OK** to close the dialog box. The **tSAPConnection** component holding the SAP connection, *sap* in this example, displays on the design workspace.

2. Double-click **tSAPConnection** to display the **Basic settings** view and define the component properties.



If you store connection details in the **Metadata** node in the **Repository** tree view, the **Repository** mode is selected in the **Property Type** list and the fields that follow are pre-filled. If not, you need to select **Built-in** as “property type” and fill in the connection details manually.

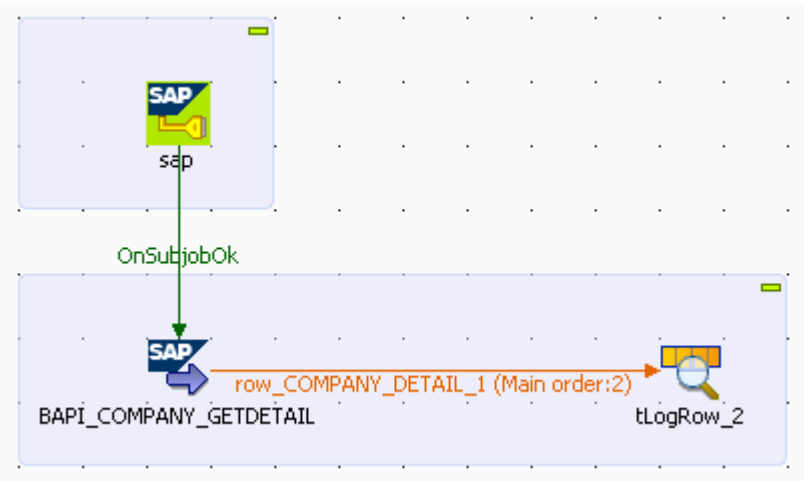
3. In the **Repository** tree-view, expand **Metadata** and **sap** in succession and drop **RFC_READ_TABLE** to the design workspace to open a component list.

4. Select **tSAPInput** from the component list and click **OK**.

5. Drop **tFilterColumns** and **tLogRow** from the **Palette** to the design workspace.

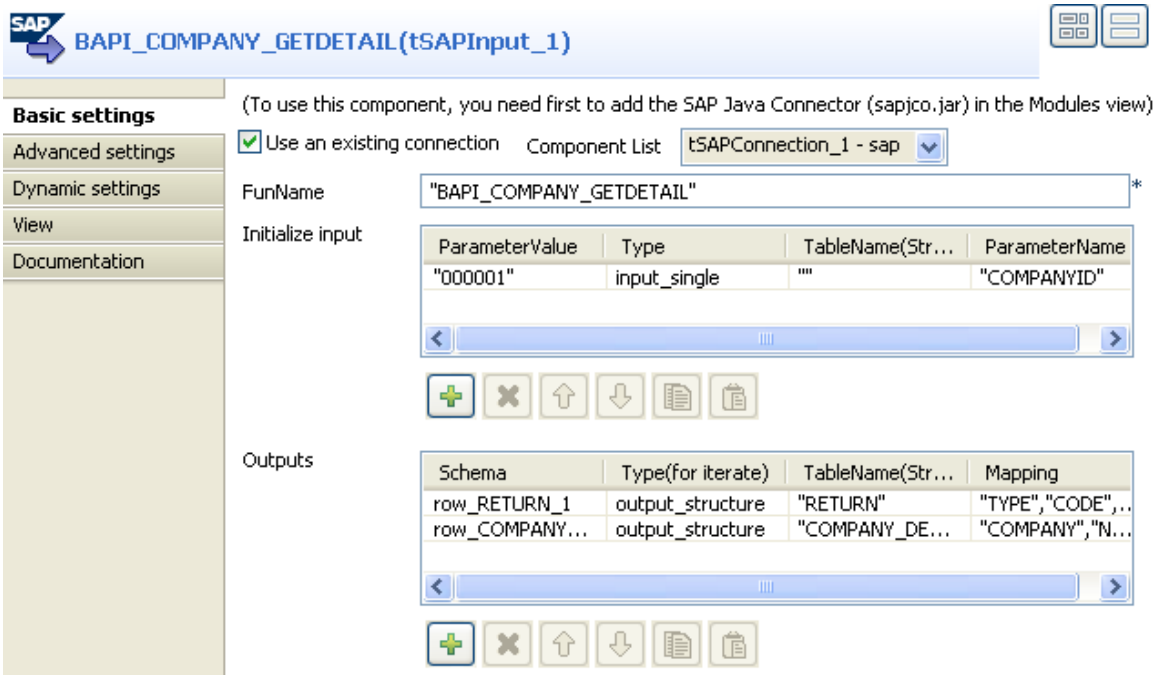
6. Connect **tSAPConnection** and **tSAPInput** using a **Trigger** > **OnSubJobOk** link

7. To connect **tSAPInput** and **tLogRow**, right-click **tSAPInput** and select **Row** > **row_COMPANY_DETAIL_1** and then click **tLogRow**.



8. In the design workspace, double-click **tSAPInput** to display its **Basic settings** view and define the component properties**.**

   The basic setting parameters for the **tSAPInput** component display automatically since the schema is stored in the **Metadata** node and the component is initialized by the SAP wizard.



9. Select the **Use an existing connection** check box and then in the **Component List**, select the relevant **tSAPConnection** component, **sap** in this scenario.

In the **Initialize input** area, we can see the input parameter needed by the *BAPI_COMPANY_GETDETAIL* function.

In the **Outputs** area, we can see all different schemas of the **BAPI_COMPANY_GETDETAIL** function, in particular, **COMPANY_DETAIL** that we want to output.

Job execution

1. In the design workspace, double-click **tLogRow** to display the **Basic settings** view and define the component properties. For more information about this component, see section *tLogRow*.

2. Press **CTRL+S** to save your Job and press **F6** to execute it.

```
Starting job BAPI_SAP at 18:32 07/10/2009.

[statistics] connecting to socket on port 4100
[statistics] connected
.-----------------------------------.
|           #1. tLogRow_2           |
+--------------+--------------------+
| key          | value              |
+--------------+--------------------+
| COMPANY      | 000001             |
| NAME1        | Gesellschaft G00000|
| NAME2        |                    |
| COUNTRY      | DE                 |
| LANGU        | D                  |
| STREET       | Neurottstrasse 16  |
| PO_BOX       |                    |
| POSTL_COD1   | 69190              |
| CITY         | Walldorf           |
| CURRENCY     | EUR                |
| COUNTRY_ISO  | DE                 |
| CURRENCY_ISO | EUR                |
| LANGU_ISO    | DE                 |
+--------------+--------------------+
[statistics] disconnected
Job BAPI_SAP ended at 18:32 07/10/2009. [exit code=0]
```

The **tSAPInput** component retrieved from the SAP system the metadata of the *COMPANY_DETAIL* “structure” parameter and **tLogRow** displayed the information on the console.

Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function

This scenario is for reference only. If you are using Talend Open Studio for Big Data, the property type and schema of a component are always built-in, which means you have to configure the connection details and schemas manually.

Talend SAP components (**tSAPInput** and **tSAPOutput**) as well as the SAP wizard are based on a library validated and provided by SAP (JCO) that allows the user to call functions and retrieve data from the SAP system at Table, RFC or BAPI, levels.

This scenario uses the SAP wizard that leads a user through dialog steps to create a SAP connection and call RFC and BAPI functions. This SAP wizard is available only for users who have subscribed to one of the *Talend* solutions. Otherwise, you need to drop the **tSAPInput** component from the **Palette** and set its basic settings manually.

This scenario uses the SAP wizard to first create a connection to the SAP system, and then call an RFC function to directly read from the SAP system a table called *SFLIGHT*. It finally displays in *Talend Open Studio for Big Data* the structure of the *SFLIGHT* table stored in the SAP system.

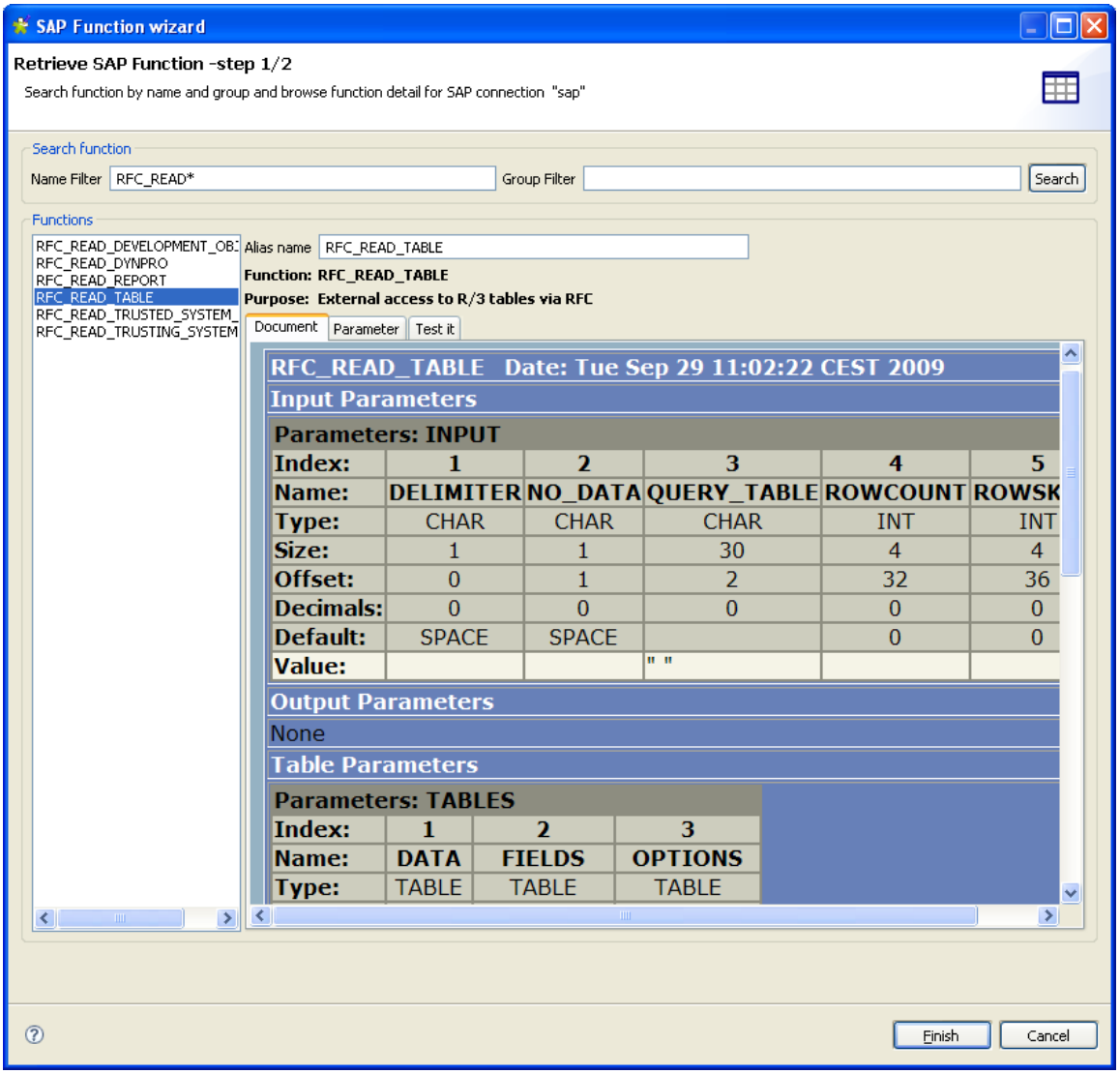
Setting and configuring the SAP connection using wizard

Setting up the connection to the SAP system

1. Create a connection to the SAP system using the SAP connection wizard, in this scenario the SAP connection is called *sap*.

2. Call the *RFC_READ_TABLE* RFC function using the SAP wizard to access the table in the SAP system and see its structure.

3. In the **Name filter** field, type in *RFC** and click the **Search** button to display all available RFC functions.



4. Select *RFC_READ_TABLE* to display the schema that describe the table structure.

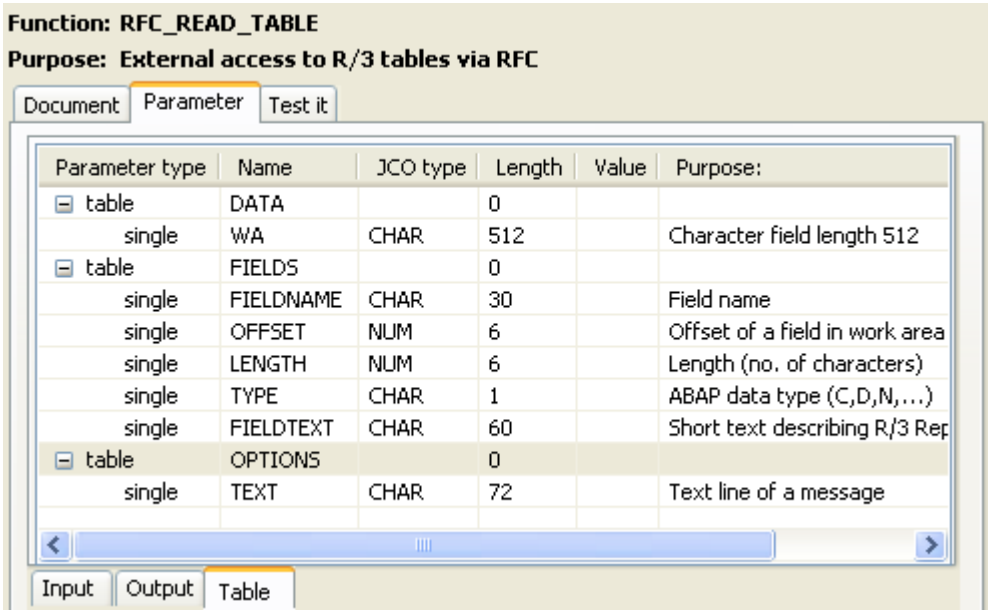
The three-tab view to the right of the wizard displays the metadata of the *RFC_READ_TABLE* function and allows you to set the necessary parameters.

The **Document** view displays the SAP html document about the *RFC_READ_TABLE* function.

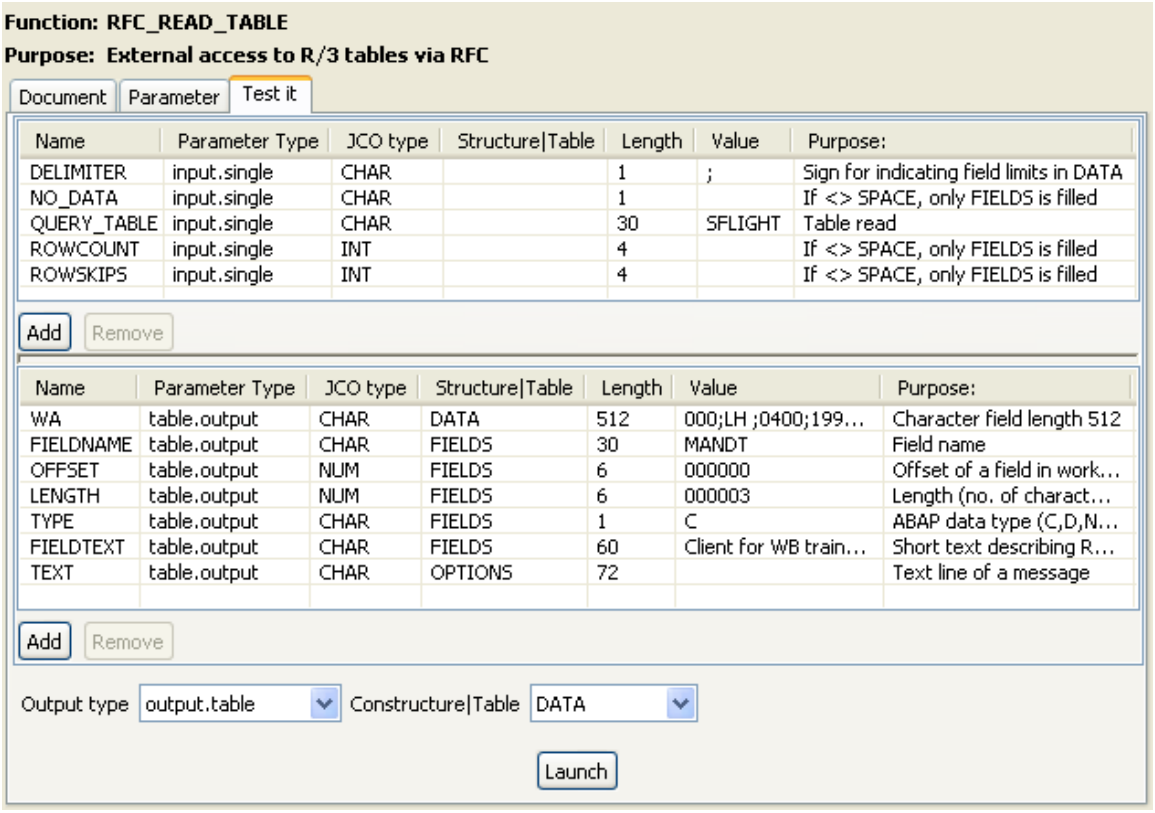
The **Parameter** view provides information about the parameters required by the *RFC_READ_TABLE* function to return parameter values.

Setting the input and output parameters using the wizard

1. In the **Parameter** view, click the **Table** tab to show a description of the structure of the different tables of the *RFC_READ_TABLE* function.



The **Test it** view allows you to add or delete input parameters according to the called function. In this example, we want to retrieve the structure of the *SFLIGHT* table and not any data.



2. In the **Value** column of the **DELIMITER** line, enter ";" as field separator.
3. In the **Value** column of the **QUERY_TABLE** line, enter *SFLIGHT* as the table to query.
4. In the **Output type** list at the bottom of the view, select **output.table**.
5. In the **Constructure|Table** list, select **DATA**.
6. Click **Launch** at the bottom of the view to display the parameter values returned by the *RFC_READ_TABLE* function. In this example, the delimiter is ";" and the table to read is *SFLIGHT*.

7. Click **Finish** to close the wizard and create the connection.

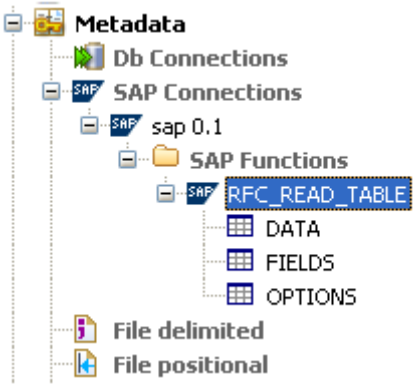
Retrieving the different schemas of the RFC_READ_TABLE function

The *sap* connection and the **RFC_READ_TABLE** function display under the **SAPConnections** node in the **Repository** tree view.

To retrieve the different schemas of the **RFC_READ_TABLE** function, do the following:

1. In the **Repository** tree view, right-click **RFC_READ_TABLE** and select **Retrieve schema** in the contextual menu. A dialog box displays.
2. Select in the list the schemas you want to retrieve, **DATA**, **FIELDS** and **OPTIONS** in this example.
3. Click **Next** to open a new view on the dialog box and display these different schemas.
4. Click **Finish** to validate your operation and close the dialog box.

The three schemas display under the **RFC_READ_TABLE** function in the **Repository** tree view.

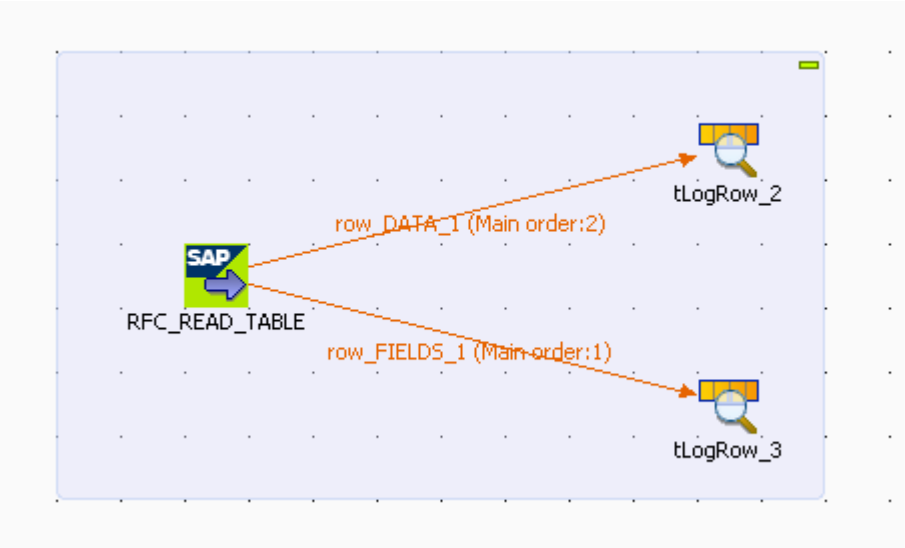


Retrieving the data column names of the SFLIGHT table

In this example, we want to retrieve the data and column names of the **SFLIGHT** table and display them in *Talend Open Studio for Big Data*. To do that, proceed as the following:

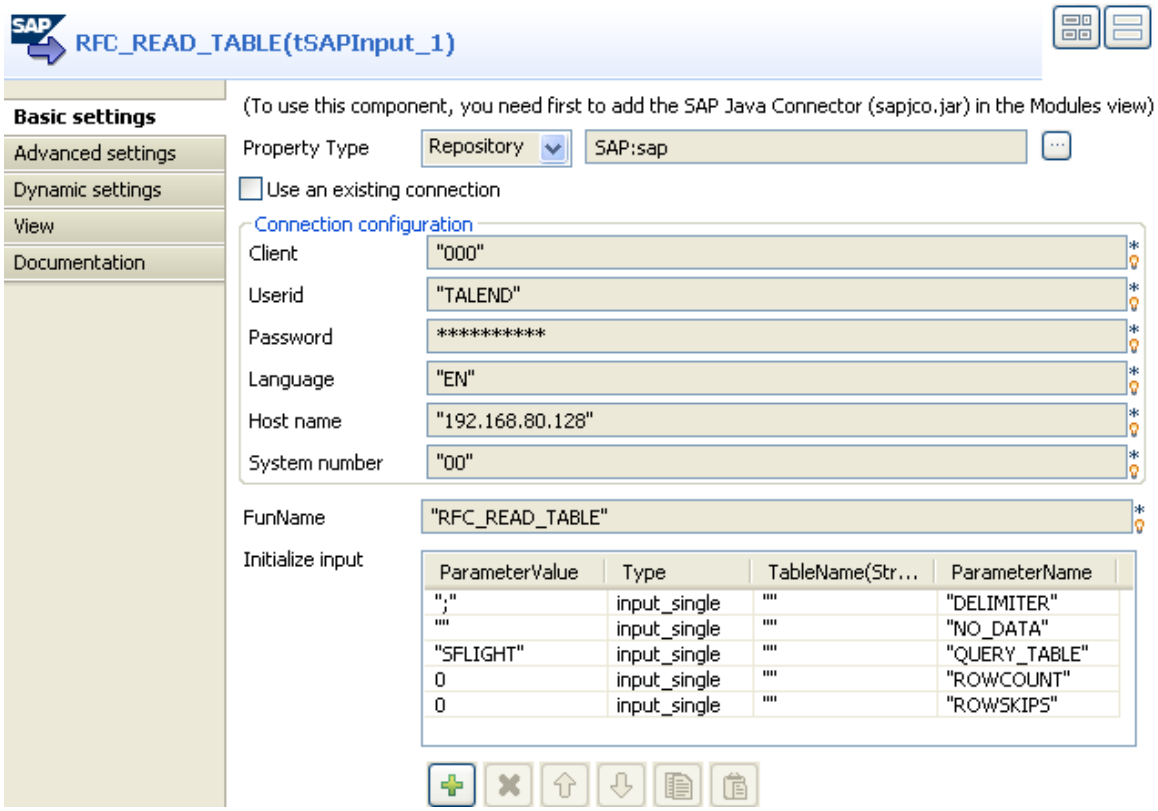
Setting up the Job

1. In the **Repository** tree view, drop the **RFC_READ_TABLE** function of the *sap* connection to the design workspace to open a dialog box where you can select **tSAPInput** from the component list and then click **OK** to close the dialog box. The **tSAPInput** component displays on the design workspace.
2. Drop two **tLogRow** components from the **Palette** to the design workspace.
3. Right-click **tSAPInput** and select **Row** > **row_DATA_1** and click the first **tLogRow** component.
4. Right-click **tSAPInput** and select **Row** > **row_FIELDS_1** and click the second **tLogRow** components.



In this example, we want to retrieve the **FIELDS** and **DATA** schemas and put them in two different output flows.

5. In the design workspace, double-click **tSAPInput** to open the **Basic settings** view and display the component properties.

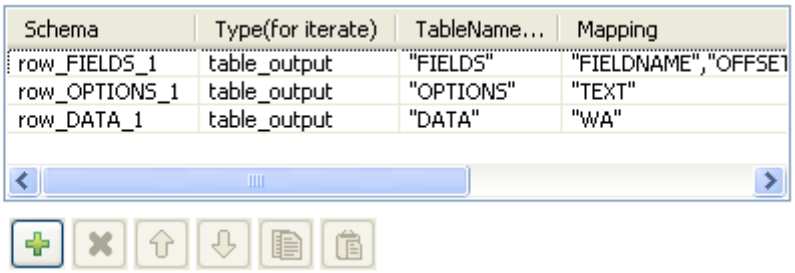


The basic setting parameters for the **tSAPInput** component display automatically since the schema is stored in the **Metadata** node and the component is initialized by the SAP wizard.

In the **Initialize input** area, we can see the input parameters necessary for the *RFC_READ_TABLE* function, the field delimiter ";" and the table name "**SFLIGHT"**.

In the **Outputs** area, we can see the different schemas of the **SFLIGHT** table.

Outputs



Schema	Type(for iterate)	TableName...	Mapping
row_FIELDS_1	table_output	"FIELDS"	"FIELDNAME","OFFSET
row_OPTIONS_1	table_output	"OPTIONS"	"TEXT"
row_DATA_1	table_output	"DATA"	"WA"

Job execution

1. In the design workspace, double click each of the two **tLogRow** components to display the **Basic settings** view and define the component properties. For more information on the properties of **tLogRow**, see section *tLogRow*.

2. Press **CTRL+S** to save your Job and press **F6** to execute it.

```
Starting job jobsap at 16:13 01/10/2009.

[statistics] connecting to socket on port 4093
[statistics] connected
.----------+------+------+----+----------------------------------------.
|                                tLogRow_3                               |
|=---------+------+------+----+----------------------------------------=|
|FIELDNAME |OFFSET|LENGTH|TYPE|FIELDTEXT                               |
|=---------+------+------+----+----------------------------------------=|
|MANDT     |0     |3     |C   |Client for WB train. data model BC_Travel|
|CARRID    |4     |3     |C   |Airline carrier ID                      |
|CONNID    |8     |4     |N   |Flight connection Id                    |
|FLDATE    |13    |8     |D   |Flight date                             |
|PRICE     |22    |15    |P   |Airfare                                 |
|CURRENCY  |38    |5     |C   |Local currency of airline               |
|PLANETYPE |44    |10    |C   |Plane type                              |
|SEATSMAX  |55    |10    |X   |Maximum capacity                        |
|SEATSOCC  |66    |10    |X   |Occupied seats                          |
|PAYMENTSUM|77    |17    |P   |Total of current bookings               |
'----------+------+------+----+----------------------------------------'

.-------------------------------------------------------------------------------.
|                                    tLogRow_2                                  |
|=------------------------------------------------------------------------------=|
|WA                                                                             |
|=------------------------------------------------------------------------------=|
|000;LH ;0400;19950228;  899.00 ;DEM  ;A319      ;5E010000  ;03000000  ;  2639.00|
|000;LH ;0454;19951117;1499.00 ;DEM  ;A319      ;5E010000  ;02000000  ;  2949.00|
|000;LH ;0455;19950606;1090.00 ;USD  ;A319      ;DC000000  ;01000000  ;  1499.00|
|000;LH ;3577;19950428;6000.00 ;LIT  ;A319      ;DC000000  ;01000000  ;   600.00|
|000;SQ ;0026;19950228;  849.00 ;DEM  ;DC-10-10  ;7C010000  ;02000000  ;  1684.00|
'-------------------------------------------------------------------------------'

[statistics] disconnected
Job jobsap ended at 16:13 01/10/2009. [exit code=0]
```

The **tSAPInput** component retrieves from the SAP system the column names of the *SFLIGHT* table as well as the corresponding data. The **tLogRow** components display the information in a tabular form in the Console.

tSAPOutput

tSAPOutput Properties

Component family	Business	
Function	Writes to an SAP system.	
Purpose	Allows to write data into an SAP system.	
Basic settings	*Property type*	. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant connection component on the **Component** list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive. For more information about **Dynamic settings**, see your studio user guide.
	Connection configuration	**Client type**: Enter your SAP usual connection code **Userid:** Enter the user connection Id. **Password**: Enter the password. **Language**: Specify a language. **Host name** Enter the SAP system IP address. **System number** Enter the system number.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
	FunName	Enter the name of the function you want to use to write data.
	Mapping	Set the parameters to select the data to write to the SAP system.
Advanced settings	*Release Connection*	Clear this check box to continue to use the selected connection once the component has performed its task.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as an output component. An input component is required.	
Limitation	n/a	

Related scenario

For a related scenarios, see section *Scenario 1: Retrieving metadata from the SAP system* and section *Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function*.

tSAPRollback

tSAPRollback properties

This component is closely related to **tSAPCommit** and **tSAPConnection**. It usually does not make much sense to use these components separately in a transaction.

Component family	Business/SAP	
Function	**tSAPRollback** cancels the transaction commit in the connected SAP.	
Purpose	**tSAPRollback** avoids to commit only a fragment of a transaction.	
Basic settings	*SAPConnection Component list*	Select the **tSAPConnection** component in the list if more than one connection are planned for the current Job.
	Release Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is intended to be used along with SAP components, especially with **tSAPConnection** and **tSAPCommit**.	
Limitation	n/a	

Related scenarios

For **tSAPRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

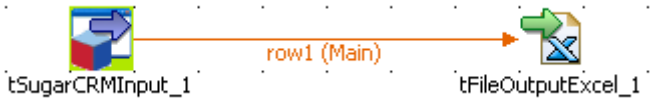
tSugarCRMInput

tSugarCRMInput Properties

Component family	Business/Cloud	
Function	Connects to a Sugar CRM database module via the relevant webservice.	
Purpose	Allows you to extract data from a SugarCRM DB based on a query.	
Basic settings	*SugarCRM Webservice URL*	Type in the webservice URL to connect to the SugarCRM DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Module	Select the relevant module from the list To use customized tables, select **Use custom module** from the list. The **Custom module package name** and **Custom module name** fields which appear are automatically filled in with the relevant names.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**make changes to the schema. In this component the schema is related to the Module selected.
	Query condition	Type in the query to select the data to be extracted. Example: account_name= 'Talend'
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario: Extracting account data from SugarCRM

This scenario describes a two-component Job which extracts account information from a SugarCRM database and writes it to an Excel output file.

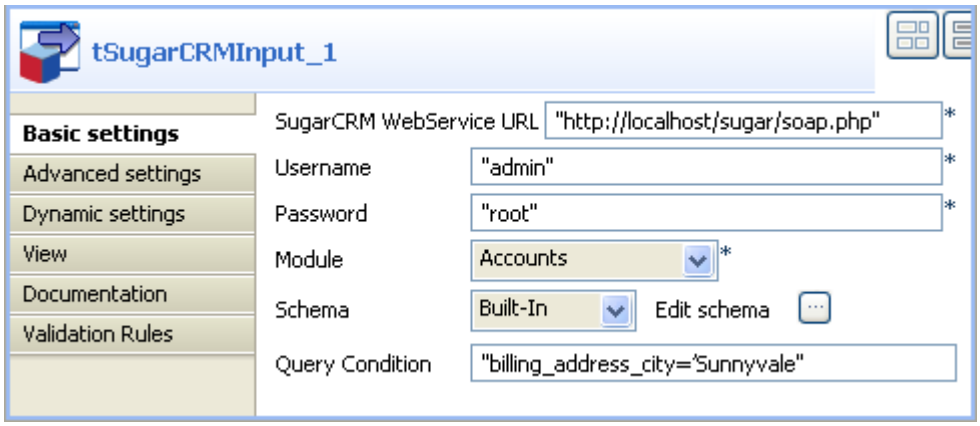


Setting up the Job

1. Drop a **tSugarCRMInput** and a **tFileOutputExcel** component from the **Palette** onto the workspace.
2. Connect the input component to the output component using a **Row** > **Main** connection.

Configuring the input component

1. Double-click **tSugarCRMInput** to define the component properties in its **Basic settings** view.



2. Fill the **SugarCRM WebService URL** field with the connection inforamtion, and the **Username** and **Password** fields with the authentication you have.

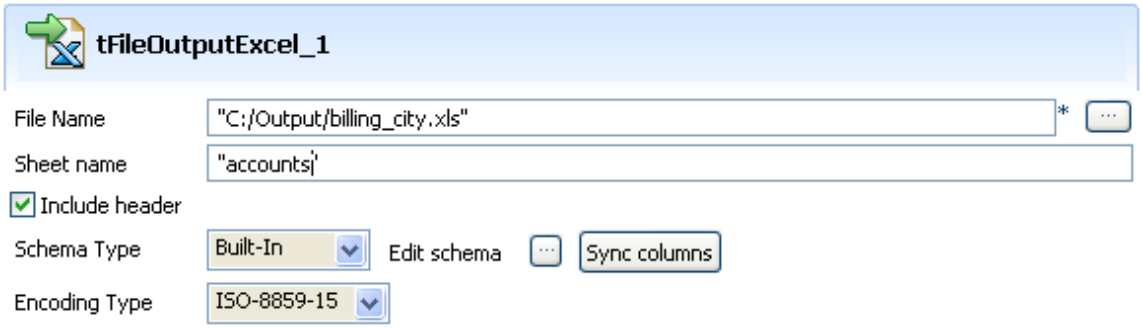
3. Select the **Module** from the list of modules offered. In this example, *Accounts* is selected.

   The **Schema** is then automatically set according to the module selected. But you can change it and remove the columns that you do not require in the output.

4. In the **Query Condition** field, type in the query you want to extract from the CRM. In this example: “billing_address_city=’Sunnyvale’”.

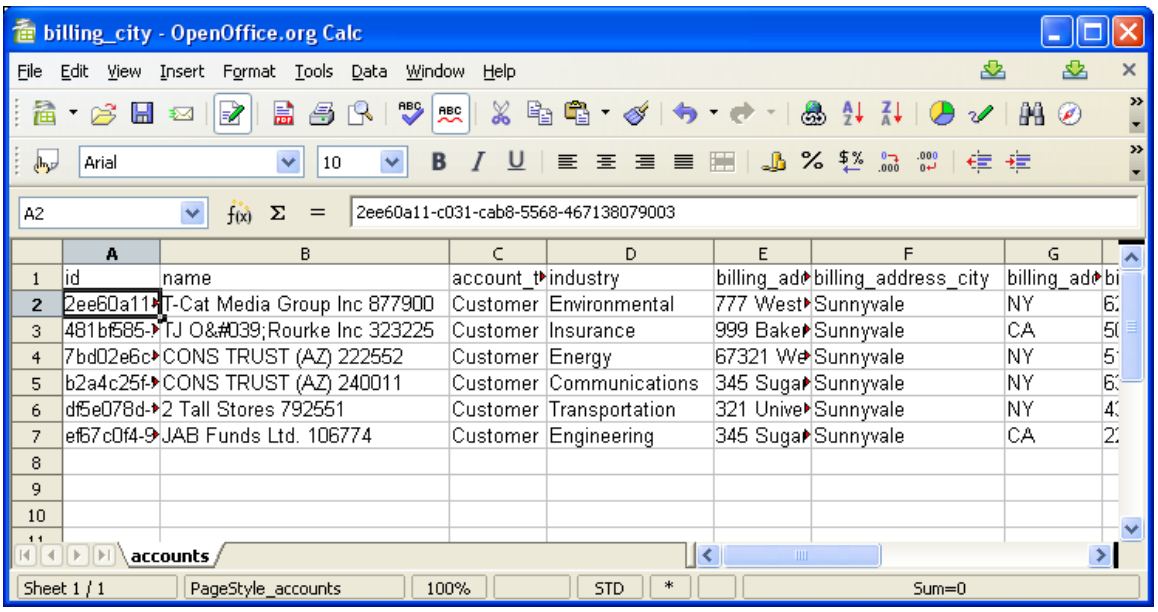
Job execution

1. Double-click **tFileOutputExcel** to define the component properties in its **Basic settings** view.



2. Set the destination file name as well as the **Sheet** name and select the **Include header** check box.

3. Press **CTRL**+**S** to save your Job and press **F6** to execute it.



	A	B	C	D	E	F	G
1	id	name	account_t	industry	billing_ad	billing_address_city	billing_ad
2	2ee60a11	T-Cat Media Group Inc 877900	Customer	Environmental	777 West	Sunnyvale	NY
3	481bf585-	TJ O'Rourke Inc 323225	Customer	Insurance	999 Baker	Sunnyvale	CA
4	7bd02e6c	CONS TRUST (AZ) 222552	Customer	Energy	67321 We	Sunnyvale	NY
5	b2a4c25f-	CONS TRUST (AZ) 240011	Customer	Communications	345 Sugar	Sunnyvale	NY
6	df5e078d-	2 Tall Stores 792551	Customer	Transportation	321 Unive	Sunnyvale	NY
7	ef67c0f4-9	JAB Funds Ltd. 106774	Customer	Engineering	345 Sugar	Sunnyvale	CA
8							
9							
10							

The filtered data is output in the defined spreadsheet of the specified Excel file.

tSugarCRMOutput

tSugarCRMOutput Properties

Component family	Business/Cloud	
Function	Writes in a Sugar CRM database module via the relevant webservice.	
Purpose	Allows you to write data into a SugarCRM DB.	
Basic settings	*SugarCRM WebService URL*	Type in the webservice URL to connect to the SugarCRM DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Module	Select the relevant module from the list To use customized tables, select **Use custom module** from the list. The **Custom module package name** and **Custom module name** fields which appear are automatically filled in with the relevant names.
	Action	Insert or Update the data in the SugarCRM module.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tVtigerCRMInput

tVtigerCRMInput Properties

Component family	Business/VtigerCRM	
Function	Connects to a module of a VtigerCRM database.	
Purpose	Allows to extract data from a VtigerCRM DB.	
Basic settings		
Vtiger Version	Select the version of the Vtiger Web Services you want to use (either **Vtiger 5.0** or **Vtiger 5.1**)	
Vtiger 5.0	*Server Address*	Type in the IP address of the VtigerCRM server
	Port	Type in the Port number to access the server
	Vtiger Path	Type in the path to access the VtigerCRM server
	Username and *Password*	Type in the user authentication data.
	Version	Type in the version of VtigerCRM you are using.
	Module	Select the relevant module in the list
	Method	Select the relevant method in the list. The method specifies the action you can carry out on the VtigerCRM module selected.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. In this component the schema is related to the Module selected.
Vtiger 5.1	*Endpoint*	Type in the URL address of the invoked Web server.
	Username	Type in the user name to log in to the vTigerCRM..
	Access key	Type in the access key for the user name.
	Query condition	Type in the query to select the data to be extracted.
	Manual input of SQL query	Manually type in your query in the corresponding field.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tVtigerCRMOutput

tVtigerCRMOutput Properties

Component family	Business/VtigerCRM	
Function	Writes data into a module of a VtigerCRM database.	
Purpose	Allows to write data from a VtigerCRM DB.	
Basic settings		
Vtiger Version	Select the version of the Vtiger Web Services you want to use (either **Vtiger 5.0** or **Vtiger 5.1**)	
Vtiger 5.0	*Server Address*	Type in the IP address of the VtigerCRM server.
	Port	Type in the Port number to access the server.
	Vtiger Path	Type in the path to access the server.
	Username and *Password*	Type in the user authentication data.
	Version	Type in the version of VtigerCRM you are using.
	Module	Select the relevant module in the list
	Method	Select the relevant method in the list. The method specifies the action you can carry out on the VtigerCRM module selected.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. In this component the schema is related to the Module selected.
Vtiger 5.1	*Endpoint*	Type in the URL address of the invoked Web server.
	Username	Type in the user name to log in to the VtigerCRM..
	Access key	Type in the access key for the user name.
	Action	Insert or Update the data in the SugarCRM module.
	Module	Select the relevant module in the list
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. In this component the schema is related to the Module selected.
	Die on error	This check box is clear by default to skip the row on error and complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

Business Intelligence components

This chapter details the main components which belong to the **Business Intelligence** family in the **Palette** of the **Integration** perspective of the **Talend** Studio.

The BI family groups connectors that cover needs such as reading or writing multidimensional or OLAP databases, outputting Jasper reports, tracking DB changes in slow changing dimension tables and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tDB2SCD

tDB2SCD properties

Component family	Databases/DB2	
Function	**tDB2SCD** reflects and tracks changes in a dedicated DB2 SCD table.	
Purpose	**tDB2SCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Table Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs.

		For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see section *tMysqlSCD*.

tDB2SCDELT

tDB2SCDELT Properties

Component family	Databases/DB2	
Function	**tDB2SCDELT** reflects and tracks changes in a dedicated DB2 SCD table.	
Purpose	**tDB2SCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated DB2 SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input DB2 SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created. **Create table if not exists**: A table gets created if it does not exist.

		Clear table: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	Source Keys	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Debug mode*	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related Scenario

For related topics, see section *tDB2SCD* and section *tMysqlSCD*.

tGreenplumSCD

tGreenplumSCD Properties

Component family	Databases/Greenplum	
Function	**tGreenplumSCD** reflects and tracks changes in a dedicated Greenplum SCD table.	
Purpose	**tGreenplumSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Select the relevant driver on the list.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs.

		For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related scenarios, see section *tMysqlSCD*.

tInformixSCD

tInformixSCD properties

Component family	Databases/Business Intelligence/Informix	
Function	**tInformixSCD** tracks and shows changes which have been made to Informix SCD dedicated tables.	
Purpose	**tInformixSCD** addresses Slowly Changing Dimension transformation needs, by regularly reading a data source and listing the modifications in an SCD dedicated table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	DB server listening port.
	Database	Name of the database.
	Schema	Name of the schema.
	Username et *Password*	User authentication information.
	Instance	Name of the Informix instance to be used. This information can generally be found in the SQL hosts file.
	Table	Name of the table to be created
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs.

		For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to improve system performance.
	Use Transaction	Select this check box when the database is configured in NO_LOG mode.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
	Debug mode	Select this check box to display each step of the process by which data is written in the database.
Usage	This component is an output component. Consequently, it requires an input component and a connection of the **Row > Main** type.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which **tInformixSCD** might be used, see section *tMysqlSCD*.

tIngresSCD

tIngresSCD Properties

Component family	Databases/Ingress	
Function	**tIngresSCD** reflects and tracks changes in a dedicated Ingres SCD table.	
Purpose	**tIngresSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your Studio user guide.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.

	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related scenarios, see section *tMysqlSCD*.

tMSSqlSCD

tMSSqlSCD Properties

Component family	Databases/MSSQL Server	
Function	**tMSSqlSCD** reflects and tracks changes in a dedicated MSSQL SCD table.	
Purpose	**tMSqlSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Schema	Name of the DB schema.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.

	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related topics, see section *tMysqlSCD*.

tMysqlSCD

tMysqlSCD Properties

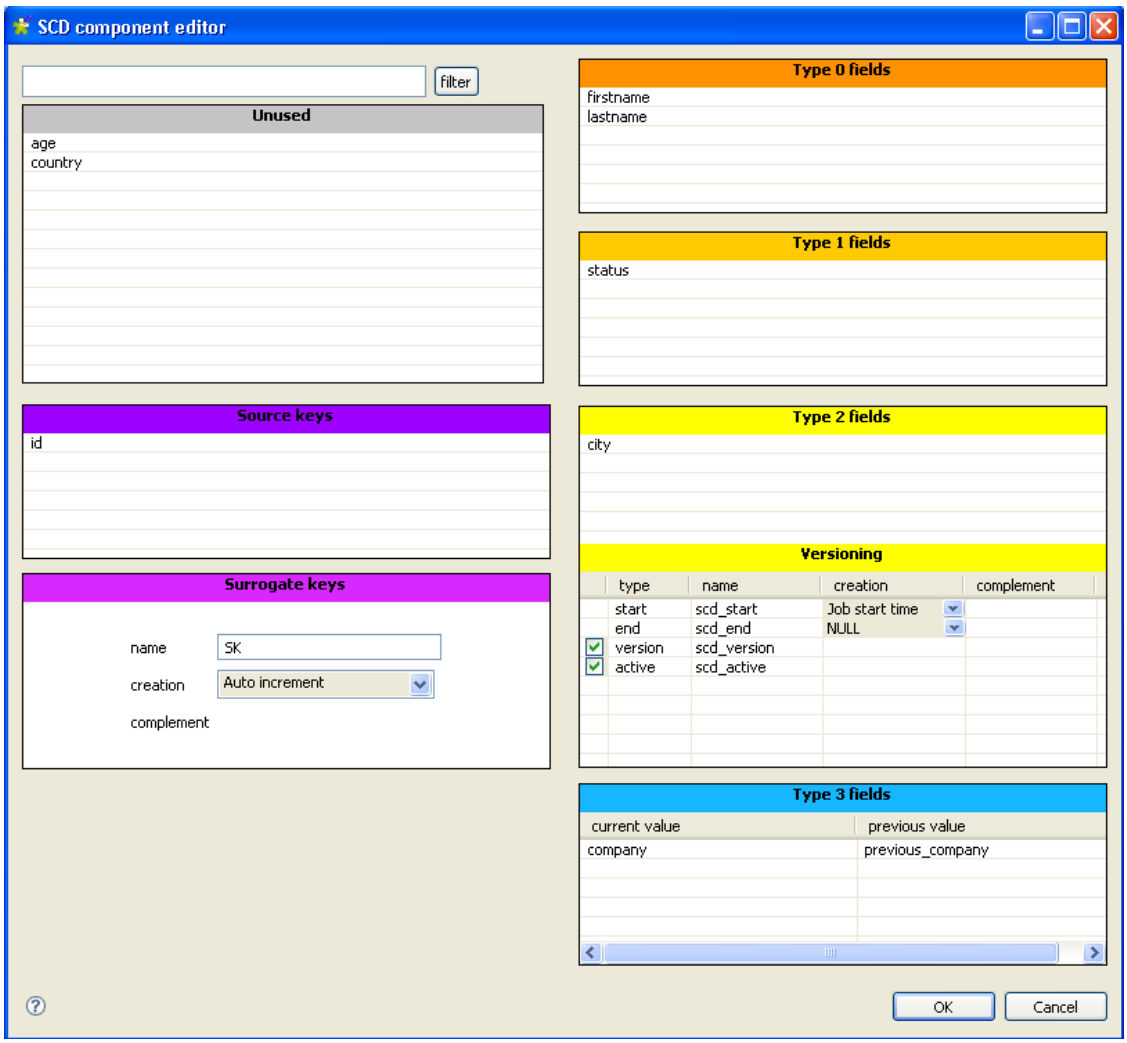
Component family	Databases/MySQL	
Function	**tMysqlSCD** reflects and tracks changes in a dedicated MySQL SCD table.	
Purpose	**tMysqlSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	DB Version	Select the Mysql version you are using.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.

		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.
	Use memory saving mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Additional JDBC Parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

SCD management methodologies

Slowly Changing Dimensions (SCDs) are dimensions that have data that slowly changes. The SCD editor offers the simplest method of building the data flow for the SCD outputs. In the SCD editor, you can map columns, select surrogate key columns, and set column change attributes through combining SCD types.

The following figure illustrates an example of the SCD editor.



SCD keys

You must choose one or more source keys columns from the incoming data to ensure its unicity.

You must set one surrogate key column in the dimension table and map it to an input column in the source table. The value of the surrogate key links a record in the source to a record in the dimension table. The editor uses this mapping to locate the record in the dimension table and to determine whether a record is new or changing. The surrogate key is typically the primary key in the source, but it can be an alternate key as long as it uniquely identifies a record and its value does not change.

Source keys: Drag one or more columns from the **Unused** panel to the **Source keys** panel to be used as the key(s) that ensure the unicity of the incoming data.

Surrogate keys: Set the column where the generated surrogate key will be stored. A surrogate key can be generated based on a method selected on the **Creation** list.

Creation: Select any of the below methods to be used for the key generation:

Auto increment: auto-incremental key.

Input field: key is provided in an input field.

When selected, you can drag the appropriate field from the **Unused** panel to the **complement** field.

Routine: from the **complement** field, you can press **Ctrl+ Space** to display the autocompletion list and select the appropriate routine.

Table max +1: the maximum value from the SCD table is incremented to create a surrogate key.

DB Sequence: from the **complement** field, you can enter the name of the existing database sequence that will automatically increment the column indicated in the **name** field.

This option is only available through the **SCD Editor** of the **tOracleSCD** component.

Combining SCD types

The Slowly Changing Dimensions support four types of changes: **Type 0** through **Type 3**. You can apply any of the SCD types to any column in a source table by a simple drag-and-drop operation.

Type 0: is not used frequently. Some dimension data may be overwritten and other may stay unchanged over time. This is most appropriate when no effort has been made to deal with the changing dimension issues.

Type 1: no history is kept in the database. New data overwrites old data. Use this type if tracking changes is not necessary. this is most appropriate when correcting certain typos, for example the spelling of a name.

Type2: the whole history is stored in the database. This type tracks historical data by inserting a new record in the dimensional table with a separate key each time a change is made. This is most appropriate to track updates, for example.

SCD **Type 2** principle lies in the fact that a new record is added to the SCD table when changes are detected on the columns defined. Note that although several changes may be made to the same record on various columns defined as SCD **Type 2**, only one additional line tracks these changes in the SCD table.

The SCD schema in this type should include SCD-specific extra columns that hold standard log information such as:

-start: adds a column to your SCD schema to hold the start date. You can select one of the input schema columns as a start date in the SCD table.

-end: adds a column to your SCD schema to hold the end date value for a record. When the record is currently active, the end date is **NULL** or you can select **Fixed Year Value** and fill in a fictive year to avoid having a null value in the end date field.

-version: adds a column to your SCD schema to hold the version number of the record.

-active: adds a column to your SCD schema to hold the **true** or **false** status value. this column helps to easily spot the active record.

Type 3: only the information about a previous value of a dimension is written into the database. This type tracks changes using separate columns. This is most appropriate to track only the previous value of a changing column.

Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3)

This five-component Java scenario describes a Job that tracks changes in four of the columns in a source delimited file, writes changes and the history of changes in an SCD table, and displays error information on the **Run** console.

The source delimited file contains various personal details including *firstname*, *lastname*, *address*, *city*, *company*, *age*, and *status*. An *id* column helps ensuring the unicity of the data.

	A	B	C	D	E	F
1	id;firstname;lastname;address;city;company;age;status					
2	1;Janet;Anderson;511 Maple Ave. Apt. 1B;Agoura Hills;Adecco;30;married					
3	2;Harold;Smith;662 Lyons Circle;Alameda;Adidas;33;married					
4	3;Adam;Brown;220 Vine Ave.;Albany;Bridgestone;28;single					
5	4;Martin;Clinton;770 Exmoor Rd.;Alhambra;Cutco;25;single					
6	5;Sue;White;1486 Oakwood;Covina;Black&White;35;married					

We want any change in the marital status to overwrite the existing old status record. This type of change is equivalent to an SCD **Type 1**.

We want to insert a new record in the dimensional table with a separate key each time a person changes his/her company. This type of change is equivalent to an SCD **Type 2**.

We want to track only the previous city and previous address of a person. This type of change is equivalent to an SCD **Type 3**.

To realize this kind of scenario, it is better to divide it into three main steps: defining the main flow of the Job, setting up the SCD editor, and finally creating the relevant SCD table in the database.

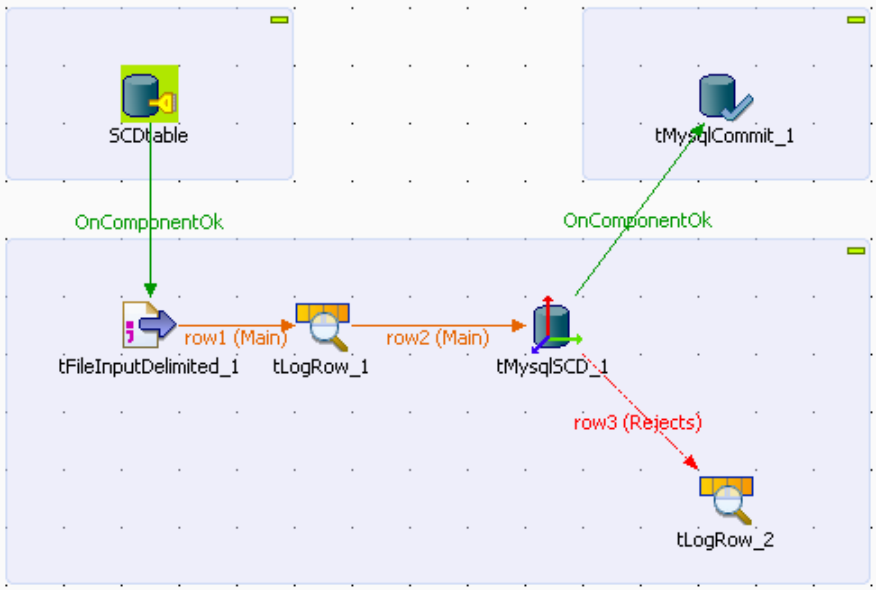
Defining the main flow of the Job

1. Drop the following components from the **Palette** onto the design workspace: a **tMysqlConnection**, a **tFileInputDelimited**, a **tMysqlSCD**, a **tMysqlCommit**, and two **tLogRow** components.

2. Connect the **tFileInputDelimited**, the first **tLogRow**, and the **tMysqlSCD** using the **Row Main** link. This is the main flow of your Job.

3. Connect the **tMysqlConnection** to the **tFileInputDelimited** and **tMysqlSCD** to **tMysqlCommit** using the **OnComponntOk** trigger.

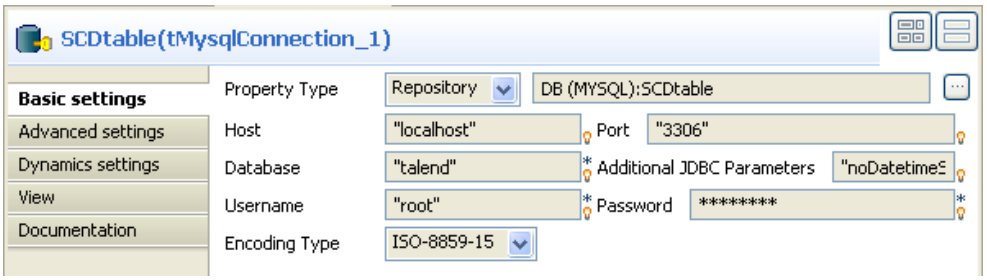
4. Connect the **tMysqlSCD** to the second **tLogRow** using the **Row Rejects** link. Two columns, *errorCode* and *errorMessage*, are added to the schema. This connection collects error information.



Configuring the DB connection and the input component

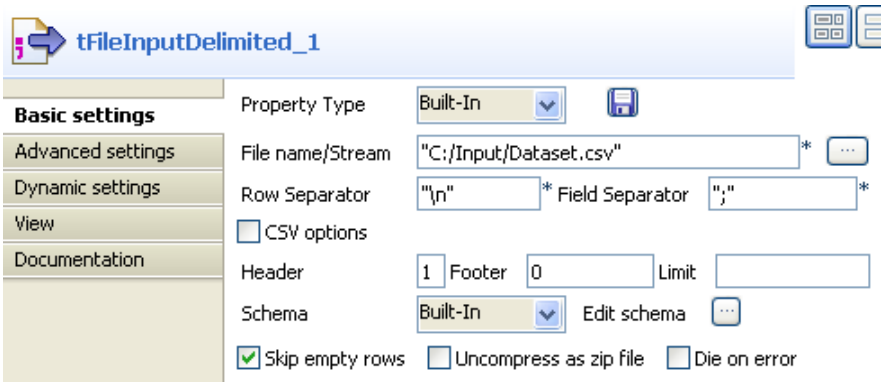
1. In the design workspace, double-click **tMysqlConnection** to display its **Basic settings** view and set the database connection details. The **tMysqlConnection** component should be used to avoid setting several times the same DB connection when multiple DB components are used.

   In this scenario, we want to connect to the SCD table where changes in the source delimited file will be tracked down.



   *For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

2. In the design workspace, double-click **tFileInputDelimited** to display its **Basic settings** view.



3. Click the three-dot button next to the **File Name** field to select the path to the source delimited file, *dataset.csv* in this scenario, that contains the personal details.

4. Define the row and field separators used in the source file.

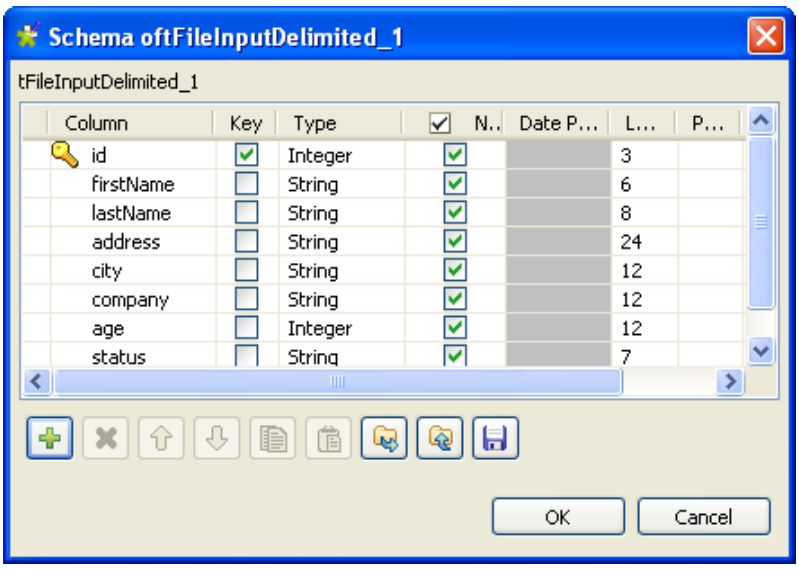
   The **File Name**, **Row separator**, and **Field separators** are mandatory.

5. If needed, set **Header**, **Footer**, and **Limit**.

   In this scenario, set **Header** to *1*. Footer and limit for the number of processed rows are not set.

6. Click **Edit schema** to describe the data structure of the source delimited file.

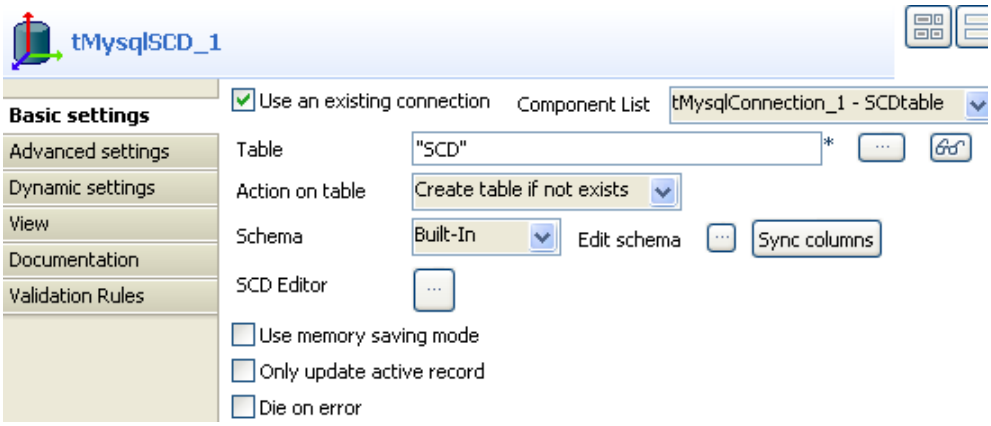
   In this scenario, the source schema is made of eight columns: *id*, *firstName*, *lastName*, *address*, *city*, *company*, *age*, and *status*.



7. Define the basic settings for the first **tLogRow** in order to view the content of the source file with varying attributes in cells of a table on the console before being processed through the SCD component.

Configuring tMysqlSCD and tMysqlCommit

1. In the design workspace, click the **tMysqlSCD** and select the **Component** tab to define its basic settings.



2. In the **Basic settings** view, select the **Use an existing connection** check box to reuse the connection details defined on the **tMysqlConnection** properties.

3. In the **Table** field, enter the table name to be used to track changes.

4. If needed, click **Sync columns** to retrieve the output data structure from the **tFileInputDelimited**.

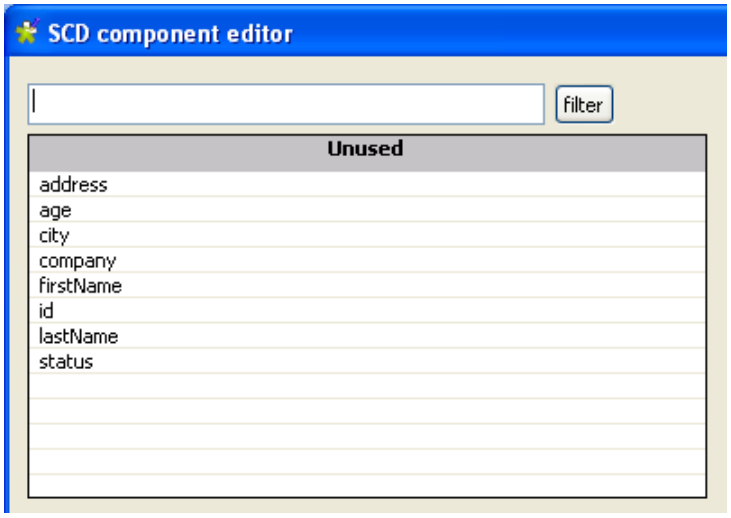
5. In the design workspace, double-click **tMysqlCommit** to define its basic settings.

6. Select the relevant connection on the **Component list** if more than one connection exists.

7. Define the basic settings of the second **tLogRow** in order to view reject information in cells of a table.

Setting up the SCD editor

1. Double-click the **tMysqlSCD** component in the design workspace or click the three-dot button next to the **SCD Editor** in the component's **Basic settings** view to open the **SCD editor** and build the data flow for the SCD outputs.



   All the columns from the preceding component are displayed in the **Unused** panel of the **SCD editor**. All the other panels in the **SCD editor** are empty.

2. From the **Unused** list, drop the *id* column to the **Source keys** panel to use it as the key to ensure the unicity of the incoming data.

3. In the **Surrogate keys** panel, enter a name for the surrogate key in the **Name** field, *SK1* in this scenario.

4. From the **Creation** list, select the method to be used for the surrogate key generation, **Auto-increment** in this scenario.

5. From the **Unused** list, drop the *firstname* and *lastname* columns to the **Type 0** panel, changes in these two columns do not interest us.

6. Drop the *status* column to the **Type 1** panel. The new value will overwrite the old value.

7. Drop the *company* column to the **Type 2** panel. Each time a person changes his/her company, a new record will be inserted in the dimensional table with a separate key.

   In the **Versioning** area:

   - Define the **start** and **end** columns of your SCD table that will hold the start and end date values. The end date is null for current records until a change is detected. Then the end date gets filled in and a new record is added with no end date.

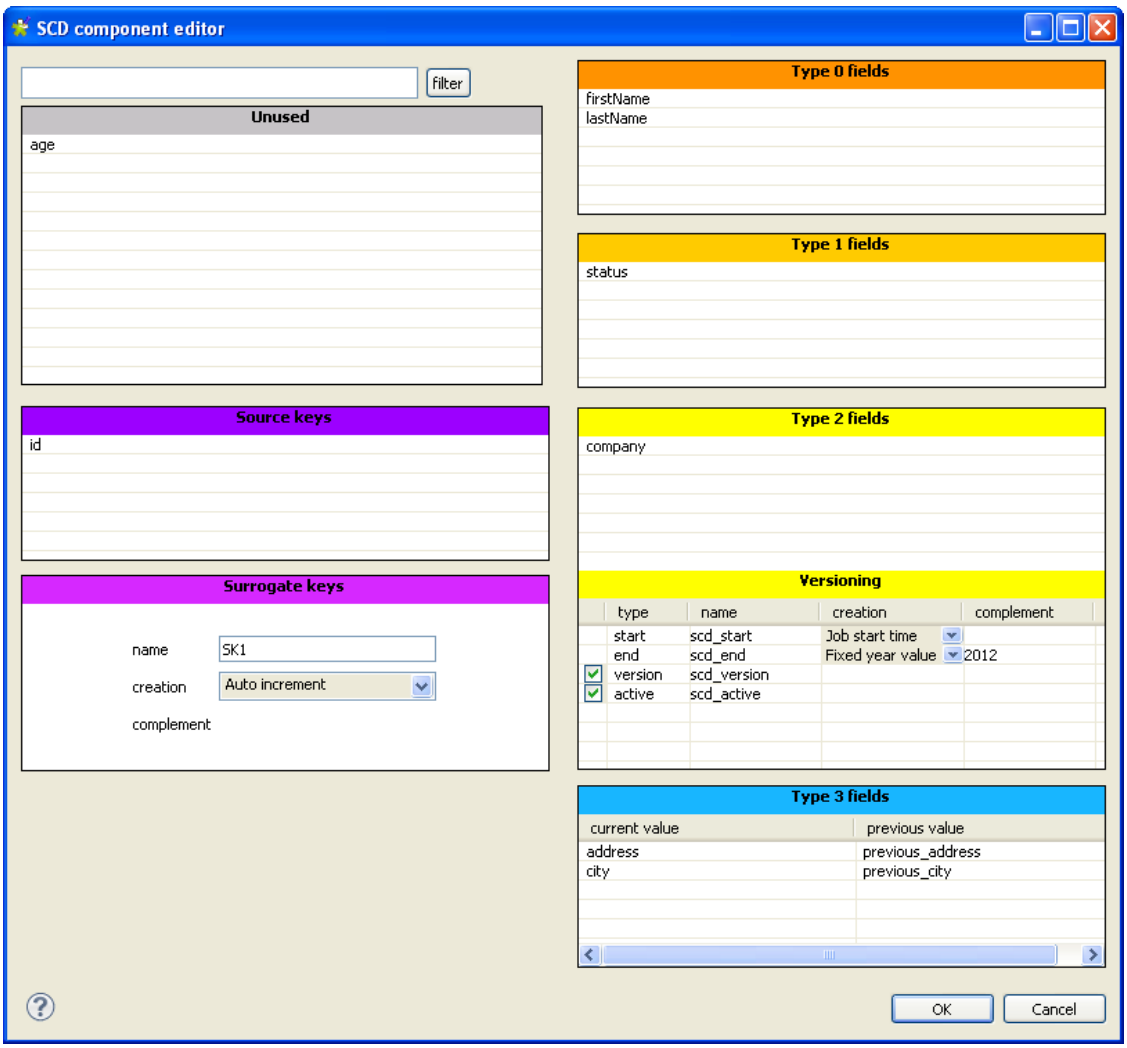
   In this scenario, we select **Fixed Year Value** for the **end** column and fill in a fictive year to avoid having a null value in the end date field.

   - Select the **version** check box to hold the version number of the record.

   - Select the **active** check box to spot the column that will hold the **True** or **False** status. **True** for the current active record and **False** for the modified record.

8. Drop the *address* and *city* columns to the **Type 3** panel to track only the information about the previous value of the address and city.

   For more information about SCD types, see section *SCD management methodologies*.

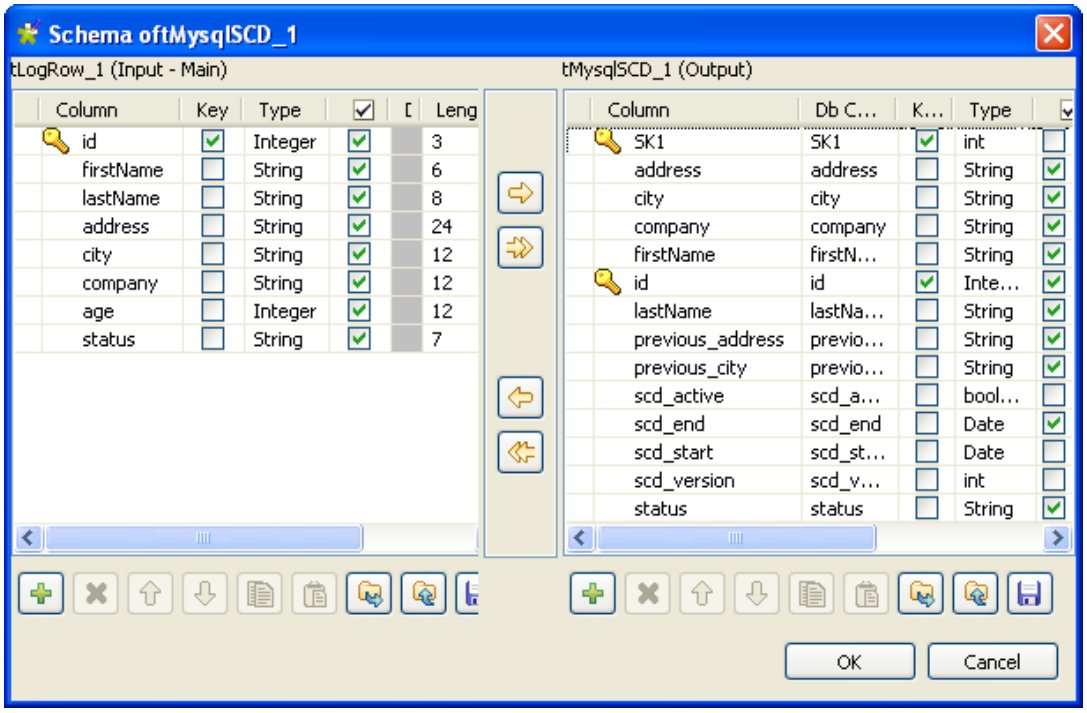


9. Click **OK** to validate your configuration and close the **SCD editor**.

Creating the SCD table

1. Click **Edit schema** to view the input and output data structures.

   The SCD output schema should include the SCD-specific columns defined in the **SCD editor** to hold standard log information.



If you adjust any of the input schema definitions, you need to check, and reconfigure if necessary, the output flow definitions in the **SCD editor** to ensure that the output data structure is properly updated.

2. In the **Basic settings** view of the **tMysqlSCD** component, select **Create table if not exists** from the **Action on table** list to avoid creating and defining the SCD table manually.

Job execution

Save your Job and press **F6** to execute it.

The console shows the content of the input delimited file, and your SCD table is created in your database, containing the initial dataset.

SK1	address	city	company	firstName	id	lastName	previous_ad	previous_city	scd_ac	scd_end	scd_st	scd_ve	status
1	511 Maple Ave. Apt. 1B	Agoura Hills	Adecco	Janet	1	Anderson	(NULL)	(NULL)	□	2012-01-	2011-0	1	married
2	662 Lyons Circle	Alameda	Adidas	Harold	2	Smith	(NULL)	(NULL)	□	2012-01-	2011-0	1	married
3	220 Vine Ave.	Albany	Bridgestone	Adam	3	Brown	(NULL)	(NULL)	□	2012-01-	2011-0	1	single
4	770 Exmoor Rd.	Alhambra	Cutco	Martin	4	Clinton	(NULL)	(NULL)	□	2012-01-	2011-0	1	single
5	1486 Oakwood	Covina	Black&White	Sue	5	White	(NULL)	(NULL)	□	2012-01-	2011-0	1	married
JLL)	(NULL)	(NULL)	(NULL)	(NULL)	0	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Janet gets divorced and moves to *Adelanto* at *355 Golf Rd.* She works at *Greenwood.*

Adam gets married and moves to *Belmont* at *2505 Alisson ct.* He works at *Scoop.*

Martin gets a new job at *Phillips and Brothers*.

Update the delimited file with the above information and press **F6** to run your Job.

The console shows the updated personal information and the rejected data, and the SCD table shows the history of valid changes made to the input file along with the status and version number. Because the name of Martin's new company exceeds the length of the column *company* defined in the schema, this change is directed to the reject flow instead of being logged in the SCD table.

SK1	address	city	company	firstName	id	lastName	previous_ad	previous_city	scd_ac	scd_end	scd_st	scd_ve	status
1	511 Maple Ave. Apt. 1B	Agoura Hills	Adecco	Janet	1	Anderson	(NULL)	(NULL)	(Binary/	2011-02-	2011-0	1	single
2	662 Lyons Circle	Alameda	Adidas	Harold	2	Smith	(NULL)	(NULL)	□	2012-01-	2011-0	1	married
3	220 Vine Ave.	Albany	Bridgestone	Adam	3	Brown	(NULL)	(NULL)	(Binary/	2011-02-	2011-0	1	married
4	770 Exmoor Rd.	Alhambra	Cutco	Martin	4	Clinton	(NULL)	(NULL)	(Binary/	2011-02-	2011-0	1	single
5	1486 Oakwood	Covina	Black&White	Sue	5	White	(NULL)	(NULL)	□	2012-01-	2011-0	1	married
6	355 Golf Rd.	Adelanto	Greenwood	Janet	1	Anderson	511 Maple Av	Agoura Hills	□	2012-01-	2011-0	2	single
7	2505 Alisson ct	Belmont	Scoop	Adam	3	Brown	220 Vine Ave.	Albany	□	2012-01-	2011-0	2	married
JLL)	(NULL)	(NULL)	(NULL)	(NULL)	0	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

tMysqlSCDELT

tMysqlSCDELT Properties

Component family	Databases/MySQL	
Function	**tMysqlSCDELT** reflects and tracks changes in a dedicated MySQL SCD table.	
Purpose	**tMysqlSCD**ELT addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated MySQL SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	DB Version	Select the Mysql version you are using.
	Use an existing connection	Select this check box and click the relevant **tMySqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input MySQL SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create the table**: The table is removed and created again **Create a table**: A new table gets created.

		Create a table if not exists: A table gets created if it does not exist. **Clear a table**: The table content is deleted. You have the possibility to rollback the operation. **Truncate a table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	Source Keys	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the strat date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Debug mode*	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see: section *tMysqlSCD* and section *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3)*.

tOracleSCD

tOracleSCD Properties

Component family	Databases/Oracle	
Function	**tOracleSCD** reflects and tracks changes in a dedicated Oracle SCD table.	
Purpose	**tOracleSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Select the relevant driver on the list.
	DB Version	Select the Oracle version you are using.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	Select to perform one of the following operations on the table defined: - None: No action is carried out on the table. - Create table: A new table is created. - Create table if not exists: A table is created if it does not exist.

	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related scenarios, see section *tMysqlSCD*.

tOracleSCDELT

tOracleSCDELT Properties

Component family	Databases/Oracle	
Function	**tOracleSCDELT** reflects and tracks changes in a dedicated Oracle SCD table.	
Purpose	**tOracleSCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated DB2 SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	Use an existing connection	Select this check box and click the relevant **tOracleConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Select the relevant driver on the list.
	DB Version	Select the Oracle version you are using.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input DB2 SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created.

		Create table if not exists: A table gets created if it does not exist. **Clear table**: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	*Source Key*s	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see section *tOracleSCD* and section *tMysqlSCD*.

tPaloCheckElements

tPaloCheckElements Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component checks whether elements are present in an incoming data flow existing in a given cube.	
Purpose	This component can be used along with **tPaloOutputMulti**. It checks if the elements from the input stream exist in the given cube, before writing them. It can also define a default value to be used for nonexistent elements.	
Basic settings	*Use an existing connection*	Select this check box and choose the relevant DB connection component from the **Connection configuration** list to use predefined connection details. When a Job contains a parent Job and a child Job, **Connection configuration** only lists the connection components on the same Job level, so if you need to use an existing connection from another level, ensure that the connection components available are sharing the connection required. For further information about sharing DB connections across Job levels, refer to **Use or register a shared DB connection** in the properties table of the relevant connection component in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can deactivate the connection components and use the component's **Dynamic settings** to define the connection manually. In this case, ensure that the connection name is not used elsewhere in the job, on any level. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database in which the data is to be written.
	Cube	Type in the name of the cube in which the data should be written.
	On element error	Select what should happen if an element does not exist: - **Reject row**: the corresponding row is rejected and placed in the reject flow. - **Use default**: the defined Default value is used. - **Stop**: the entire process is interrupted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.

		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Define the elements to be checked in the table provided. - **Column**: shows the column(s) from the input schema. It is completed automatically once a schema is retrieved or created. - **Element type**: select the element type for the input column. Only one column can be defined as **Measure**. - **Default**: type in the default value to be used if you have selected the **Use default** option in the **On element error** field.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data on the component level.
Usage	This component requires an input component.	
Connections		Outgoing links (from one component to another): **Row**: Main; Rejects **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Rejects For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	This component only works on **Normal** Palo cubes. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario 2: Rejecting inflow data when the elements to be written do not exist in a given cube*.

tPaloConnection

tPaloConnection Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component opens a connection to a Palo Server and keeps it open throughout the duration of the process it is required for. Every other Palo component used in the process is able to use this connection.	
Purpose	This component allows other components involved in a process to share its connection to a Palo server for the duration of the process.	
Basic settings	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is used along with Palo components to offer a shared connection to a Palo server.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if, On Subjob Ok, On Subjob Error, On Component Ok, On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related scenarios, see section *Scenario: Creating a dimension with elements*.

tPaloCube

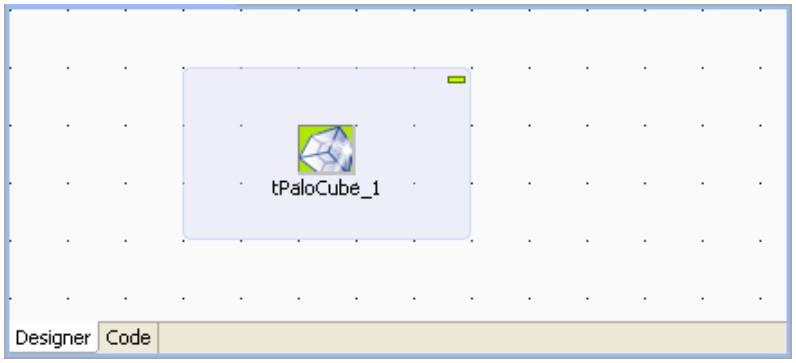
tPaloCube Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component creates, deletes or clears Palo cubes from existing dimensions in a Palo database.	
Purpose	This component performs operations on a given Palo cube.	
Basic settings	*Use an existing connection*	Select this check box and choose the relevant DB connection component from the **Connection configuration** list to reuse predefined connection details. When a Job contains a parent Job and a child Job, **Connection configuration** only lists the connection components on the same Job level, so if you need to use an existing connection from another level, ensure that the connection components available are sharing the connection required. For further information about sharing DB connections across Job levels, refer to **Use or register a shared DB connection** in the properties table of the relevant connection component in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can deactivate the connection components and use the component's **Dynamic settings** to define the connection manually. In this case, ensure that the connection name is not used elsewhere in the job, on any level. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database in which the operation is to take place.
	Cube	Type in the name of the cube where the operation is to take place.
	Cube type	From the drop-down list, select the type of cube on which the operation is to be carried out: - **Normal**: this is the normal and default type of cube. - **Attribut**: an **Attribute** cube is created with a normal cube. - **User Info**: **User Info** cubes can be created/modified with this component.
	Action on cube	Select the operation you want to carry out on the cube defined: **- Create cube**: the cube does not exist and will be created. **- Create cube if not exists**: the cube is created if it does not exist.

		- **Delete cube if exists and create**: the cube is deleted if it already exists and a new one will be created. - **Delete cube**: the cube is deleted from the database. - **Clear cube**: the data is cleared from the cube.
	Dimension list	Add rows and enter the name of existing database dimension's to be used in the cube. The order of the dimensions in the list determines the order of the dimensions created.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	Can be used as a standalone component for dynamic cube creation with a defined dimension list.	
Global Variables		**Cubename**: Indicates the name of the cube processed. This is available as an **After** variable. Returns a String. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The cube creation process does not create dimensions from scratch, so the dimensions to be used in the cube must be created beforehand. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Creating a cube in an existing database

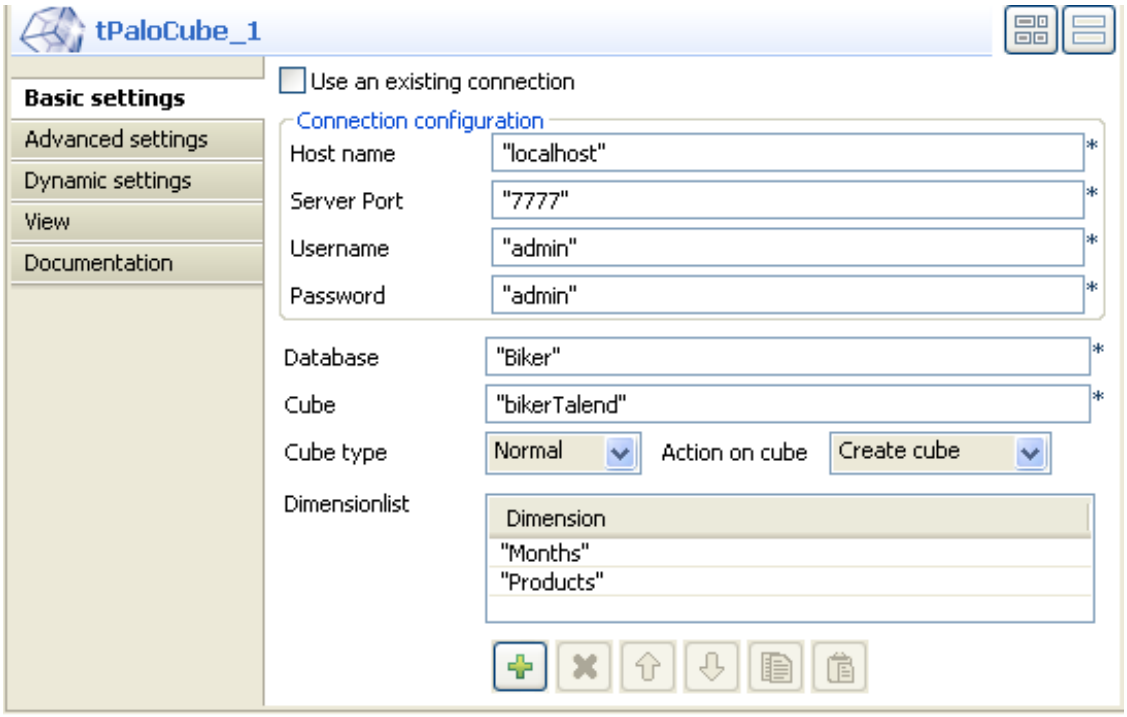
The Job in this scenario creates a new two dimensional cube in the Palo demo database *Biker*.



To replicate this scenario, proceed as follows:

Configuring the tPaloCube component

1. Drop **tPaloCube** from the **Palette** onto the design workspace.
2. Double-click **tPaloCube** to open its **Component** view.

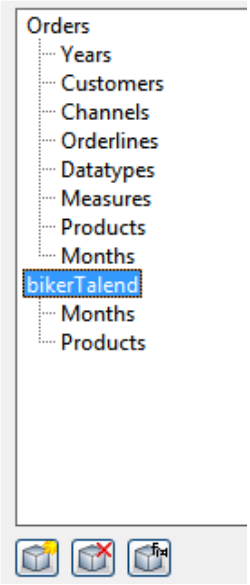


3. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
4. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
5. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
6. In the **Database** field, type in the database name in which you want to create the cube, *Biker* in this example.
7. In the **Cube** field, type in the name you want to use for the cube to be created, for example, *bikerTalend*.
8. In the **Cube type** field, select the **Normal** type from the drop-down list for the cube to be created, meaning this cube will be normal and default.
9. In the **Action on cube** field, select the action to be performed. In this scenario, select **Create cube**.
10. Under the **Dimension list** table, click the plus button twice to add two rows into the table.
11. In the **Dimension list** table, type in the name for each newly added row to replace the default row name. In this scenario, type in *Months* for the first row and *Products* for the second. These two dimensions exist already in the Biker database where the new cube will be created.

Job execution

Press **F6** to run the Job.

A new cube has been created in the *Biker* database and the two dimensions are added into this cube.


Orders
Years
Customers
Channels
Orderlines
Datatypes
Measures
Products
Months
bikerTalend
Months
Products

tPaloCubeList

tPaloCubeList Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component retrieves a list of cube details from the given Palo database.	
Purpose	This component lists cube names, cube types, number of assigned dimensions, the number of filled cells from the given database.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database whose cube details you want to retrieve.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used as a start component. It requires an output component.	
Global Variables		**Number of cubes**: indicates the number of the cubes processed from the given database. This is available as an **After** variable. Returns an Integer **Cube_ID**: indicates the IDs of the cubes being processed from the given database. This is available as a **Flow** variable. Returns an Integer

		Cubename: indicates the name of the cubes being processed from the given database. This is available as an **Flow** variable. Returns a String. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main, Iterate; **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The output schema is fixed and read-only. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

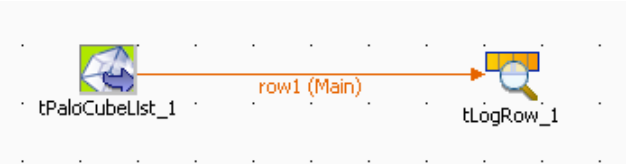
Discovering the read-only output schema of tPaloCubeList

The below table presents information related to the read-only schema of the **tPaloCubeList** component.

Column	Type	Description
Cube_id	int	Internal id of the cube.
Cube_name	string	Name of the cube.
Cube_dimensions	int	Number of dimensions inside the cube.
Cube_cells	long	Number of calculated cells inside the cube.
Cube_filled_cells	long	Number of filled cells inside the cube.
Cube_status	int	Status of the cube. It may be: - **0**: unloaded - **1**: loaded - **2**: changed
Cube_type	int	Type of the cube. It may be: - **0**: normal - **1**: system - **2**: attribute - **3**: user info - **4**. gpu type

Scenario: Retrieving detailed cube information from a given database

The Job in this scenario retrieves detailed information of the cubes pertaining to the demo Palo database, *Biker*.



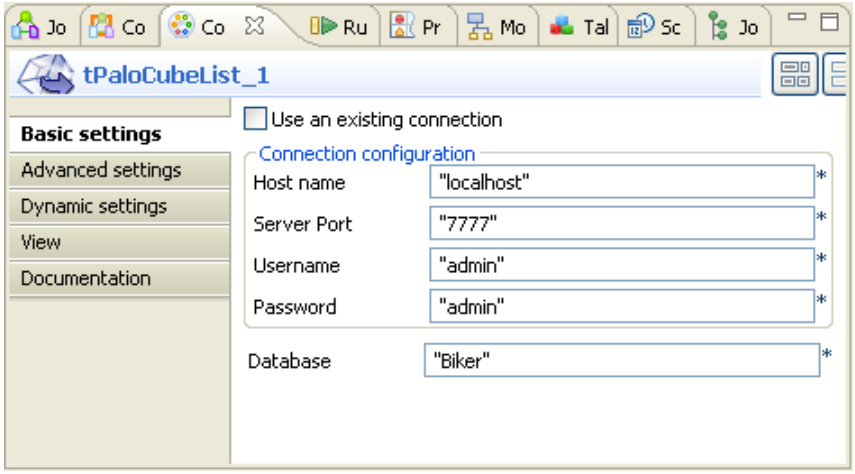
To replicate this scenario, proceed as follows:

Setting up the Job

1. Drop **tPaloCubeList** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tPaloCubeList** to open the contextual menu.
3. From this menu, select **Row** > **Main** to link the two components.

Configuring the tPaloCube component

1. Double-click the **tPaloCube** component to open its **Component** view.

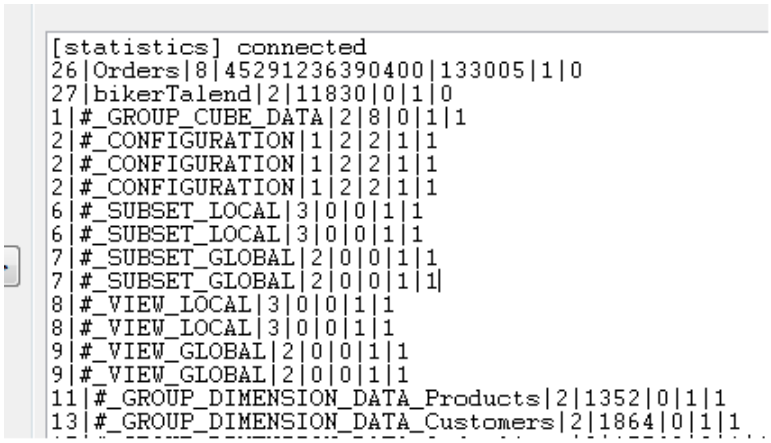


2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
5. In the **Database** field, type in the database name in which you want to create the cube, *Biker* in this example.

Job execution

Press **F6** to run the Job.

The cube details are retrieved from the *Biker* database and are listed in the console of the **Run** view.



For further information about how to intepret the cube details listed in the console, see section *Discovering the read-only output schema of tPaloCubeList*.

tPaloDatabase

tPaloDatabase Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component creates, drops or recreates databases in a given Palo server.	
Purpose	This component manages the databases inside a Palo server.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database on which the given operation should take place.
	Action on database	Select the operation you want to perform on the database of interest: - **Create database**: the database does not exist and will be created. - **Create database if not exists**: the database is created when it does not exist. - **Delete database if exists and create**: the database is deleted if exist and a new one is then created. - **Delete database**: the database is removed from the server
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used in standalone for database management in a Palo server.	

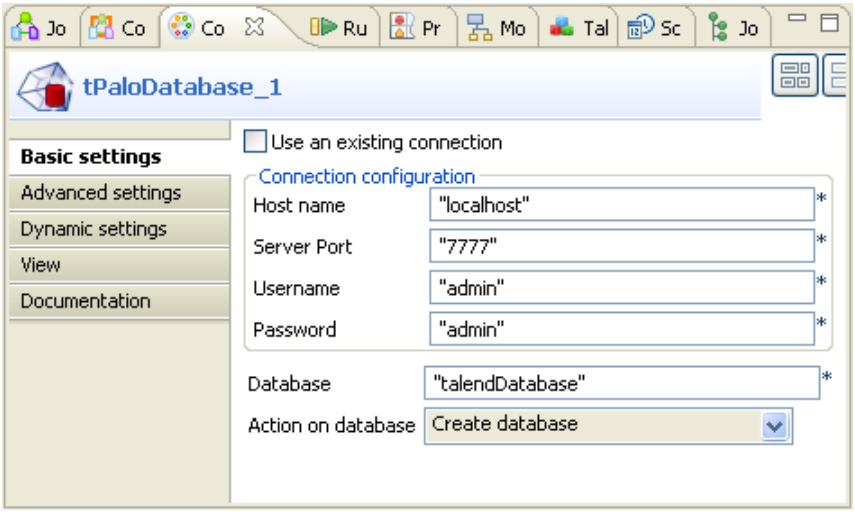
Global Variables		**Databasename**: Indicates the name of the database being processed. This is available as an **After** variable. Returns a String. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Creating a database

The Job in this scenario creates a new database on a given Palo server.

To replicate this scenario, proceed as follows:

1. Drop **tPaloDatabase** from the component **Palette** onto the design workspace.
2. Double-click the **tPaloDatabase** component to open its **Component** view.



3. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
4. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
5. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
6. In the **Database** field, type in the database name in which you want to create the cube, *talenddatabase* in this example.
7. In the **Action on database** field, select the action to be performed. In this scenario, select **Create database** as the database to be created does not exist.
8. Press **F6** to run the Job.

A new database is created on the given Palo server.

tPaloDatabaseList

tPaloDatabaseList Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component retrieves a list of database details from the given Palo server.	
Purpose	This component lists database names, database types, number of cubes, number of dimensions, database status and database id from a given Palo server.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used as a start component. It requires an output component.	
Global Variables		**Number of databases**: Indicates the number of the databases processed. This is available as an **After** variable. Returns a Integer. **Database_id**: Indicates the id of the database being processed. This is available as an **Flow** variable. Returns a Long **Databasename**: Indicates the name of the database processed. This is available as an **After** variable. Returns a String.

		For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The output schema is fixed and read-only. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

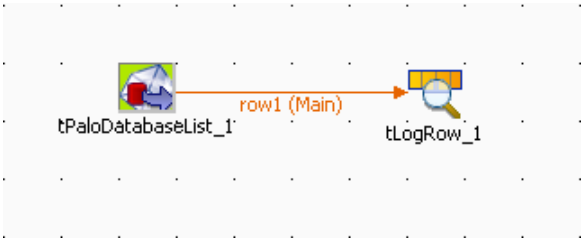
Discovering the read-only output schema of tPaloDatabaseList

The below table presents information related to the read-only output schema of the **tPaloDatabaseList** component.

Database	Type	Description
Database_id	long	Internal ID of the database.
Database_name	string	Name of the database.
Database_dimensions	int	Number of dimensions inside the database.
Database_cubes	int	Number of cubes inside the database.
Database_status	int	Status of the database. - **0** = unloaded - **1** = loaded - **2** = changed
Database_types	int	Type of the database. - **0** =normal - **1** =system - **3** =user info

Scenario: Retrieving detailed database information from a given Palo server

The Job in this scenario retrieves details of all of the databases from a given Palo server.



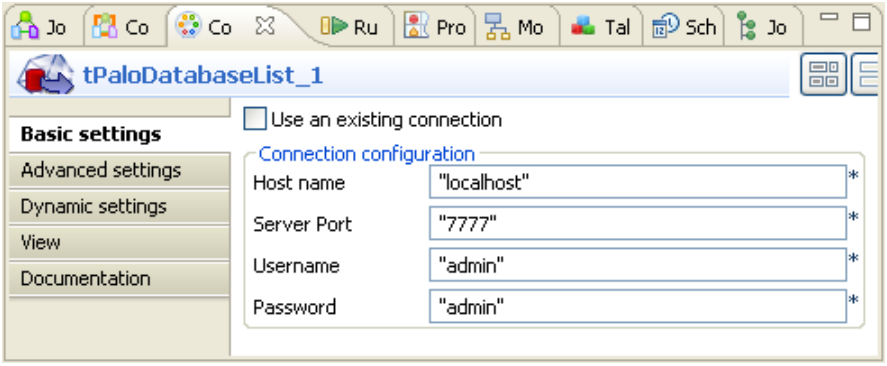
To replicate this scenario, proceed as follows:

Setting up the Job

1. Drop **tPaloDatabaseList** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tPaloDatabaseList** to open the contextual menu.
3. From this menu, select **Row** > **Main** to link the two components.

Configuring the tPaloDatabaseList component

1. Double-click the **tPaloDatabaseList** component to open its **Component** view.



2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.

Job execution

Press **F6** to run the Job.

Details of all of the databases in the Palo server are retrieved and listed in the console of the **Run** view.

```
0|System|7|6|1|1
1|Demo|23|23|1|0
6|Biker|27|28|1|0
9|tx_g|17|17|1|0
25|DBHSE24|13|12|1|0
31|elmCreate|13|12|1|0
32|tPaloOutputMulit|15|15|1|0
```

For further information about the output schema, see section *Discovering the read-only output schema of tPaloDatabaseList*.

tPaloDimension

tPaloDimension Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component creates, drops or recreates dimensions with or without dimension elements inside a Palo database.	
Purpose	This component manages Palo dimensions, even elements inside a database	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database in which the dimensions are managed.
	Dimension	Type in the name of the dimension on which the given operation should take place.
	Action on dimension	Select the operation you want to perform on the dimension of interest: - **None**: no action is taken on this dimension. - **Create dimension**: the dimension does not exist and will be created. - **Create dimension if not exists**: this dimension is created only when it does not exist.

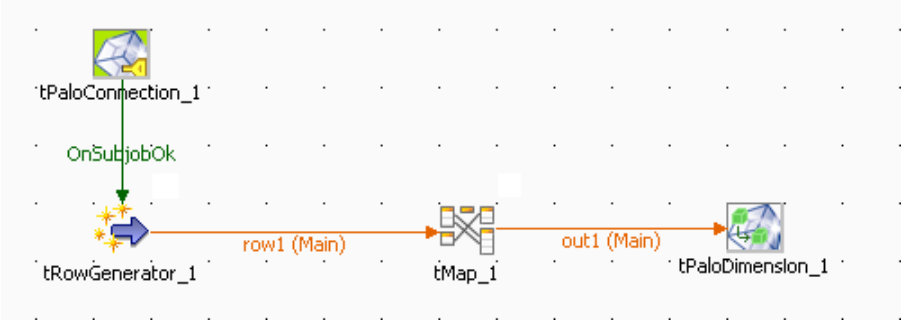
		- **Delete dimension if exists and create**: this dimension is deleted if exist and then a new one will be created. - **Delete dimension**: this dimension is removed from the database.
	Create dimension elements	Select this check box to activate the dimension management fields and create dimension elements along with the creation of this dimension.
The below fields are available only when the **Create dimension elements** check box is selected	*Dimension type* Available only when the action on dimension is **None**.	Select the type of the dimension to be created. The type may be: - **Normal** - **User info** - **System** - **Attribute**
	Commit size	Type in the number of elements which will be created before saving them inside the dimension.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Consolidation type - None With this option, you activate the corresponding parameter fields to be completed.	Select this check box to move directly the incoming elements into the given dimension. With this option, you will not define any consolidations or hierarchy.
		Input Column: select a column from the drop-down list. The columns in the drop-down list are those you defined for the schema. The values from this selected column would be taken to process dimension elements.
		Element type: Select the type of elements. It may be: - **Numeric** - **Text**
		Creation mode: Select creation mode for elements to be processed. This mode may be: - **Add**: add simply an element to the dimension. - **Force add**: force the creation of this element. If exist this element will be recreated. - **Update**: updates this element if it exists. - **Add or Update**: if this element does not exist, it will be created otherwise it will be updated. This is the default option. - **Delete**: delete this element from the dimension
	Consolidation type - Normal With this option, you activate the corresponding parameter fields to be completed.	Select this check box to create elements and consolidate them inside the given dimension. This consolidation structures the created elements in different levels.
		Input Column: select a column from the drop-down list. The columns in the drop-down list are those you defined for the schema. The values from this selected column would be taken to process dimension elements.

		Element type: Select the type of elements. It may be: - **Numeric** - **Text**
		Creation mode: Select creation mode for elements to be created. This mode may be - **Add**: add simply an element to the dimension. - **Force add**: force the creation of this element. If the element exists, it will be recreated. - **Update**: updates this element if it exists. - **Add or Update**: if this element does not exist, it will be created, otherwise it will be updated. This is the default option.
	Consolidation type - Self-referenced With this option, you activate the corresponding parameter fields to be completed.	Select this check box to create elements and structure them based on a parent-child relationship. The input stream is responsible for the grouping of the consolidation.
	Element's type	Select the type of elements. It may be: - **Numeric** - **Text**
	Creation mode	Select creation mode for elements to be created. This mode may be - **Add**: add simply an element to the dimension. - **Force add**: force the creation of this element. If exist this element will be recreated. - **Update**: update this element if it exists. - **Add or Update**: if this element does not exist, it will be created otherwise it will be updated. This is the default option.
		Input Column: select a column from the drop-down list. The columns in the drop-down list are those you defined for the schema. The values from this selected column would be taken to process dimension elements.
		Hierarchy Element: select the type and the relationship of this input column in the consolidation. - **Parent**: set the input value as parent element. - **Child**: relate the input value to the parent value and build the consolidation. - **Factor**: define the factor for this consolidation.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used in standalone or as end component of a process.	
Global Variables		**Dimensionname**: Indicates the name of the dimension processed. This is available as an **After** variable. Returns a String. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another):

		Trigger: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Deletion of dimension elements is only possible with the consolidation type **None**. Only consolidation type **Self-Referenced** allows the placing of an factor on this consolidation. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Creating a dimension with elements

The Job in this scenario creates a date dimension with simple element hierarchy composed of three levels: *Year, Month, Date*.



To replicate this scenario, proceed as follows:

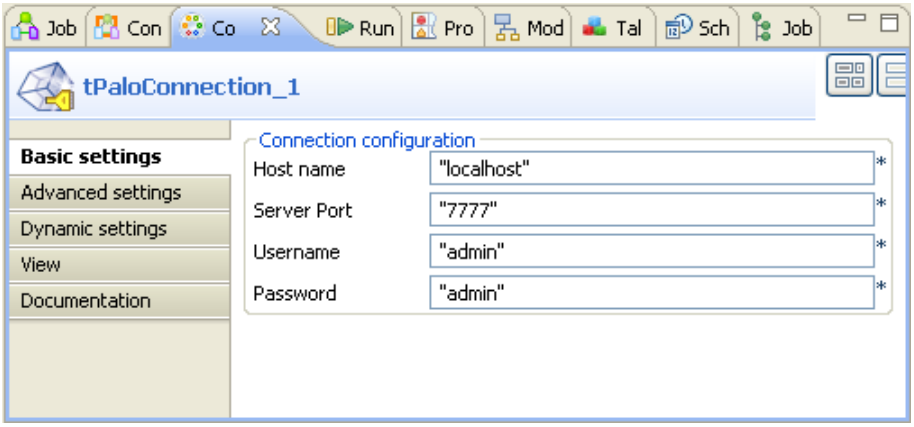
Setting up the Job

1. Drop **tPaloConnection**, **tRowGenerator**, **tMap**, **tPaloDimension** from the component **Palette** onto the design workspace.
2. Right-click **tPaloConnection** to open the contextual menu and select **Trigger** > **On Subjob Ok** to link it to **tRowGenerator**.
3. Right-click **tRowGenerator** to open the contextual menu and select **Row** > **Main** to link it to **tMap**.

   tRowGenerator is used to generate rows at random in order to simplify this process. In the real case, you can use one of the other input components to load your actual data.
4. Right-click **tMap** to open the contextual menu and select **Row** > **New output** to link to **tPaloDimension**, then name it as *out1* in the dialog box that pops up.

Setting up the DB connection

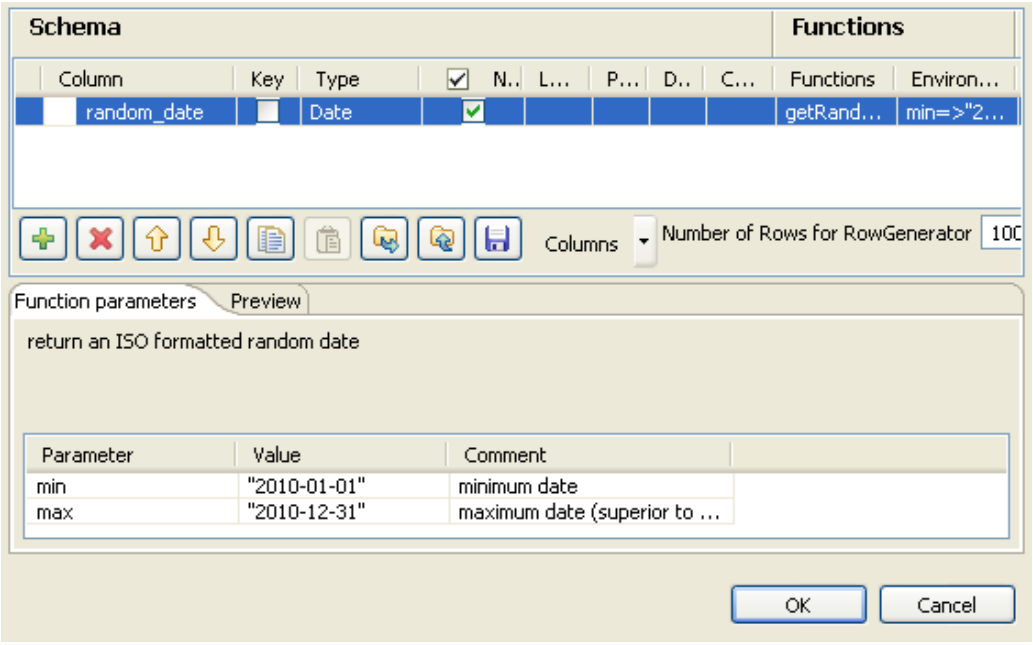
1. Double-click the **tPaloConnection** component to open its **Component** view.



2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.

Configuring the input component

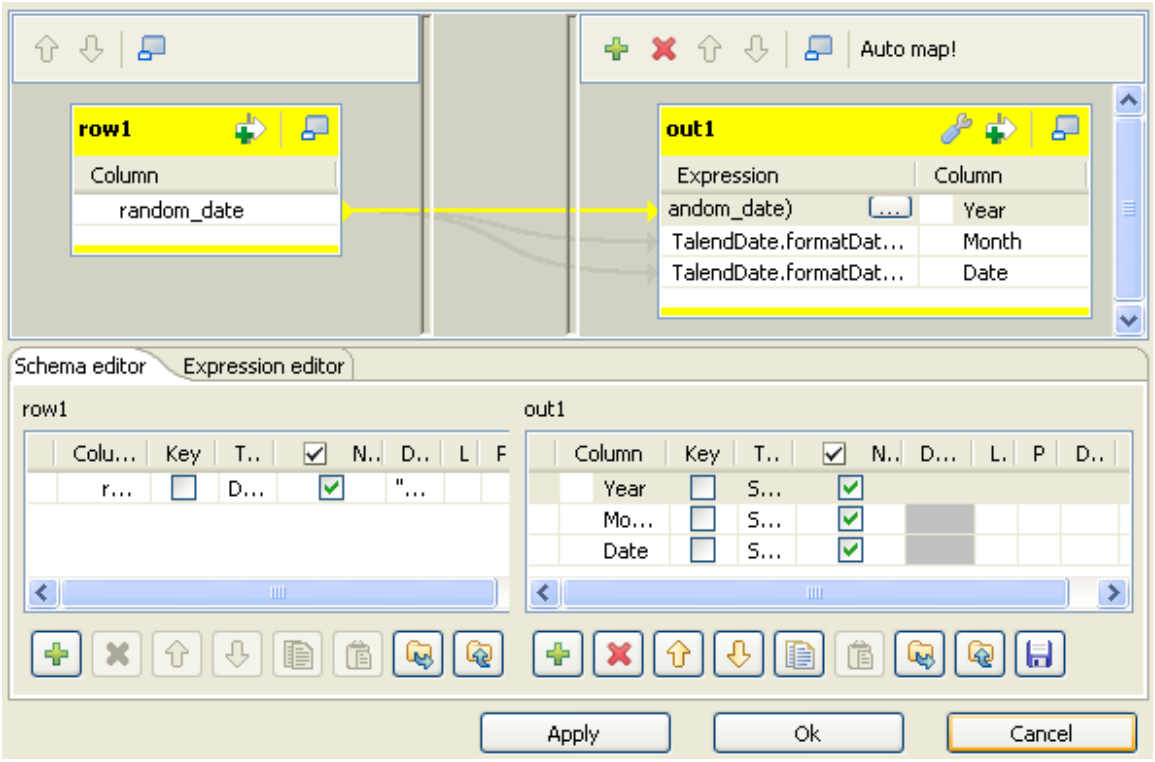
1. Double-click **tRowGenerator** to open its editor.



2. On the upper part of the editor, click the plus button to add one column and rename it as *random_date* in the **Column** column.
3. In the newly added row, select *Date* in the **Type** column and **getRandomDate** in the **Functions** column.
4. In the **Function parameters** view on the lower part of this editor, type in the new minimum date and maximum date values in the **Value** column. In this example, the minimum is *2010-01-01*, the maximum is *2010-12-31*.
5. Click **OK** to validate your modifications and close the editor.
6. On the dialog box that pops up, click **OK** to propagate your changes.

Configuration in the tMap editor

1. Double-click **tMap** to open its editor.



2. On the **Schema editor** view on the lower part of the **tMap** editor, under the **out1** table, click the plus button to add three rows.

3. In the **Column** column of the **out1** table, type in the new names for the three newly added rows. They are *Year*, *Month*, and *Date*. These rows are then added automatically into the **out1** table on the upper part of the **tMap** editor.

4. In the **out1** table on the upper part of the **tMap** editor, click the **Expression** column in the *Year* row to locate the cursor.

5. Press **Ctrl+space** to open the drop-down variable list.

6. Double-click **TalendDate.formatDate** to select it from the list. The expression to get the date displays in the *Year* row under the **Expression** column. The expression is `TalendDate.formatDate("yyyy-MM-dd HH:mm:ss",myDate)`.

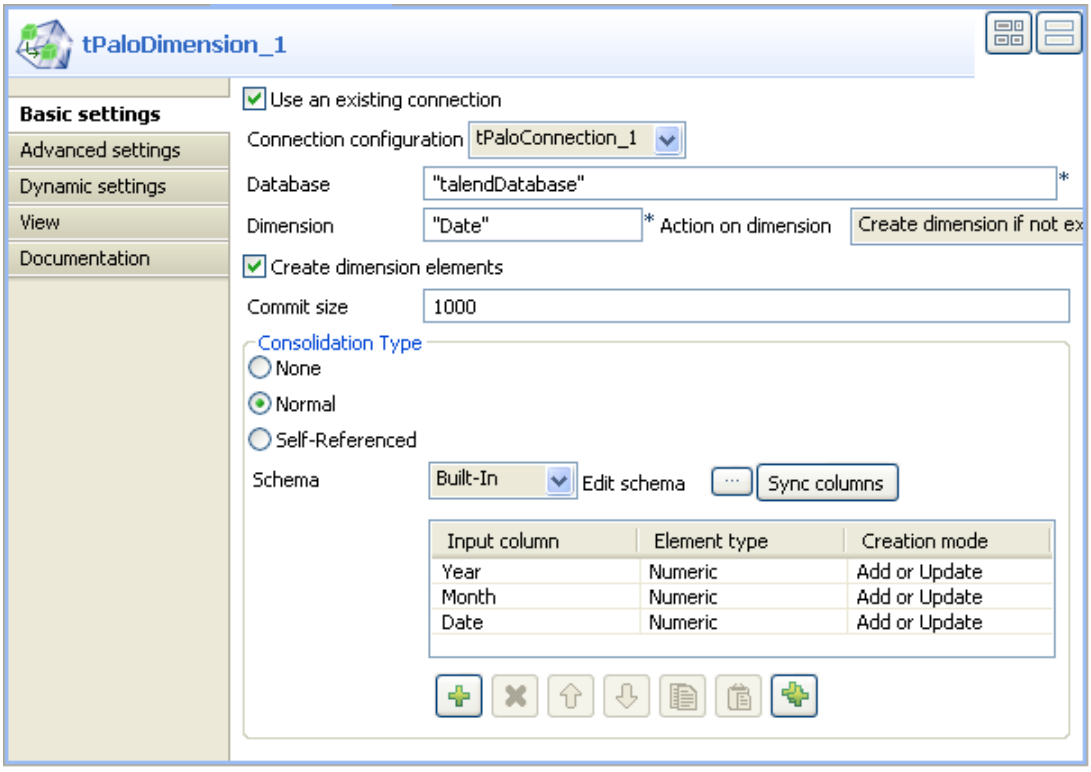
7. Replace the default expression with `TalendDate.formatDate("yyyy",row1.random_date)` .

8. Do the same for the *Month* row and the *Date* row to add this default expression and to replace it with `TalendDate.formatDate("MM",row1.random_date)` for the *Month* row and with `TalendDate.formatDate("dd-MM-yyyy", row1.random_date)` for the *Date* row.

9. Click **OK** to validate this modification and accept the propagation by clicking **OK** in the dialog box that pops up.

Configuring the tPaloDimension component

1. On the workspace, double-click **tPaloDimension** to open its **Component** view.



2. Select the **Use an existing connection** check box. Then **tPaloConnection_1** displays automatically in the **Connection configuration** field.

3. In the **Database** field, type in the database in which the new dimension is created, *talendDatabase* for this scenario.

4. In the **Dimension** field, type in the name you want to use for the dimension to be created, for example, *Date*.

5. In the **Action on dimension** field, select the action to be performed. In this scenario, select **Create dimension if not exist**.

6. Select the **Create dimension elements** check box.

7. In the **Consolidation Type** area, select the **Normal** check box.

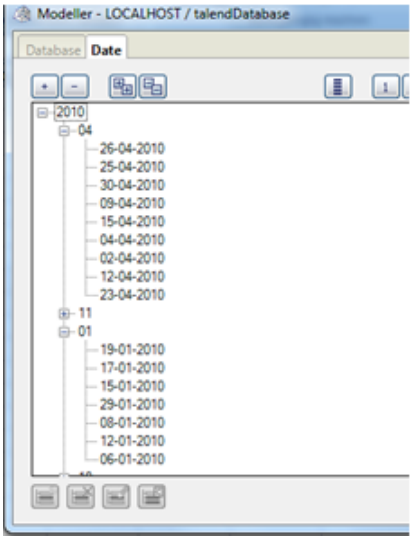
8. Under the element hierarchy table in the **Consolidation Type** area, click the plus button to add three rows into the table.

9. In the **Input column** column of the element hierarchy table, select *Year* from the drop-down list for the first row, *Month* for the second and *Date* for the third. This determinates levels of elements from different columns of the input schema.

Job execution

Press **F6** to run the Job.

A new dimension is then created in your Palo database *talendDatabase*.


Modeller - LOCALHOST / talendDatabase
Database
Date
2010
04
26-04-2010
25-04-2010
30-04-2010
09-04-2010
15-04-2010
04-04-2010
02-04-2010
12-04-2010
23-04-2010
11
01
19-01-2010
17-01-2010
15-01-2010
29-01-2010
08-01-2010
12-01-2010
06-01-2010

tPaloDimensionList

tPaloDimensionList Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component retrieves a list of dimension details from the given Palo database.	
Purpose	This component lists dimension names, dimension types, number of dimension elements, maximum dimension indent, maximum dimension depth, maximum dimension level, dimension id from a given Palo server.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	The name of the database where the dimensions of interest reside.
	Retrieve cube dimensions	Select this check box to retrieve dimension information from an existing cube.
	Cube Available when you select the **Retrieve cube dimensions** check box.	Type in the name of the cube from which dimension information is retrieved.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.

		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used in standalone or as start component of a process.	
Global Variables		**Dimension name**: Indicates the name of the dimension being processed. This is available as an **Flow** variable. Returns a String. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The output schema is fixed and read-only. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Discovering the read-only output schema of tPaloDimensionList

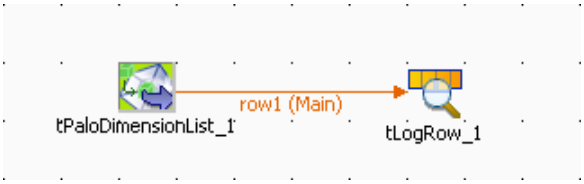
The below table presents information related to the read-only output schema of the **tPaloDimensionList** component.

Database	Type	Description
Dimension_id	long	Internal ID of the dimension.
Dimension_name	string	Name of the dimension.
Dimension_attribute_cube	string	Name of the cube of attributes.
Dimension_rights_cube	string	Name of the cube of rights.
Dimension_elements	int	Number of the dimension elements
Dimension_max_level	int	Maximum level of the dimension
Dimension_max_indent	int	Maximum indent of the dimension
Dimension_max_depth	int	Maximum depth of the dimension
Dimension_type	int	Type of the dimension. - **0** =normal - **1** =system - **2** =attribute

Database	Type	Description
		- **3** =user info

Scenario: Retrieving detailed dimension information from a given database

The Job in this scenario retrieves details of all of the dimensions from a given database.



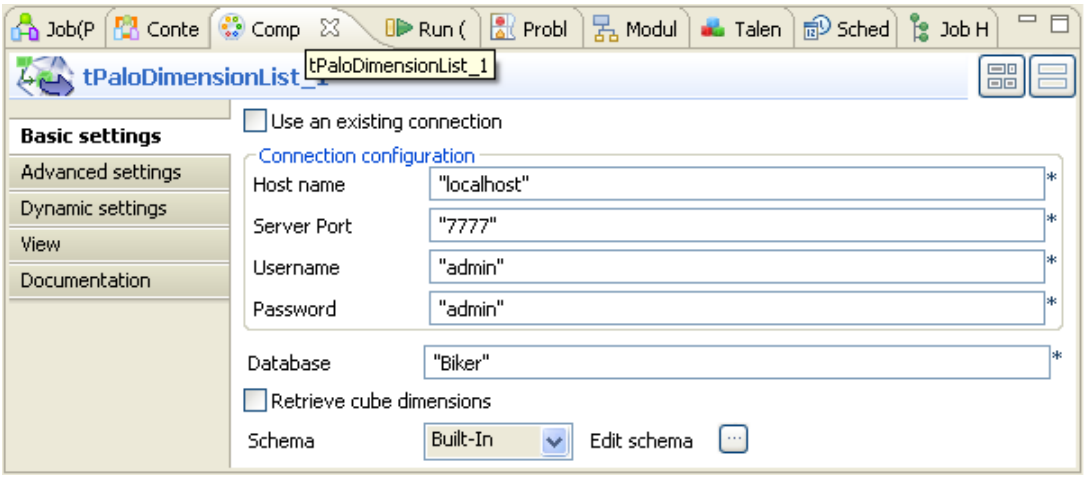
To replicate this scenario, proceed as follows:

Setting up the Job

1. Drop **tPaloDimensionList** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tPaloDimensionList** to open the contextual menu.
3. From this menu, select **Row** > **Main** to link the two components.

Configuring the tPaloDimensionList component

1. Double-click the **tPaloDimensionList** component to open its **Component** view.



2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
5. In the **Database** field, type in the database name where the dimensions of interest reside, *Biker* in this example.

Job execution

Press **F6** to run the Job.

Details of all the dimensions in the *Biker* database are retrieved and listed in the console of the **Run** view.

For further information about the output schema, see section *Discovering the read-only output schema of tPaloDimensionList*.

tPaloInputMulti

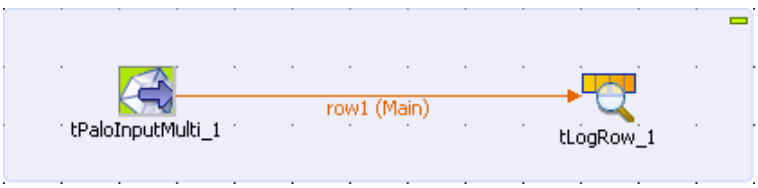
tPaloInputMulti Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component retrieves data (elements as well as values) from a Palo cube.	
Purpose	This component retrieves the stored or calculated values in combination with the element records out of a cube.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database where the elements of interest reside.
	Cube	Type in the name of the cube where the dimension elements to be retrieved are stored.
	Cube type	Select the cube type from the drop-down list for the cube of concern. This type may be: - **Normal** - **Attribut** - **System** - **User Info**
	Commit size	Type in the row count of each batch to be retrieved.

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. The **MEASURE** column and the **TEXT** column are read-only, but you can add other columns aside.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Cube Query	Complete this table with the query you want to use to retrieve data. The columns to be filled are:
		Column: the schema columns are added automatically to this column once defined in the schema editor. The schema columns are used to stored the retrieved dimension elements.
		Dimensions: type in each of the dimension names of the cube from which you want to retrieve dimension elements. *The dimension order listed in this column must be consistent with the order given in the cube that stores these dimensions.*
		Elements: type in the dimension elements from which data is retrieved. If several elements are needed from one single dimension, separate them with a coma.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component requires an output component.	
Connections		Outgoing links (from one component to another): **Row**: Main **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	According to the architecture of OLAP-Systems only one single value (text or numeric) could be retrieved from the cube. The **MEASURE** column and the **TEXT** column are fixed and read-only. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Retrieving dimension elements from a given cube

The Job in this scenario retrieves several dimension elements from a demo Palo cube *Sales*.



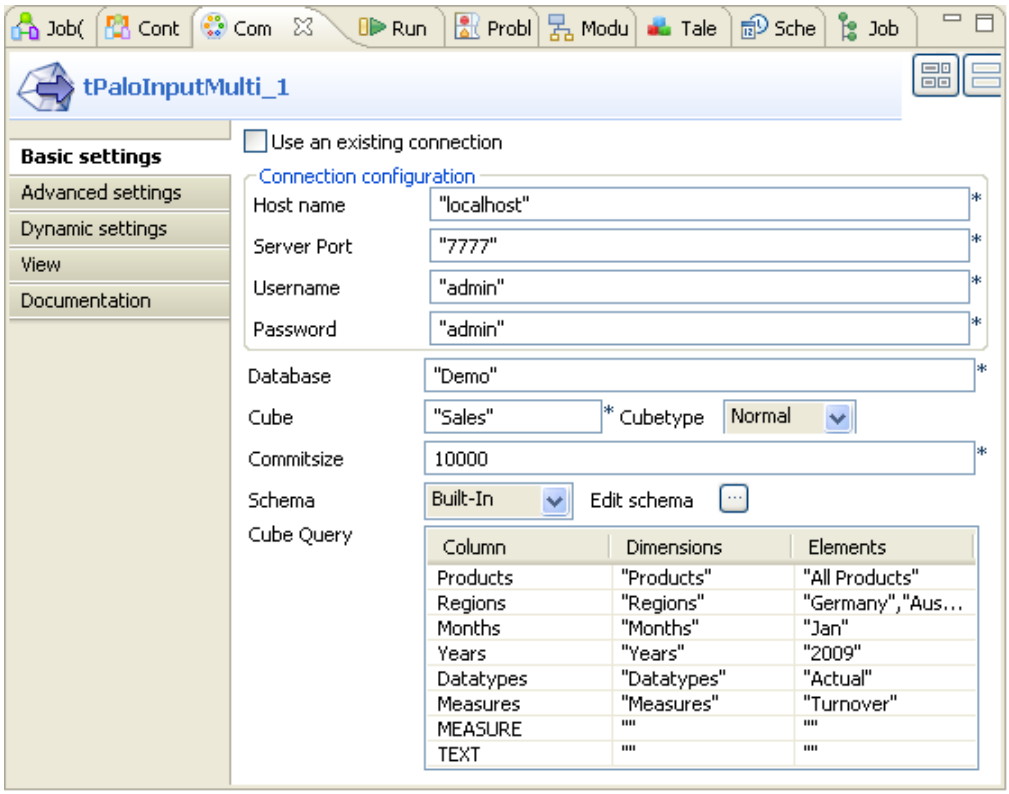
To replicate this scenario, proceed as follows:

Setting up the Job

1. Drop **tPaloInputMulti** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tPaloInputMulti** to open its contextual menu.
3. In the menu, select **Row** > **Main** to connect **tPaloInputMulti** to **tLogRow** with a **row** link.

Setting up the DB connection

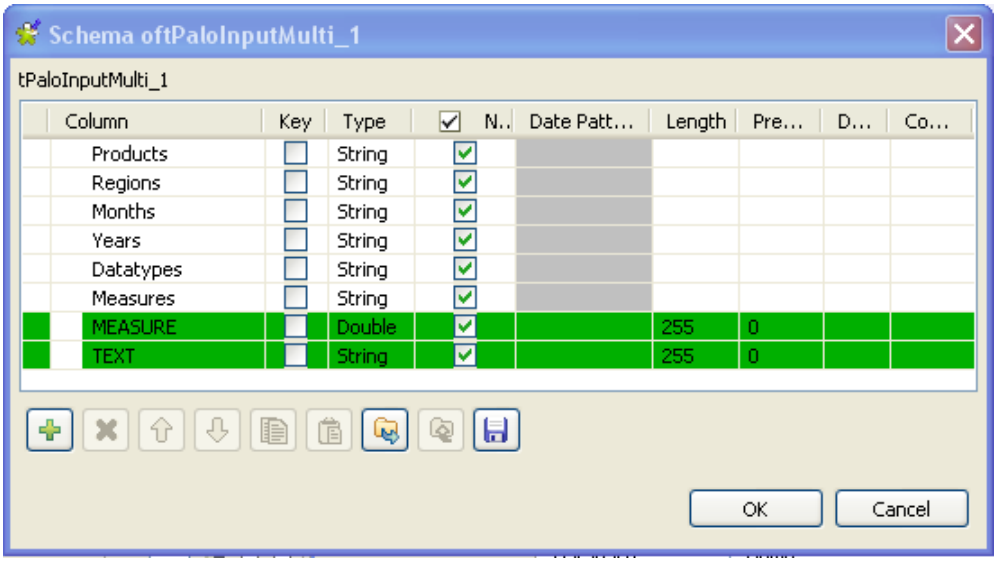
1. Double-click the **tPaloInputMulti** component to open its **Component** view.



2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.

Configuring the Cube Query

1. In the **Database** field, type in the database name in which the cube to be used is stored.
2. In the **Cube** field, type in the cube name in which the dimensions of interests are stored. In this scenario, it is one of the demo cubes *Sales*.
3. In the **Cube type** field, select the **Normal** type from the drop-down list for the cube to be created, meaning this cube will be normal and default.
4. Next to the **Edit schema** field, click the three-dot button to open the schema editor.



5. In the schema editor, click the plus button to add the rows of the schema to be edited. In this example, add rows corresponding to all of the dimensions stored in the *Sales* cube: *Products*, *Regions*, *Months*, *Years*, *Datatypes*, *Measures*. Type in them in the order given in this cube.

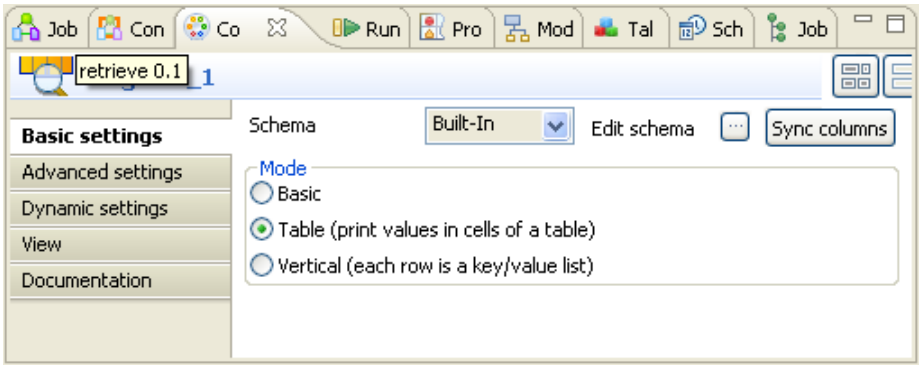
6. Click **OK** to validate this editing and accept the propagation of this change to the next component. Then these columns are added automatically into the **Column** column of the **Cube query** table in the **Component** view. If the order is not consistent with the one in the *Sales* cube, adapt it using the up and down arrows under the schema table.

7. In the **Dimensions** column of the **Cube query** table, type in each of the dimension names stored in the *Sales* cube regarding to each row in the **Column** column. In the *Sales* cube, the dimension names are: *Products*, *Regions*, *Months*, *Years*, *Datatypes*, *Measures*.

8. In the **Elements** columns of the **Cube query** table, type in the dimension elements you want to retrieve regarding to the dimensions they belong to. In this example, the elements to be retrieved are *All Products*, *Germany* and *Austria* (Belonging to the same dimension **Regions**, these two elements are entered in the same row and separated with a coma.), *Jan, 2009*, *Actual*, *Turnover*.

Job execution

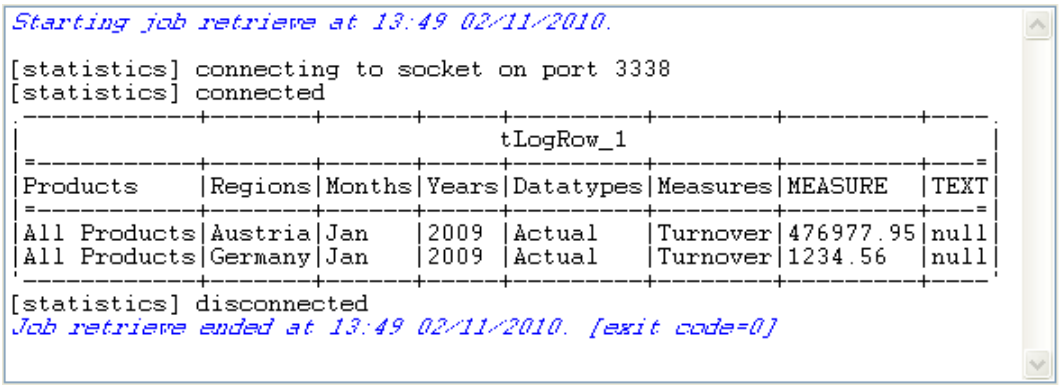
1. Click **tLogRow** to open its **Component** view.



2. In the **Mode** area, select the **Table (print values in cells of a table)** check box to display the execution result in a table.

3. Press **F6** to run the Job.

The dimension elements and the corresponding Measure values display in the **Run** console.



tPaloOutput

tPaloOutput Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component writes one row of data (elements as well as values) into a Palo cube.	
Purpose	This component takes the input stream and writes it to a given Palo cube.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database where the cube of interest resides.
	Cube	Type in the name of the cube in which the incoming data is written.
	Commit size	Type in the row count of each batch to be written into the cube.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Column as Measure	Select the column from the input stream which holds the Measure or Text values.
	Create element if not exist	Select this check box to create the element being processed if it does not exist originally.

	Save cube at process end	Select this check box to save the cube you have written the data in at the end of this process.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component requires an input component.	
Global variable		**Number of lines**: Indicates the number of the lines processed. This is available as an **After** variable. Returns a Integer.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: Run if Incoming links (from one component to another): **Row**: Main; Reject For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	This component is able to write only one row of data into a cube. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related topic, see section *Scenario 1: Writing data into a given cube*.

tPaloOutputMulti

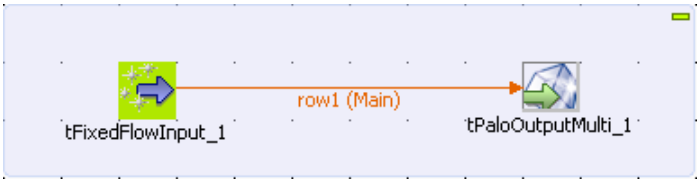
tPaloOutputMulti Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component writes data (elements as well as values) into a Palo cube.	
Purpose	This component takes the input stream and writes it to a given Palo cube.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database where the cube of interest resides.
	Cube	Type in the name of the cube in which the incoming data is written.
	Cube type	Select the cube type from the drop-down list for the cube of concern. This type may be: - **Normal** - **Attribut** - **System** - **User Info**
	Commit size	Type in the row count of each batch to be written into the cube.

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Measure value	Select the column from the input stream which holds the Measure or Text values.
	Splash mode	Select the splash mode used to write data into a consolidated element. The mode may be: - **Add**: it writes values to the underlying elements. - **Default**: it uses the default splash mode. - **Set**: it simply sets or replaces the current value and make the distribution based on the other values. - **Disable**: it applies no splashing. For further information about the Palo splash modes, see Palo's user guide.
	Add values	Select this check box to add new values to the current values for a sum. Otherwise these new values will overwrite the current ones.
	Use eventprocessor	Select this checkbox to call the supervision server.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component requires an input component.	
Connections		Outgoing links (from one component to another): **Row**: Main **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Numeric measures are only be accepted as Double or String type. When the string type is used, write the value to be processed between quotation marks. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario 1: Writing data into a given cube

The Job in this scenario writes new values in the *Sales* cube given as demo in the *Demo* database installed with Palo.



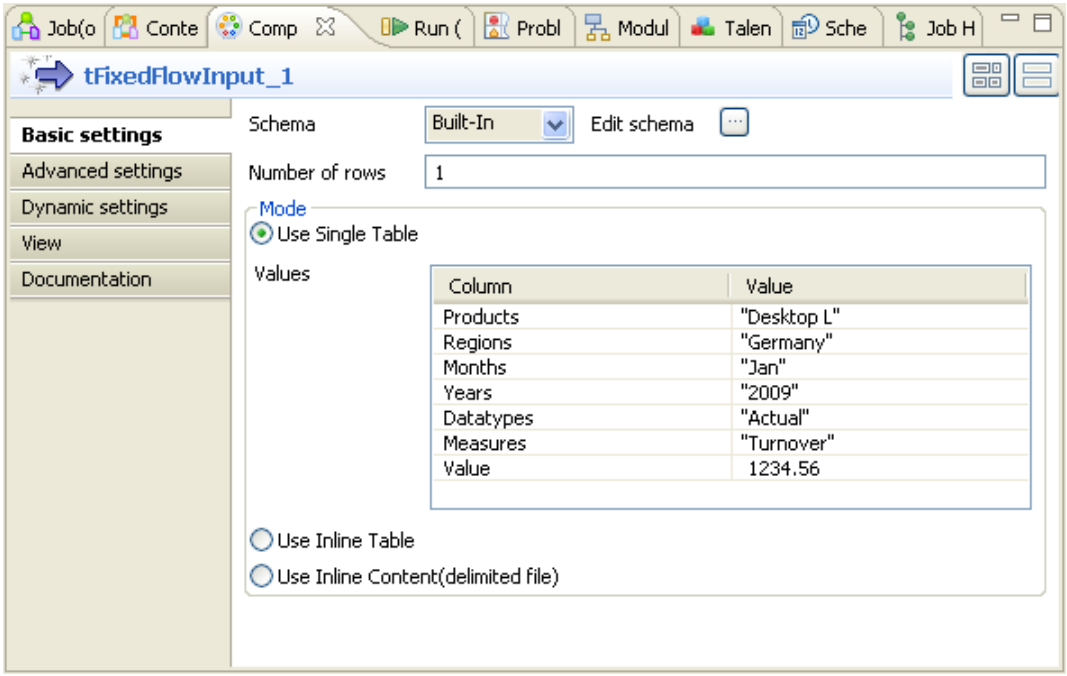
To replicate this scenario, proceed as follows:

Setting up the Job

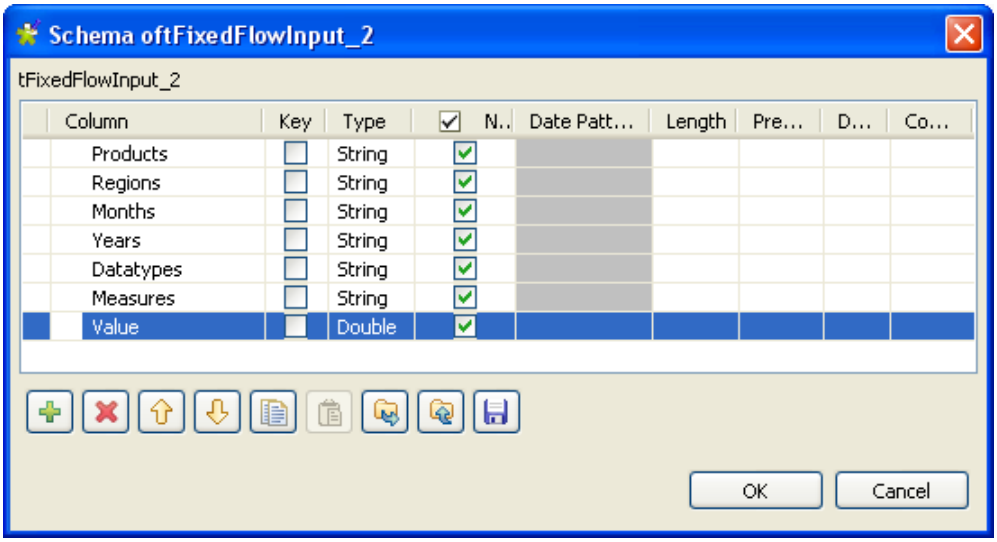
1. Drop **tFixedFlowInput** and **tPaloOutputMulti** from the component **Palette** onto the design workspace.
2. Right-click **tFixedFlowInput** to open its contextual menu.
3. In this menu, select **Row** > **Main** to connect this component to **tPaloOutputMulti**.

Configuring the input component

1. Double-click the **tFixedFlowInput** component to open its **Component** view.



2. Click the three-dot button to open the schema editor.



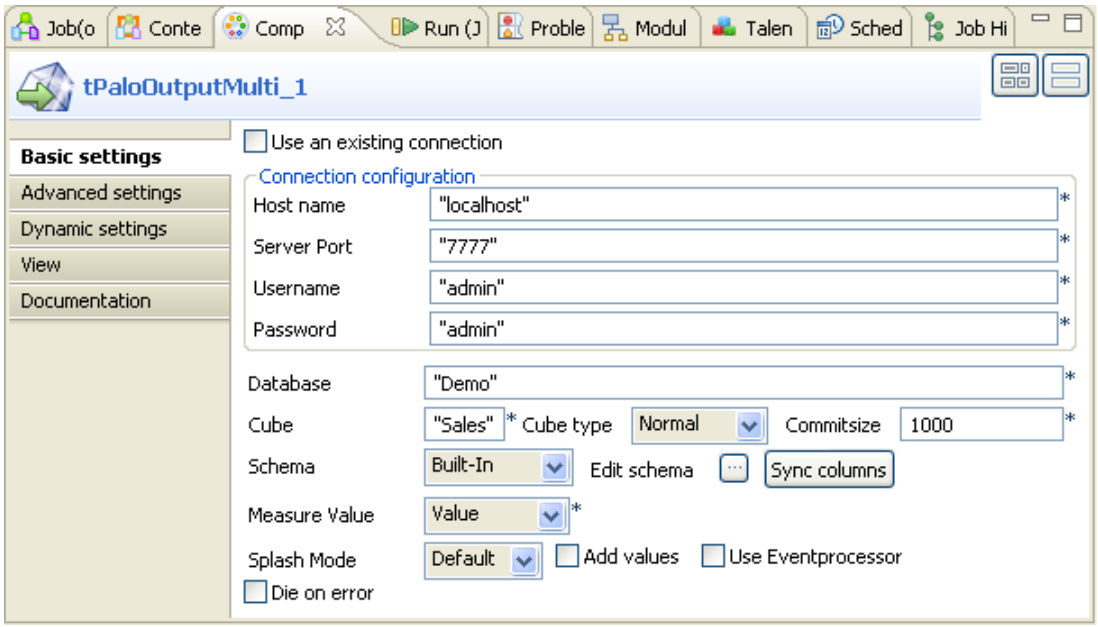
3. In the schema editor, click the plus button to add 7 rows and rename them respectively as *Products*, *Regions*, *Months*, *Years*, *Datatypes*, *Measures* and *Values*. The order of these rows must be consistent with that of the corresponding dimensions in the *Sales* cube and the type of the *Value* column where the measure value resides is set to double/Double.

4. Click **OK** to validate the editing and accept the propagation prompted by the dialog box that pops up. Then the schema column labels display automatically in the **Value** table under the **Use single table** check box, in the **Mode** area.

5. In the **Value** table, type in values for each row in the **Value** column. In this example, these values are: *Desktop L, Germany, Jan, 2009, Actual, Turnover, 1234.56*.

Configuring the output component

1. Double-click **tPaloOutputMulti** to open its **Component** view.



2. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.

3. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.

4. In the **Database** field, type in the database name in which you want to create the cube, *Demo* in this example.

5. In the **Cube** field, type in the name of the cube you want to write data in, for example, *Sales*.

6. In the **Cube type** field, select the **Normal** type from the drop-down list for the cube to be created, meaning this cube will be normal and default.

7. In the **Measure Value** field, select the Measure element. In this scenario, select **Value**.

Job execution

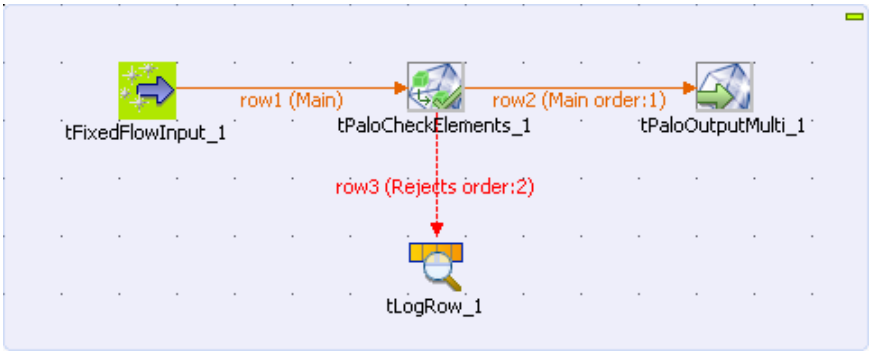
Press **F6** to run the Job.

The inflow data has been written into the *Sales* cube.

	A
1	localhost/Demo
2	Sales
3	Desktop L
4	**Germany**
5	Jan
6	2009
7	Actual
8	Turnover
9	
10	1 234,56
11	

Scenario 2: Rejecting inflow data when the elements to be written do not exist in a given cube

The Job in this scenario tries to write data into the *Sales* cube but as the elements of interest do not exist in this cube, the inflow data is rejected.



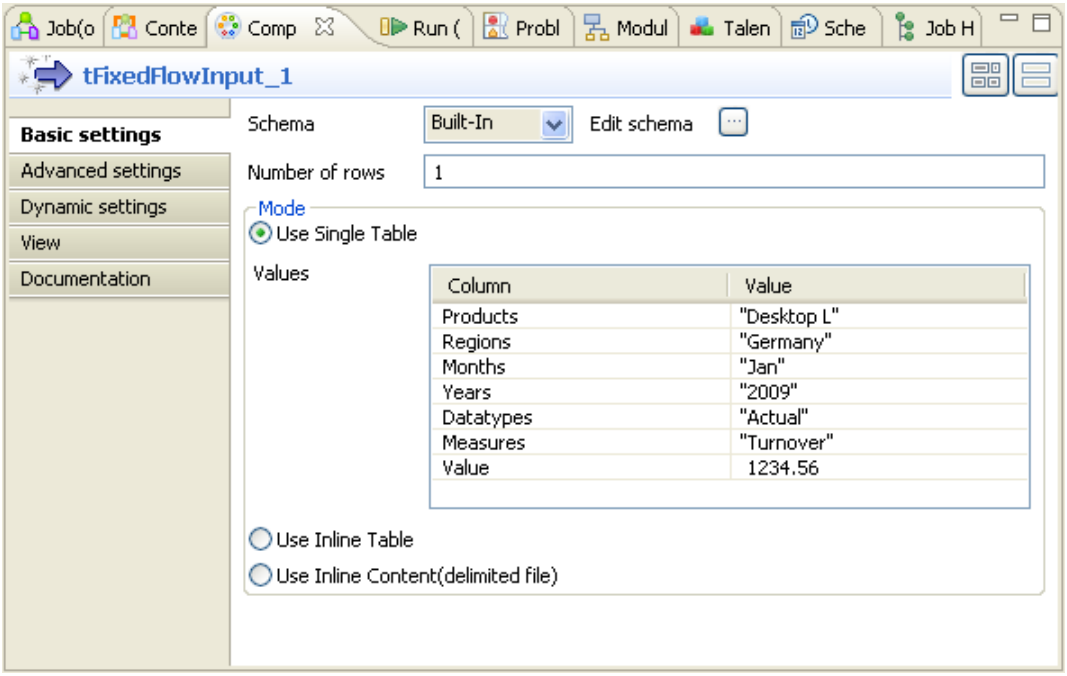
To replicate this scenario, proceed as follows:

Setting up the Job

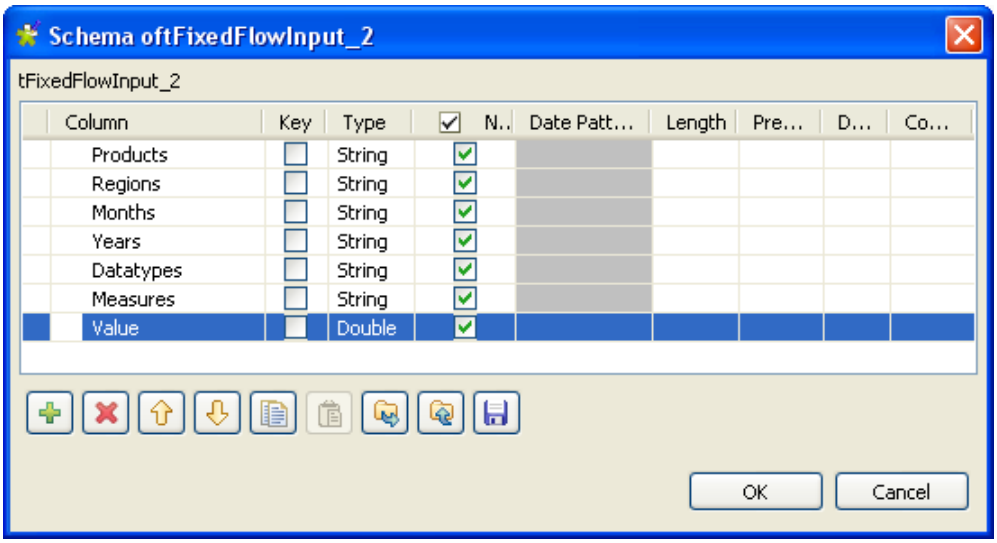
1. Drop **tFixedFlowInput**, **tPaloCheckElements**, **tPaloOutputMulti** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tFixedFlowInput** to open its contextual menu.
3. In this menu, select **Row** > **Main** to connect this component to **tPaloCheckElements**.
4. Do the same to connect **tPaloOutputMulti** using **row** link.
5. Right-click **tPaloCheckElements** to open its contextual menu.
6. In this menu, select **Row** > **Reject** to connect this component to **tLogRow**.

Configuring the input component

1. Double-click the **tFixedFlowInput** component to open its **Component** view.



2. Click the three-dot button to open the schema editor.



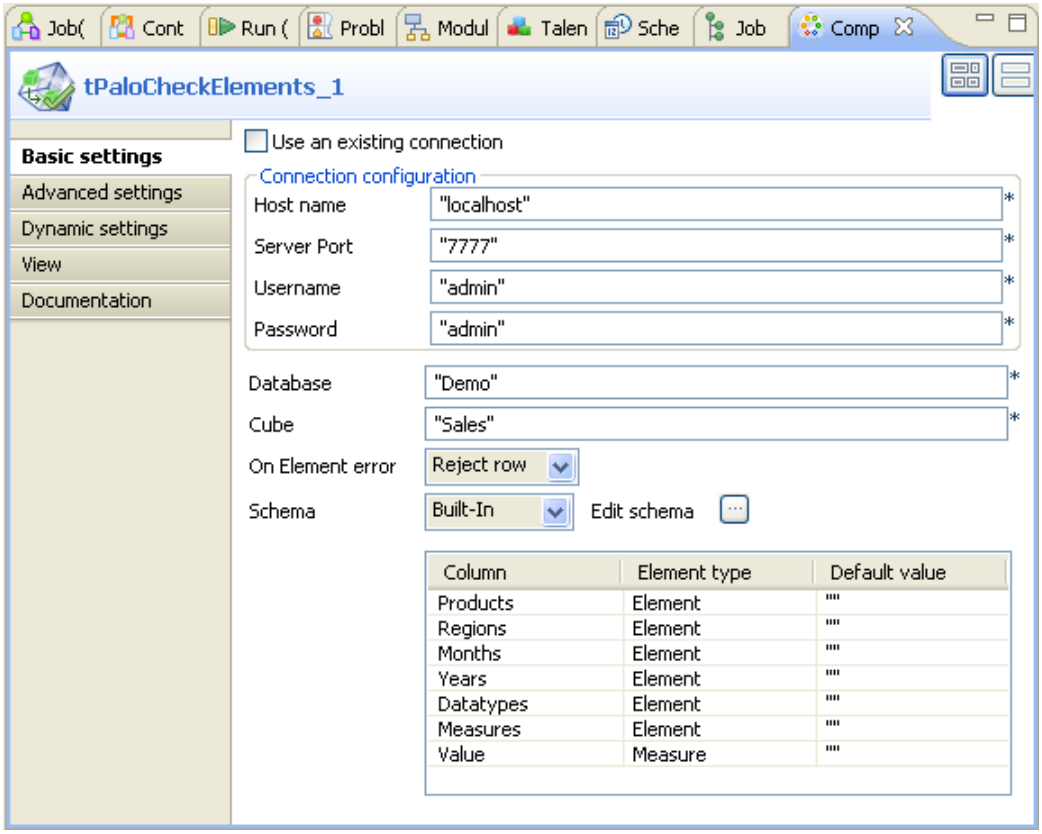
3. In the schema editor, click the plus button to add 7 rows and rename them respectively as *Products*, *Regions*, *Months*, *Years*, *Datatypes*, *Measures* and *Values*. The order of these rows must be consistent with that of the corresponding dimensions in the *Sales* cube and the type of the *Value* column where the measure value resides is set to double/Double.

4. Click **OK** to validate the editing and accept the propagation prompted by the dialog box that pops up. Then the schema column labels display automatically in the **Value** table under the **Use single table** check box, in the **Mode** area.

5. In the **Value** table, type in values for each row in the **Value** column. In this example, these values are: *Smart Products*, *Germany*, *Jan*, *2009*, *Actual*, *Turnover*, *1234.56*. The *Smart Products* element does not exist in the *Sales* cube.

Configuring the tPaloCheckElements component

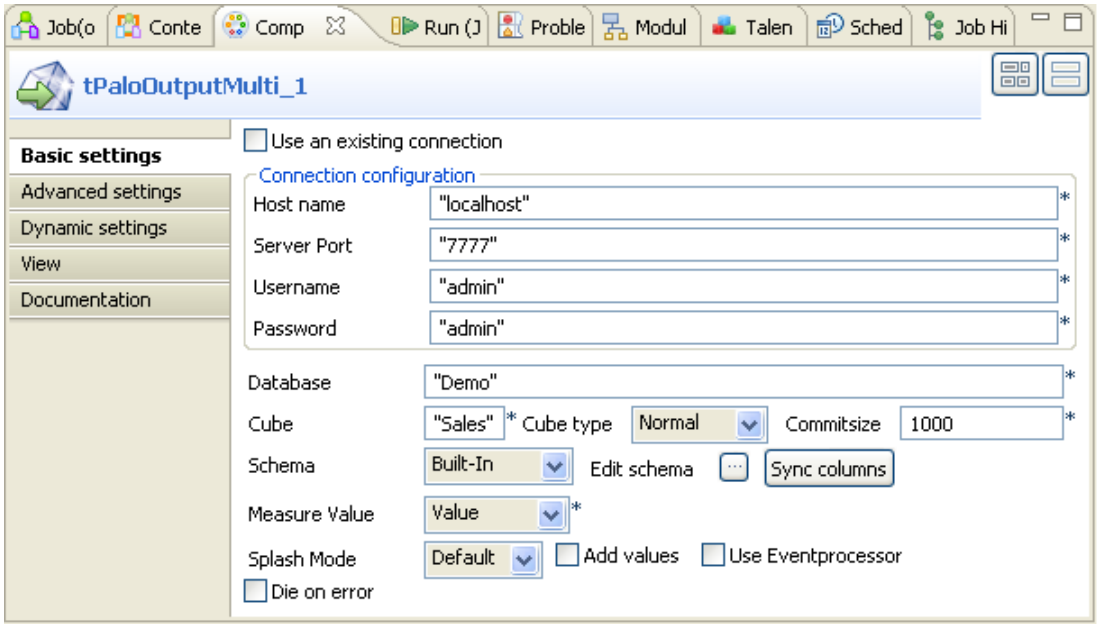
1. Double-click **tPaloCheckElements** to open its **Component** view.



2. In the **Host name** field, type in *localhost*.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
5. In the **Database** field, type in the database name in which you want to create the cube, *Demo* in this example.
6. In the **Cube** field, type in the name of the cube you want to write data in, for example, *Sales*.
7. In the **On Element error** field, select **Reject row** from the drop-down list.
8. In the element table at the bottom of the **Basic settings** view, click the **Element type** column in the *Value* row and select **Measure** from the drop down list.

Configuring the output component

1. Double-click **tPaloOutputMulti** to open its **Component** view.



2. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
3. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
4. In the **Database** field, type in the database name in which you want to create the cube, *Demo* in this example.
5. In the **Cube** field, type in the name of the cube you want to write data in, for example, *Sales*.
6. In the **Cube type** field, select the **Normal** type from the drop-down list for the cube to be created, meaning this cube will be normal and default.
7. In the **Measure Value** field, select the Measure element. In this scenario, select **Value**.

Job execution

Press **F6** to run the Job.

The data to be written is rejected and displayed in the console of the **Run** view. You can read that the error message is *Smart Products*.

```
Starting job JobOutputExistingElements at 14:45 09/11/2010.

[statistics] connecting to socket on port 3407
[statistics] connected
.--------------+-------+------+-----+---------+--------+-------+----------------------.
|                                    tLogRow_1                                         |
|=-------------+-------+------+-----+---------+--------+-------+----------------------=|
|Products      |Regions|Months|Years|Datatypes|Measures|Value  |errorMessage          |
|=-------------+-------+------+-----+---------+--------+-------+----------------------=|
|Smart Products|Germany|Jan   |2009 |Actual   |Turnover|1234.56|Products:Smart Products|
'--------------+-------+------+-----+---------+--------+-------+----------------------'
[statistics] disconnected
Job JobOutputExistingElements ended at 14:45 09/11/2010. [exit code=0]
```

tPaloRule

tPaloRule Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component creates or modifies rules in a given cube.	
Purpose	This component allows you to manage rules in a given cube.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	Type in the name of the database where the dimensions applying the rules of interest reside.
	Cube	Type in the name of the cube whose dimension information is retrieved.
	Cube rules	Complete this table to perform various actions on specific rules. **Definition**: type in the rule to be applied.
		External Id: type in the user-defined external ID.
		Comment: type in comment for this rule.
		Activated: select this check box to activate this rule.
		Action: select the action to be performed from the drop-down list. - **Create:** create this rule.

		- **Delete**: delete this rule. - **Update**: update this rule.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used in standalone for rule creation, deletion or update.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Update or deletion of a rule is available only when this rule has been created with external ID. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

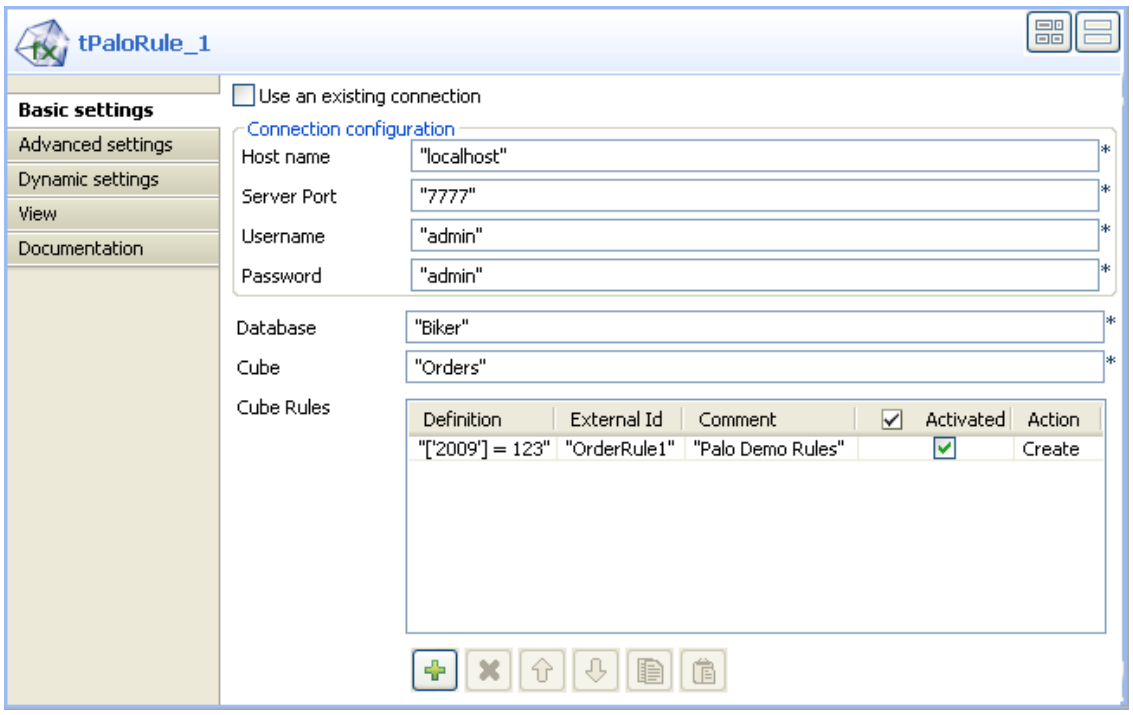
Scenario: Creating a rule in a given cube

The Job in this scenario creates a rule applied on dimensions of a given cube.

To replicate this scenario, proceed as follows:

Setting up the DB connection

1. Drop **tPaloRule** from the component **Palette** onto the design workspace.
2. Double-click the **tPaloRule** component to open its **Component** view.



3. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
4. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
5. In the **Username** field and the **Password** field, type in the authentication information. In this example, both of them are *admin*.
6. In the **Database** field, type in the database name in which the dimensions applying the created rules reside, *Biker* in this example.
7. In the **Cube** field, type in the name of the cube which the dimensions applying the created rules belong to, for example, *Orders*.

Setting the Cube rules

1. Under the **Cube rules** table, click the plus button to add a new row.
2. In the **Cube rules** table, type in `['2009'] = 123` in the **Definition** column, `OrderRule1` in the **External Id** column and `Palo Demo Rules` in the **Comment** column.
3. In the **Activated** column, select the check box.
4. In the **Action** column, select **Create** from the drop-down list.

Job execution

Press **F6** to run the Job.

The new rule has been created and the value of every *2009* element is *123*.

	A
1	localhost/Biker
2	Orders
3	**All Customers**
4	**All Channels**
5	**All Orders**
6	**All Datatypes**
7	Units
8	**All Products**
9	**All Months**
10	2009
11	
12	123
13	

tPaloRuleList

tPaloRuleList Properties

Component family	Business Intelligence/Cube OLAP/Palo	
Function	This component retrieves a list of rule details from the given Palo database.	
Purpose	This component lists all rules, formulas, comments, activation status, external IDs from a given cube.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Connection configuration** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Connection configuration** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
Connection configuration Unavailable when using an existing connection.	*Host Name*	Enter the host name or the IP address of the host server.
	Server Port	Type in the listening port number of the Palo server.
	Username and *Password*	Enter the Palo user authentication data.
	Database	The name of the database where the cube of interest resides.
	Cube	Type in the name of the cube in which you want to retrieve the rule information.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used in standalone or as start component of a process.	
Global Variables		**Number of rules**: Indicates the number of the rules processed. This is available as an **After** variable.

		Returns a Integer. **External ruleID**: Indicates the external IDs of the rules being processed. This is available as a **Flow** variable. Returns a String For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Subjob Ok; On Subjob Error; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The output schema is fixed and read-only. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

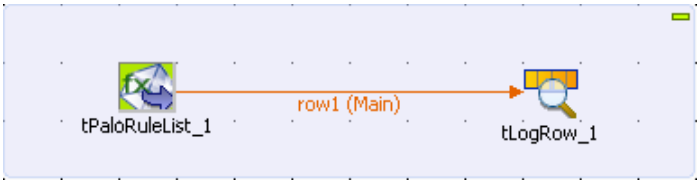
Discovering the read-only output schema of tPaloRuleList

The following table presents information related to the read-only output schema of the **tPaloRuleList** component.

Database	Type	Description
rule_identifier	long	The internal identifier/id for this rule..
rule_definition	string	The formula of this rule. For further information about this formula, see the Palo user guide.
rule_extern_id	string	The user-defined external id.
rule_comment	string	The user-edited comment on this rule.
rule_activated	boolean	Indicates if this rule had been activated or not.

Scenario: Retrieving detailed rule information from a given cube

The Job in this scenario retrieves rule details applied on the dimensions of a given cube.



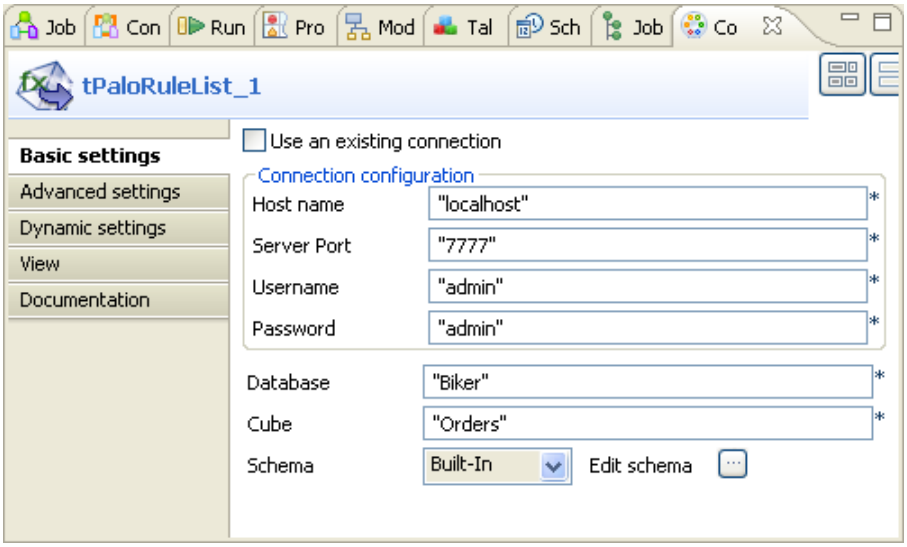
To replicate this scenario, proceed as follows:

Setting up the Job

1. Drop **tPaloRuleList** and **tLogRow** from the component **Palette** onto the design workspace.
2. Right-click **tPaloRuleList** to open the contextual menu.
3. From this menu, select **Row** > **Main** to link the two components.

Configuring the tPaloRuleList component

1. Double-click the **tPaloRuleList** component to open its **Component** view.



2. In the **Host name** field, type in the host name or the IP address of the host server, *localhost* for this example.
3. In the **Server Port** field, type in the listening port number of the Palo server. In this scenario, it is *7777*.
4. In the **Username** and **Password** fields, type in the authentication information. In this example, both of them are *admin*.
5. In the **Database** field, type in the database name where the dimensions applying the rules of interest reside, *Biker* in this example.
6. In the **Cube** field, type in the name of the cube which the rules of interest belong to.

Job execution

Press **F6** to run the Job.

Details of all of the rules in the *Orders* cube are retrieved and listed in the console of the **Run** view.

```
Starting job tpalo at 16:34 12/11/2010.

[statistics] connecting to socket on port 3743
[statistics] connected
.---------------+---------------+--------------+---------------+--------------.
|                                   tLogRow_1                                  |
|=--------------+---------------+--------------+---------------+-------------=|
|rule_identifier|rule_definition|rule_extern_id|rule_comment    |rule_activated|
|=--------------+---------------+--------------+---------------+-------------=|
|1              |['2009'] = 123 |OrderRule1    |Palo Demo Rules|true          |
'---------------+---------------+--------------+---------------+--------------'
[statistics] disconnected
Job tpalo ended at 16:35 12/11/2010. [exit code=0]
```

For further information about the output schema, see section *Discovering the read-only output schema of tPaloRuleList*.

tParAccelSCD

tParAccelSCD Properties

Component family	Databases/ParAccel	
Function	**tParAccelSCD** reflects and tracks changes in a dedicated ParAccel SCD table.	
Purpose	**tParAccelSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Select the relevant driver on the list.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs.

		For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related scenarios, see section *tMysqlSCD*.

tPostgresPlusSCD

tPostgresPlusSCD Properties

Component family	Databases/PostgresPlus Server	
Function	**tPostgresPlusSCD** reflects and tracks changes in a dedicated MSSQL SCD table.	
Purpose	**tPostgresPlusSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs.

		For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related topics, see section *tMysqlSCD*.

tPostgresPlusSCDELT

tPostgresPlusSCDELT Properties

Component family	Databases/Postgresql	
Function	**tPostgresPlusSCDELT** reflects and tracks changes in a dedicated Oracle SCD table.	
Purpose	**tPostgresPlusSCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated PostgresPlus SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	Use an existing connection	Select this check box and click the relevant **tPostgresPlusConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database.
	Schema	Exact name of the schema
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input DB2 SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created.

		Create table if not exists: A table gets created if it does not exist. **Clear table**: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	Source Keys	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Debug mode*	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see section *tMysqlSCD*.

tPostgresqlSCD

tPostgresqlSCD Properties

Component family	Databases/Postgresql Server	
Function	**tPostgresqlSCD** reflects and tracks changes in a dedicated Postrgesql SCD table.	
Purpose	**tPostgresqlSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.

	Use memory saving Mode	Select this check box to maximize system performance.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related topics, see section *tMysqlSCD*.

tPostgresqlSCDELT

tPostgresqlSCDELT Properties

Component family	Databases/Postgresql	
Function	**tPostgresqlSCDELT** reflects and tracks changes in a dedicated Postgresql SCD table.	
Purpose	**tPostgresqlSCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated DB2 SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	Use an existing connection	Select this check box and click the relevant **tPostgresqlConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input DB2 SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created. **Create table if not exists**: A table gets created if it does not exist.

		Clear table: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	*Source Key*s	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Debug mode*	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see section *tMysqlSCD*.

tSPSSInput

Before being able to benefit from all functional objectives of the SPSS components, make sure to do the following: -If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program \SPSS, or -If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in ***Talend*** *root directory.*

tSPSSInput properties

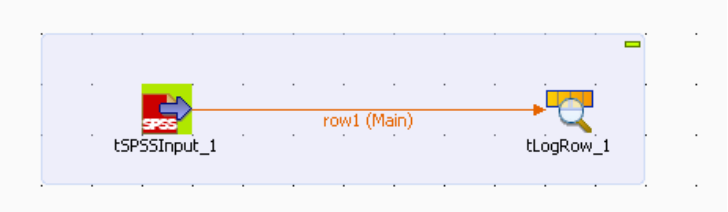
Component family	Business Intelligence	
Function	**tSPSSInput** reads data from an SPSS .sav file.	
Purpose	**tSPSSInput** addresses SPSS .sav data to write it for example in another file.	
Basic settings	*Sync schema*	Click this button to synchronize with the columns of the input SPSS .sav file.
	Schema and *Edit Schema*	The schema metadata in this component is retrieved directly from the input SPSS .sav file and thus is read-only. You can click **Edit schema** to view the retrieved metadata.
	Filename	Name or path of the SPSS .sav file to be read.
	Translate labels	Select this check box to translate the labels of the stored values. If you select this check box, you need to retrieve the metadata again.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is used as a start component. It requires an output flow.	

Scenario: Displaying the content of an SPSS .sav file

The following scenario creates a two-component Job, which aims at reading each row of a .sav file and displaying the output on the log console.

Setting up the Job

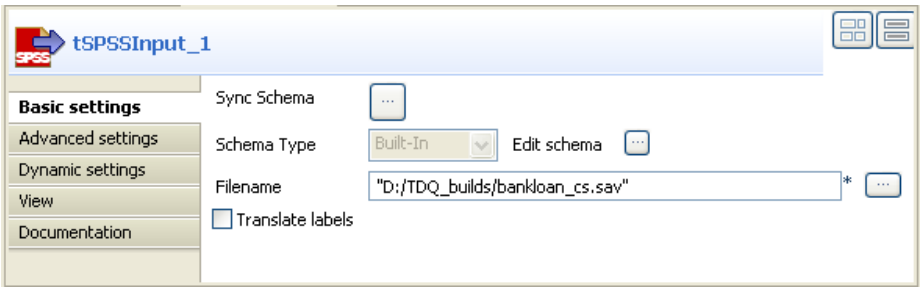
1. Drop a **tSPSSInput** component and a **tLogRow** component from the **Palette** onto the design workspace.



2. Right-click on **tPSSInput** and connect it to **tLogRow** using a **Main Row** link.

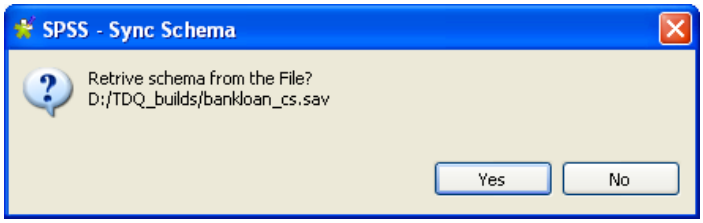
Configuring the input component

1. Click **tSPSSInput** to display its **Basic settings** view and define the component properties.



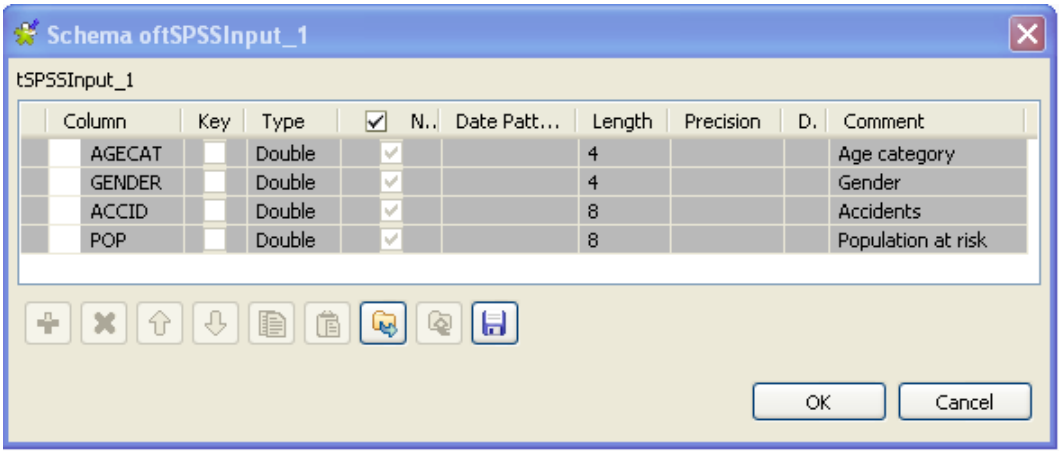
2. Click the three-dot button next to the **Filename** field and browse to the SPSS .sav file you want to read.

3. Click the three-dot button next to **Sync schema**. A message opens up prompting you to accept retrieving the schema from the defined SPSS file.



4. Click **Yes** to close the message and proceed to the next step.

5. If required, click the three-dot button next to **Edit schema** to view the pre-defined data structure of the source SPSS file.



6. Click **OK** to close the dialog box.

Job execution

Save the Job and press **F6** to execute it.

The SPSS file is read row by row and the extracted fields are displayed on the log console.

```
Starting job SPSS at 10:28 20/01/2010.

[statistics] connecting to socket on port 3802
[statistics] connected
1.0|1.0|57997.0|198522.0
2.0|1.0|57113.0|203200.0
3.0|1.0|54123.0|200744.0
1.0|0.0|63936.0|187791.0
2.0|0.0|64835.0|195714.0
3.0|0.0|66804.0|208239.0
[statistics] disconnected
Job SPSS ended at 10:28 20/01/2010. [exit code=0]
```

Translating the stored values

To translate the stored values, complete the following:

1. In the **Basic settings** view, select the **Translate label** check box.
2. Click **Sync Schema** a second time to retrieve the schema after translation.

   A message opens up prompting you to accept retrieving the schema from the defined SPSS file.
3. Click **Yes** to close the message and proceed to the next step.

   A second message opens up prompting you to accept propagating the changes.
4. Click **Yes** to close the message and proceed to the next step.
5. Save the Job and press **F6** to execute it.

The SPSS file is read row by row and the extracted fields are displayed on the log console after translating the stored values.

```
Starting job SPSS at 10:32 20/01/2010.

[statistics] connecting to socket on port 3418
[statistics] connected
Under 21|Female|57997.0|198522.0
21-25|Female|57113.0|203200.0
26-30|Female|54123.0|200744.0
Under 21|Male|63936.0|187791.0
21-25|Male|64835.0|195714.0
26-30|Male|66804.0|208239.0
[statistics] disconnected
Job SPSS ended at 10:32 20/01/2010. [exit code=0]
```

tSPSSOutput

Before being able to benefit from all functional objectives of the SPSS components, make sure to do the following: -If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program \SPSS, or -If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in ***Talend*** *root directory.*

tSPSSOutput properties

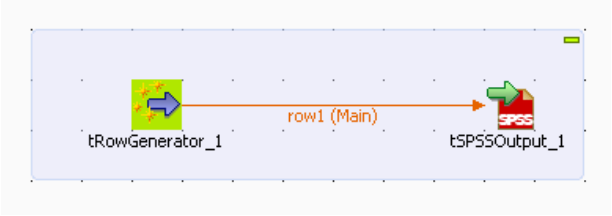
Component family	Business Intelligence	
Function	**tSPSSOutput** writes data entries in an .sav file.	
Purpose	**tSPSSOutput** writes or appends data to an SPSS .sav file. It creates SPSS files on the fly and overwrites existing ones.	
Basic settings	*Sync schema*	Click this button to synchronize with the columns of the SPSS .sav file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Filename	Name or path of the SPSS .sav file to be written.
	Write Type	Select an operation from the list: **Write**: simply writes the new data. **Append**: writes the new data at the end of the existing data.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can not be used as start component. It requires an input flow	

Scenario: Writing data in an .sav file

This Java scenario describes a very simple Job that writes data entries in an .sav file.

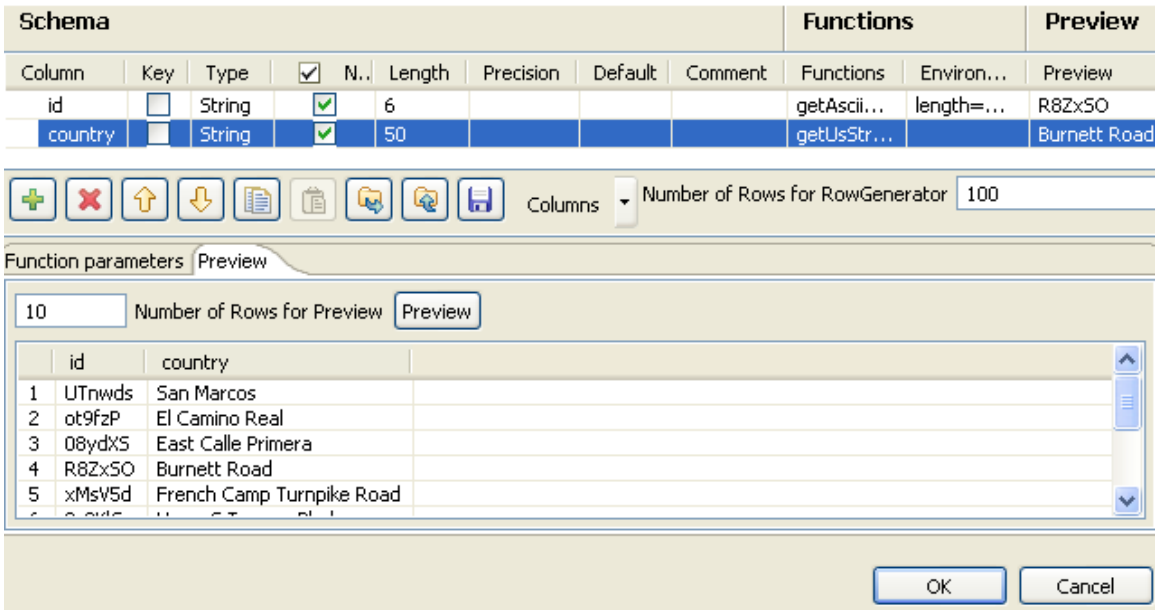
Setting up the Job

1. Drop a **tRowGenerator** component and a **tSPSSOutput** component from the **Palette** onto the design workspace.
2. Right-click on **tRowGenerator** and connect it to **tSPSSOutput** using a **Main Row** link.



Configuring the input component

1. In the design workspace, double click **tRowGenerator** to display its **Basic Settings** view and open its editor. Here you can define your schema.



2. Click the plus button to add the columns you want to write in the .sav file.

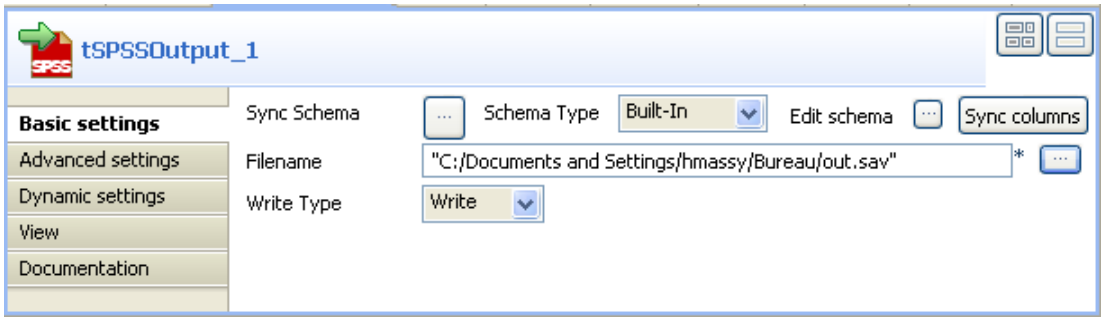
3. Define the schema and set the parameters to the columns.

Make sure to define the length of your columns. Otherwise, an error message will display when building your Job.

4. Click **OK** to validate your schema and close the editor.

Configuring the output component

1. Click **tSPSSOutput** to display its **Basic settings** view and define the component properties.



2. Click the three-dot button next to the **Filename** field and browse to the SPSS .sav file in which you want to write data.

3. Click the three-dot button next to **Sync columns** to synchronize columns with the previous component. In this example, the schema to be inserted in the .sav file consists of the two columns: *id* and *country*.

4. If required, click **Edit schema** to view/edit the defined schema.

5. From the **Write Type** list, select **Write** or **Append** to simply write the input data in the .sav file or add it to the end of the .sav file.

Job execution

Save the Job and press **F6** to execute it.

The data generated by the **tRowGenerator** component is written in the defined .sav file.

tSPSSProperties

*In order to benefit from all of the functional objectives of the SPSS components, do the following: -If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program\SPSS, or -If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in the **Talend** root directory.*

tSPSSProperties properties

Component family	Business Intelligence	
Function	**tSPSSProperties** describes the properties of a defined SPSS .sav file.	
Purpose	**tSPSSProperties** allows you to obtain information about the main properties of a defined SPSS .sav file.	
Basic settings	*Schema* and *Edit Schema*	The schema metadata in this component is predefined and thus read-only. You can click **Edit schema** to view the predefined metadata. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Filename	Name or path of the .sav file to be processed.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	Use this component as a start component. It needs an output flow.	

Related scenarios

For related topics, see:

- section *Scenario: Reading master data in an MDM hub*.
- section *Scenario: Writing data in an .sav file*.

tSPSSStructure

Before being able to benefit from all functional objectives of the SPSS components, make sure to do the following: -If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program \SPSS, or -If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in ***Talend*** *root directory.*

tSPSSStructure properties

Component family	Business Intelligence	
Function	**tSPSSStructure** retrieves information about the variables inside .sav files.	
Purpose	**tSPSSStructure** addresses variables inside .sav files. You can use this component in combination with **tFileList** to gather information about existing *.sav files to further analyze or check the findings.	
Basic settings	*Schema* and *Edit Schema*	The schema metadata in this component is predefined and thus read-only. It is based on the internal SPSS convention. You can click **Edit schema** to view the predefined metadata. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Filename	Name or path of the .sav file to be processed.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	Use this component as a start component. It needs an output flow.	

Related scenarios

For related topics, see:

- section *Scenario: Reading master data in an MDM hub*.
- section *Scenario: Writing data in an .sav file*.

tSybaseSCD

tSybaseSCD properties

Component family	Databases/Sybase	
Function	**tSybaseSCD** reflects and tracks changes in a dedicated Sybase SCD table.	
Purpose	**tSybaseSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see section *SCD management methodologies*.
	Use memory saving Mode	Select this check box to maximize system performance.

	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Debug mode	Select this check box to display each step during processing entries in a database.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see section *tMysqlSCD*.

tSybaseSCDELT

tSybaseSCDELT Properties

Component family	Databases/Sybase	
Function	**tSybaseSCDELT** reflects and tracks changes in a dedicated Sybase SCD table.	
Purpose	**tSybaselSCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated Sybase SCD table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input Sybase SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created. **Create table if not exists**: A table gets created if it does not exist.

		Clear table: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation. For more information regarding the creation methods, see section *SCD keys*.
	Source Keys	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related Scenario

For related topics, see section *tMysqlSCD* and section *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3)*.

Cloud components

This chapter details the main components which you can find in the **Cloud** family of the **Palette** in the Integration perspective of the **Talend** Studio.

Private and public cloud databases, data services and SaaS-based applications (CRM, HR, ERP, etc.) are springing up alongside on-premise applications and databases that have been the mainstay of corporate IT. The resulting hybrid IT environments have more sources, of more diverse types, which require more modes of integration, and more effort on data quality and consistency across sources.

The Cloud family comprises the most popular database connectors adapted to Cloud and SaaS applications and technologies.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAmazonMysqlClose

tAmazonMysqlClose properties

Function	**tAmazonMysqlClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tAmazonMysqlConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonMysql components, especially with **tAmazonMysqlConnection** and **tAmazonMysqlCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tAmazonMysqlCommit

tAmazonMysqlCommit Properties

This component is closely related to **tAmazonMysqlConnection** and **tAmazonMysqlRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Cloud/AmazonRDS/ MySQL	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tAmazonMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tAmazonMysqlCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonMysql components, especially with **tAmazonMysqlConnection** and **tAmazonMysqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAmazonMysqlConnection** and **tAmazonMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tAmazonMysqlConnection** component to open a connection for the current transaction.

For **tAmazonMysqlCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tAmazonMysqlConnection

tAmazonMysqlConnection Properties

This component is closely related to **tAmazonMysqlCommit** and **tAmazonMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tAmazonMysqlConnection** component to open a connection for the current transaction.

Component family	Cloud/AmazonRDS/ MySQL	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	MySQL 5 is available.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto Commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonMysql components, especially with **tAmazonMysqlCommit** and **tAmazonMysqlRollback** components.	
Limitation	n/a	

Scenario: Inserting data in mother/daughter tables

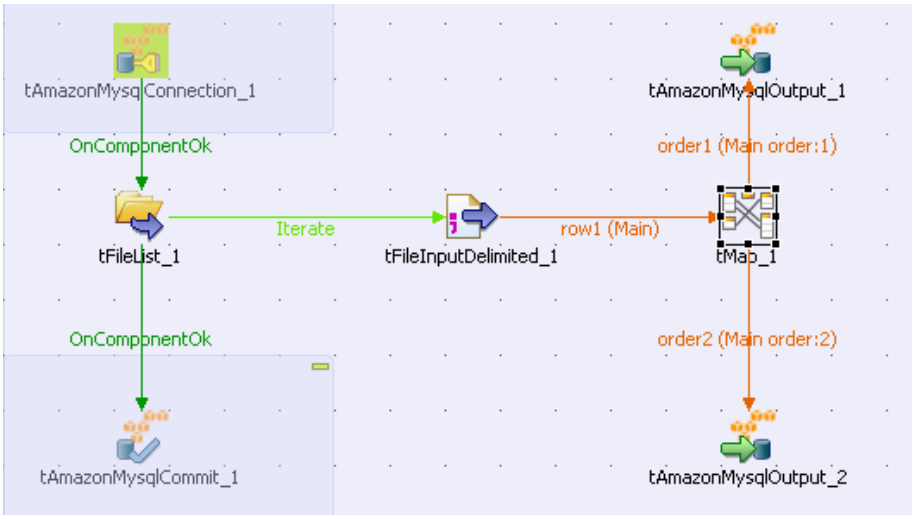
The following Job is dedicated to advanced database users, who want to carry out multiple table insertions using a parent table id to feed a child table. As a prerequisite to this Job, follow the steps described below to create the relevant tables using an engine such as *innodb*.

Setting up the Job

1. In a command line editor, connect to your Mysql server. Once connected to the relevant database, type in the following command to create the parent table: create table f1090_mum(id int not null auto_increment, name varchar(10), primary key(id)) engine=innodb.

2. Then create the second table: create table baby (id_baby int not null, years int) engine=innodb.

   Back into *Talend Open Studio for Big Data*, the Job requires seven components including **tAmazonMysqlConnection** and **tAmazonMysqlCommit**.



3. Drag and drop the following components from the **Palette**: **tFileList**, **tFileInputDelimited**, **tMap**, **tAmazonMysqlOutput** (x2).

4. Connect the **tFileList** component to the input file component using an **Iterate** link as the name of the file to be processed will be dynamically filled in from the **tFileList** directory using a global variable.

5. Connect the **tFileInputDelimited** component to the **tMap** and dispatch the flow between the two output AmazonMysql DB components. Use a **Row** link for each for these connections representing the main data flow.

6. Set the **tFileList** component properties, such as the directory name where files will be fetched from.

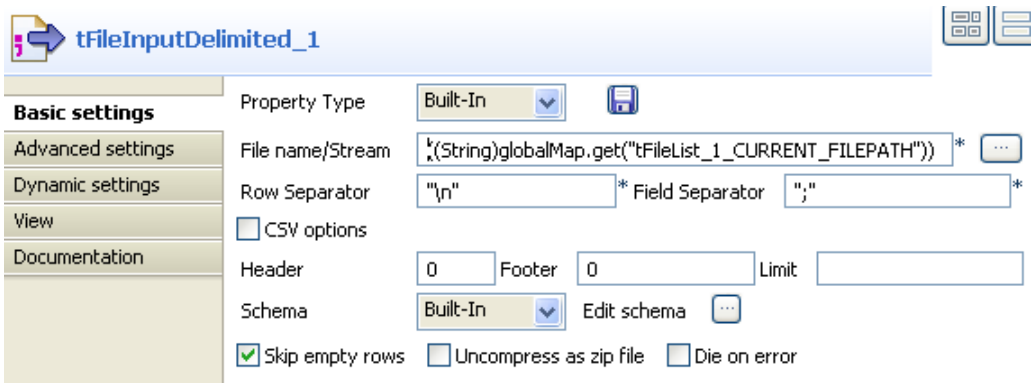
7. Add a **tAmazonMysqlConnection** component and connect it to the starter component of this job, in this example, the **tFileList** component using an **OnComponentOk** link to define the execution order.

Setting up the DB connection

In the **tAmazonMysqlConnection** Component view, set the connection details.

Configuring the input component

1. On the **tFileInputDelimited** component's **Basic settings** panel, press **Ctrl+Space** bar to access the variable list. Set the **File Name** field to the global variable: *tFileList_1.CURRENT_FILEPATH*



2. Set the rest of the fields as usual, defining the row and field separators according to your file structure. Then set the schema manually through the **Edit schema** feature. Make sure the data type is correctly set, in accordance with the nature of the data processed.

Configuring the tMap component

1. In the **tMap** Output area, add two output tables, one called mum for the parent table, the second called baby, for the child table.

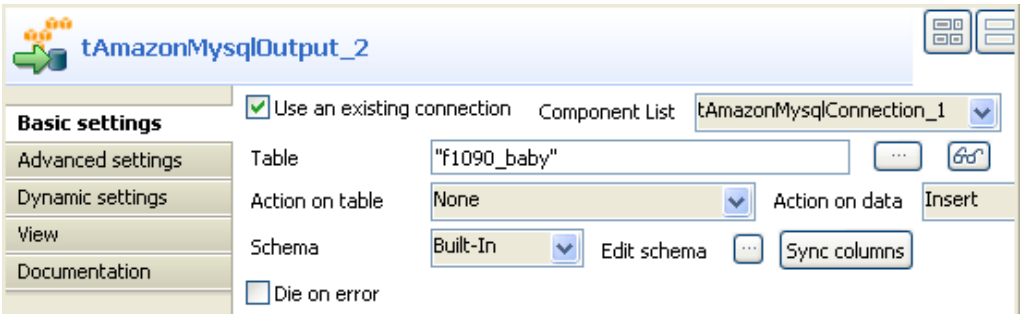
2. Drag the *Name* column from the **Input** area, and drop it to the mum table. Drag the *Years* column from the **Input** area and drop it to the baby table.

Make sure the mum table is on the top of the baby table as the order is determining for the flow sequence hence the DB insert to perform correctly.

3. Then connect the output row link to distribute correctly the flow to the relevant DB output component.

Configuring the output component

1. In each of the **tAmazonMysqlOutput** components' **Basic settings** panel, select the **Use an existing connection** check box to retrieve the **tAmazonMysqlConnection** details.

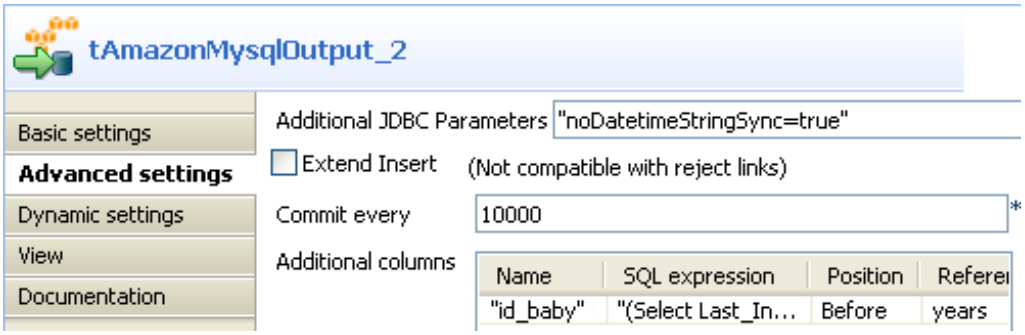


2. Set the **Table** name making sure it corresponds to the correct table, in this example either *f1090_mum* or *f1090_baby*.

   There is no action on the table as they are already created.

3. Select **Insert** as **Action on data** for both output components. Click on **Sync columns** to retrieve the schema set in the **tMap**.

4. Go to the **Advanced settings** panel of each of the **tAmazonMysqlOutput** components. Notice that the **Commit every** field will get overridden by the **tAmazonMysqlCommit**.



5. In the **Additional columns** area of the DB output component corresponding to the child table (*f1090_baby*), set the id_baby column so that it reuses the id from the parent table. In the **SQL expression** field type in: '(Select Last_Insert_id())'.

   The position is *Before* and the **Reference column** is *years*.

Configuring the tAmazonMysqlCommit component

1. Add the **tAmazonMysqlCommit** component to the design workspace and connect it from the **tFileList** component using a **OnComponentOk** connection in order for the Job to terminate with the transaction commit.

2. On the **tAmazonMysqlCommit Component** view, select in the list the connection to be used.

Job execution

Save your Job and press **F6** to execute it.

```
mysql> select * from f1090_mum
    -> ;
+----+--------+
| id | names  |
+----+--------+
|  6 | john   |
|  7 | bruce  |
|  8 | beth   |
|  9 | andrew |
| 10 | donald |
| 11 | betty  |
| 12 | john   |
| 13 | bruce  |
| 14 | beth   |
| 15 | andrew |
| 16 | donald |
| 17 | betty  |
+----+--------+
12 rows in set (0.00 sec)
```

```
mysql> select * from f1090_baby
    -> ;
+---------+-------+
| id_baby | years |
+---------+-------+
|       6 |    10 |
|       7 |    23 |
|       8 |    34 |
|       9 |    10 |
|      10 |    23 |
|      11 |    34 |
|      12 |    10 |
|      13 |    23 |
|      14 |    34 |
|      15 |    10 |
|      16 |    23 |
|      17 |    34 |
+---------+-------+
12 rows in set (0.00 sec)
```

The parent table *id* has been reused to feed the *id_baby* column.

tAmazonMysqlInput

tAmazonMysqlInput properties

Component family	Cloud/AmazonRDS/MySQL	
Function	**tAmazonMysqlInput** reads a database and extracts fields based on a query.	
Purpose	**tAmazonMysqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	MySQL 5 is available.
	Use an existing connection	Select this check box when using a configured **tAmazonMysqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. When you need to handle data of the time-stamp type *0000-00-00 00:00:00* using this component, set the parameter as: `noDatetimeStringSync=true&zeroDa-teTimeBehavior=convertToNull`.

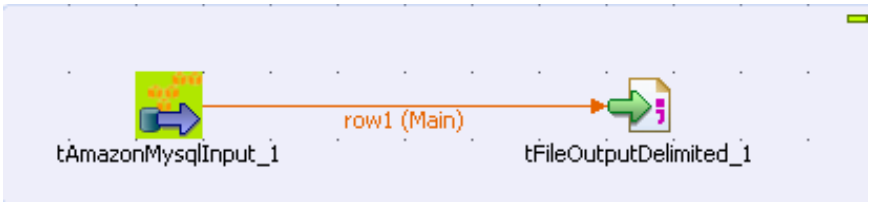
	Enable stream	Select this check box to enables streaming over buffering which allows the code to read from a large table without consuming a large amount of memory in order to optimize the performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns. Deselect **Trim all the String/Char columns** to enable Trim columns in this field.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Mysql databases.	

Scenario1: Writing columns from a MySQL database to an output file

In this scenario we will read certain columns from a MySQL database, and then write them to a table in a local output file.

Setting up the Job

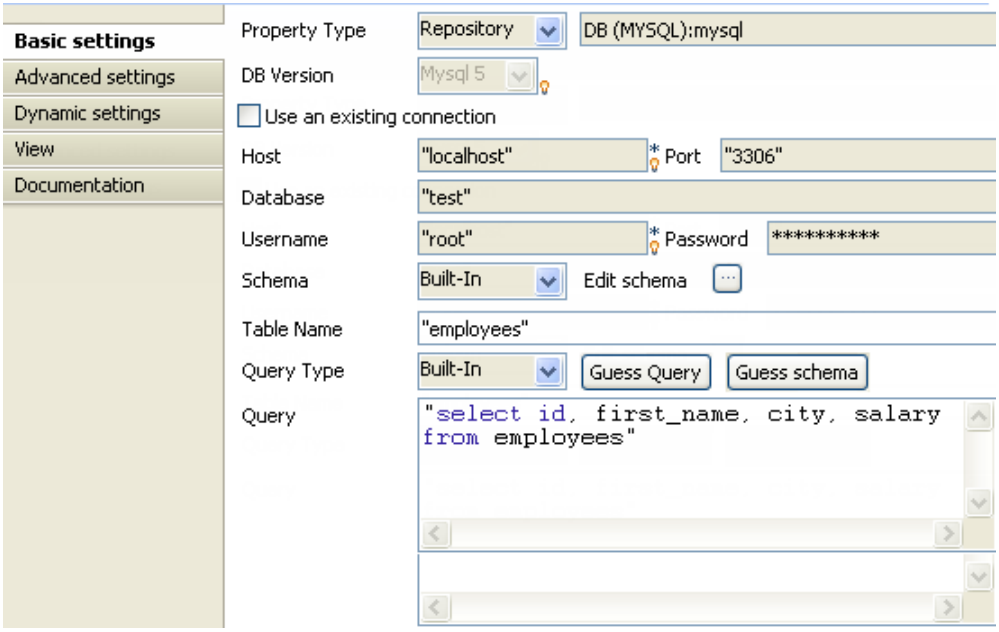
1. Drop **tAmazonMysqlInput** and **tFileOutputDelimited** from the **Palette** onto the workspace.



2. Link **tAmazonMysqlInput** to **tFileOutputDelimited** using a **Row > Main** connection.

Configuring the input component

1. Double-click **tAmazonMysqlInput** to open its **Basic Settings** view in the **Component** tab.

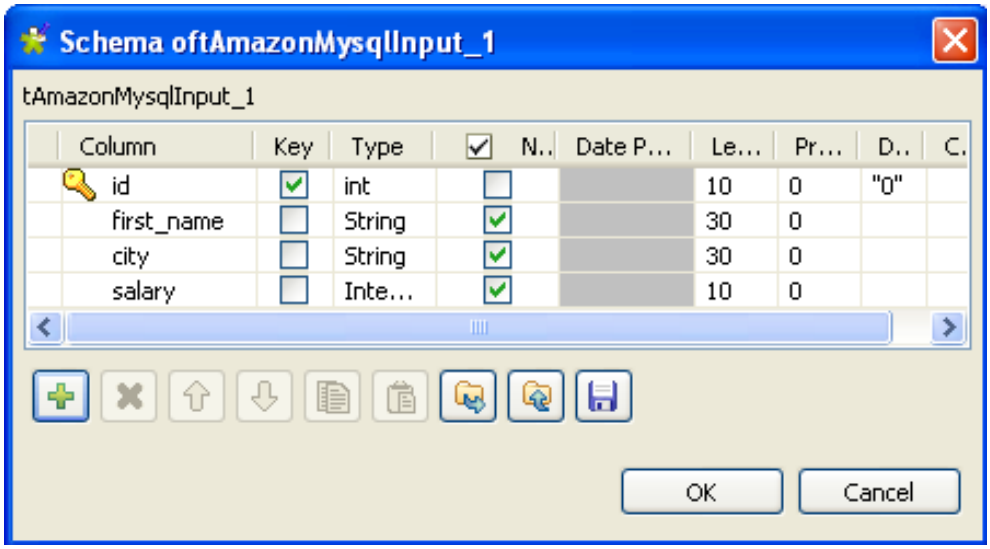


*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Define the database connection details.

3. Click **Edit schema** to define the desired schema.

   The schema editor opens:



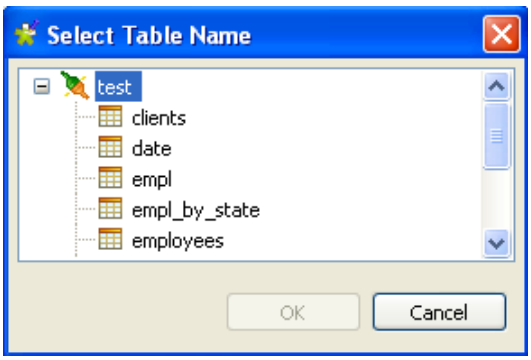
4. Click the **[+]** button to add the rows that you will use to define the schema, four columns in this example *id*, *first_name*, *city* and *salary*.

5. Under **Column**, click in the fields to enter the corresponding column names.

6. Click the field under **Type** to define the type of data. Click **OK** to close the schema editor.

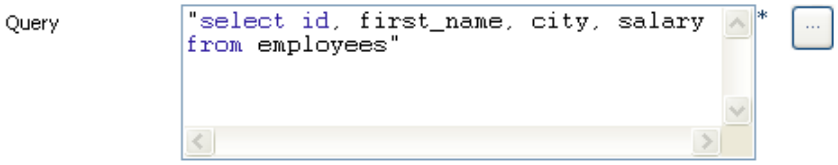
7. Next to the **Table Name** field, click the **[...]** button to select the database table of interest.

   A dialog box displays a tree diagram of all the tables in the selected database:



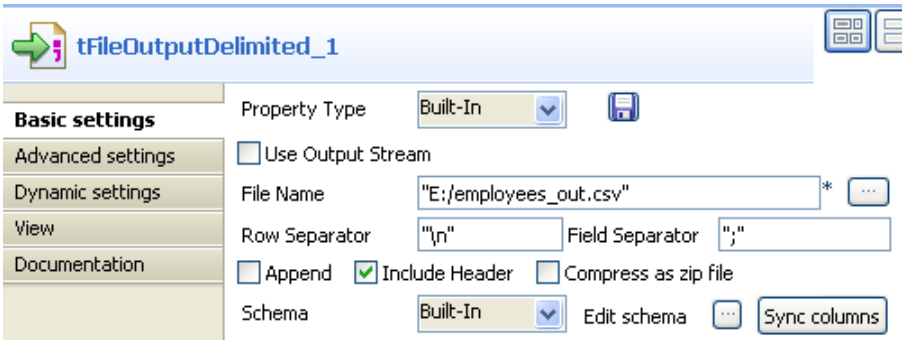
8. Click the table of interest and then click **OK** to close the dialog box.

9. In the **Query** box, enter the query required to retrieve the desired columns from the table.



Configuring the output component

1. Double-click **tFileOutputDelimited** to set its **Basic Settings** in the **Component** tab.



2. Next to the **File Name** field, click the **[...]** button to browse your directory to where you want to save the output file, then enter a name for the file.

3. Select the **Include Header** check box to retrieve the column names as well as the data.

Job execution

Save the Job and press **F6** to run it.

The output file is written with the desired column names and corresponding data, retrieved from the database:

```
1  id;first_name;city;salary
2  1;Martin;Sacramento;9011
3  2;Zachary;Atlanta;8118
4  3;James;Hartford;5087
5  4;Herbert;Charleston;9233
6  5;Herbert;Hartford;6289
7  6;Ulysses;Nashville;6269
```

The Job can also be run in the **Traces Debug** mode, which allows you to view the rows as they are being written to the output file, in the workspace.

tAmazonMysqlOutput

tAmazonMysqlOutput properties

Component family	Cloud/AmazonRDS/ MySQL	
Function	**tAmazonMysqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAmazonMysqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	MySQL 5 is available.
	Use an existing connection	Select this check box when using a configured **tAmazonMysqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear table**: The table content is deleted.

		Truncate table: The table content is quickly deleted. However, you will not be able to rollback the operation.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, the job stops. **Update**: Make changes to existing entries. **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or creates them if they do not exist. **Delete**: Remove entries corresponding to the input flow. **Replace**: Add new entries to the table. If an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted. **Insert or update on duplicate key or unique index**: Add entries if the inserted value does not exist or update entries if the inserted value already exists and there is a risk of violating a unique index or primary key. **Insert Ignore**: Add only new rows to prevent duplicate key errors. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row in error and complete the process for error-free rows. If needed, you can retrieve the rows in error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Extend Insert	Select this check box to carry out a bulk insert of a defined set of lines instead of inserting lines one by one. The gain in system performance is considerable. **Number of rows per insert**: enter the number of rows to be inserted per operation. Note that the higher the value specidied, the lower performance levels shall be due to the increase in memory demands. This option is not compatible with the **Reject** link. You should therefore clear the check box if you are using a **Row > Rejects** link with this component.

		*If you are using this component with **tMysqlLastInsertID**, ensure that the **Extend Insert** check box in **Advanced Settings** is not selected. **Extend Insert** allows for batch loading, however, if the check box is selected, only the ID of the last line of the last batch will be returned.*
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected, the **Update** or the **Delete** option in the **Action on data** field.
	Commit every	Number of rows to be included in the batch before it is committed to the DB. This option ensures transaction quality (but not rollback) and, above all, a higher performance level.
	Additional Columns	This option is not available if you have just created the DB table (even if you delete it beforehand). This option allows you to call SQL functions to perform actions on columns, provided that these are not insert, update or delete actions, or actions that require pre-processing.
		Name: Type in the name of the schema column to be altered or inserted.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the data in the corrsponding column.
		Position: Select **Before**, **Replace** or **After**, depending on the action to be performed on the reference column.
		Reference column: Type in a reference column that **tAmazonMysqlOutput** can use to locate or replace the new column, or the column to be modified.
	Use field options	Select this check box to customize a request, particularly if multiple actions are being carried out on the data.
	Use Hint Options	Select this check box to activate the hint configuration area which helps you optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */.` - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	Enable debug mode	Select this check box to display each step involved in the process of writing data in the database.
	Use duplicate key update mode insert	Updates the values of the columns specified, in the event of duplicate primary keys.: **Column**: Between double quotation marks, enter the name of the column to be updated. **Value**: Enter the action you want to carry out on the column. To use this option you must first of all select the **Insert** mode in the **Action on data** list found in the **Basic Settings** view.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a MySQL database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tAmazonMysqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

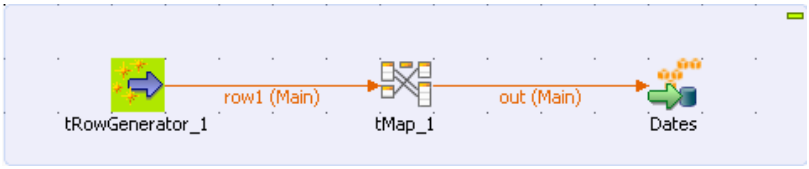
Scenario 1: Adding a new column and altering data in a DB table

This Java scenario is a three-component Job that aims at creating random data using a **tRowGenerator**, duplicating a column to be altered using the **tMap** component, and eventually altering the data to be inserted based on an SQL expression using the **tAmazonMysqlOutput** component.

Setting up the Job

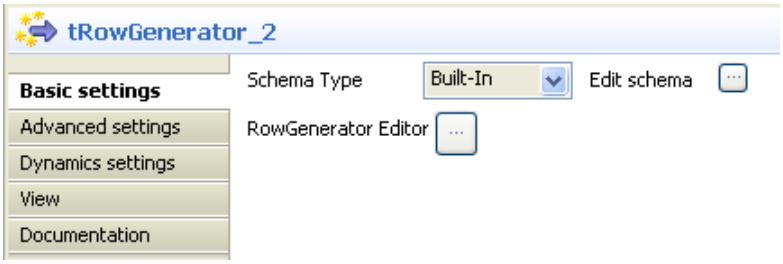
1. Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tMap** and **tAmazonMySQLOutput**.

2. Connect **tRowGenerator**, **tMap**, and **tAmazonMysqlOutput** using the **Row Main** link.

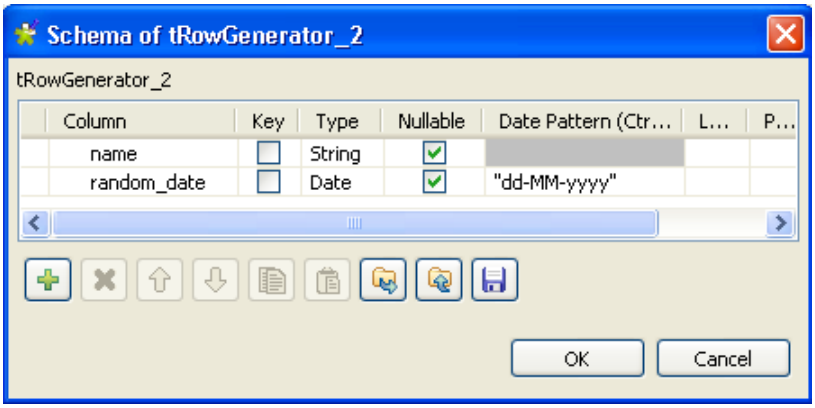


Configuring the input component

1. In the design workspace, select **tRowGenerator** to display its **Basic settings** view.

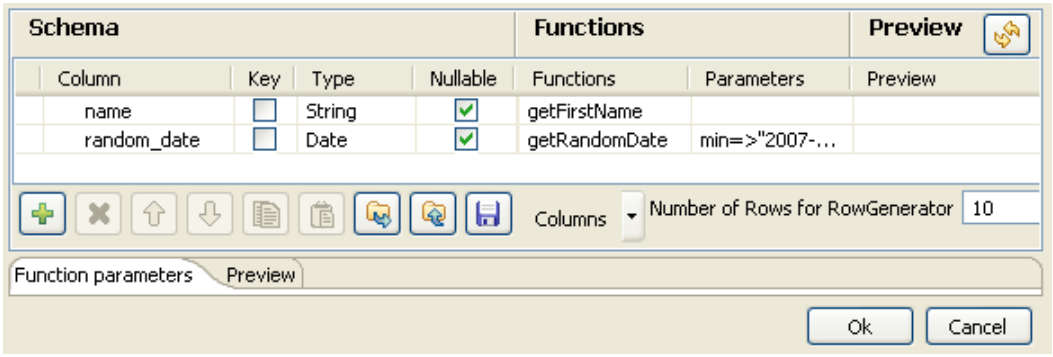


2. Click the **Edit schema** three-dot button to define the data to pass on to the **tMap** component, two columns in this scenario, *name* and *random_date*.



3. Click **OK** to close the dialog box.

4. Click the **RowGenerator Editor** three-dot button to open the editor and define the data to be generated.

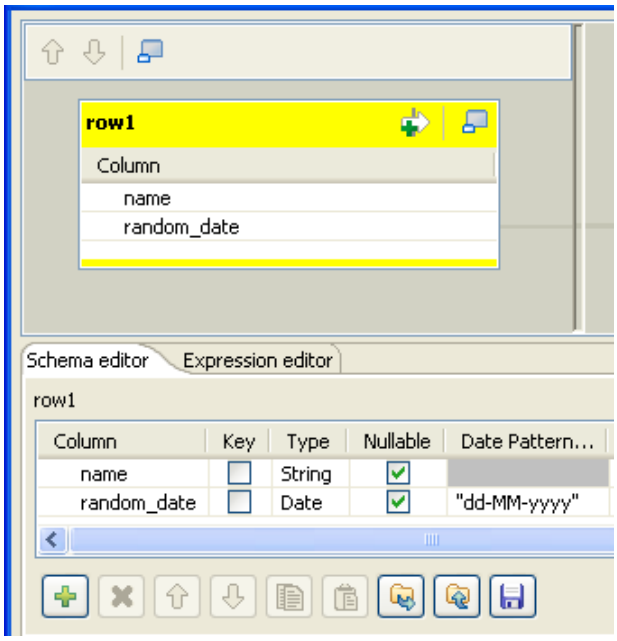


5. Click in the corresponding **Functions** fields and select a function for each of the two columns, `getFirstName` for the first column and `getrandomDate` for the second column.

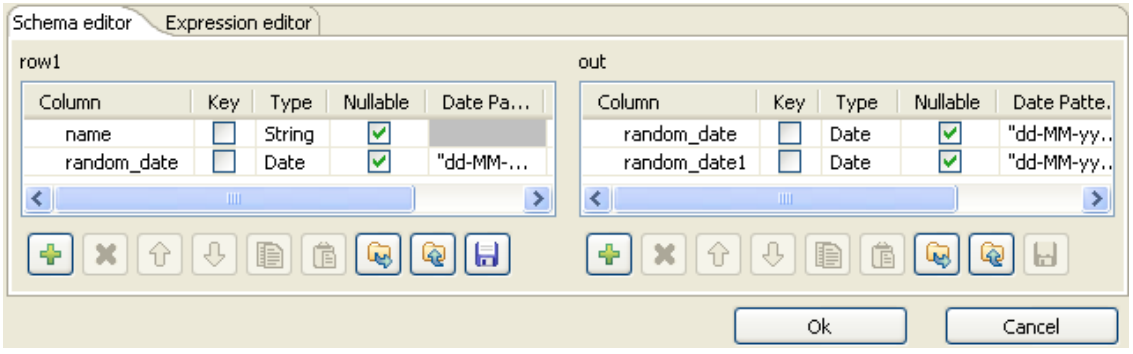
6. In the **Number of Rows for Rowgenerator** field, enter 10 to generate ten first name rows and click **Ok** to close the editor.

Configuring the tMap component

1. Double-click the **tMap** component to open the Map editor. The Map editor opens displaying the input metadata of the **tRowGenerator** component.

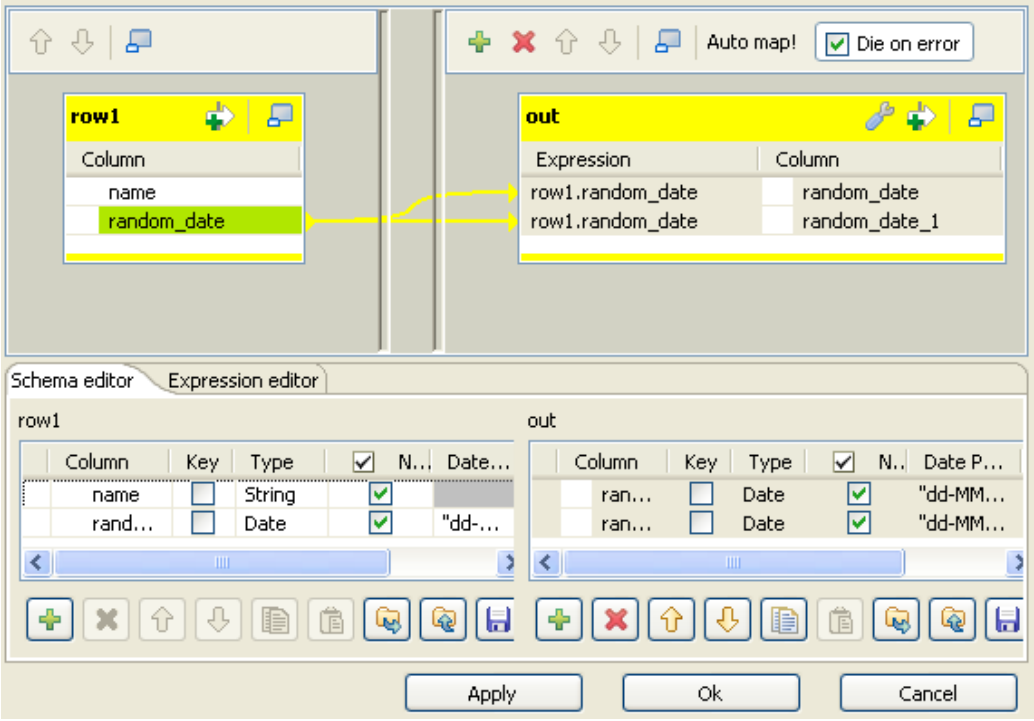


2. In the **Schema editor** panel of the Map editor, click the **[+]** button of the output table to add two rows and define the first as *random_date* and the second as *random_date1*.



In this scenario, we want to duplicate the *random_date* column and adapt the schema in order to alter the data in the output component.

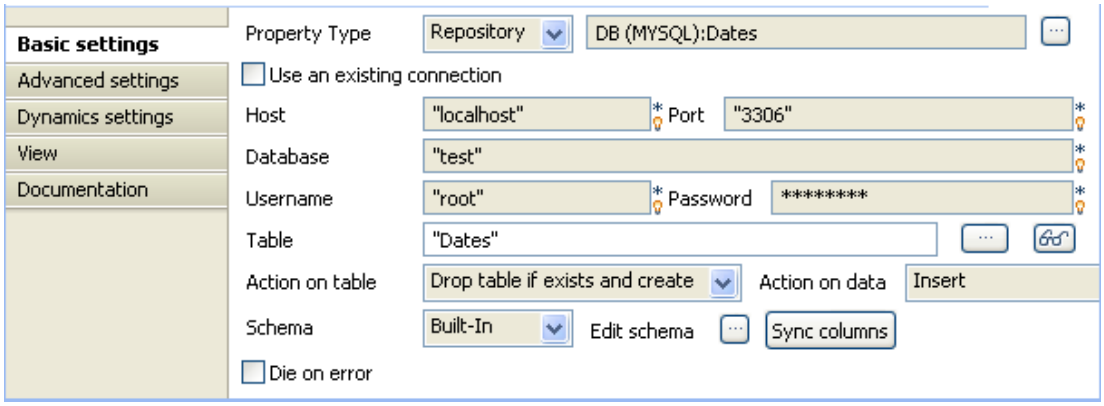
3. In the Map editor, drag the *random_date* row from the input table to the *random_date* and *random_date1* rows in the output table.



4. Click **OK** to close the editor.

Configuring the output component

1. In the design workspace, double-click the **tAmazonMysqlOutput** component to display its **Basic settings** view and set its parameters.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

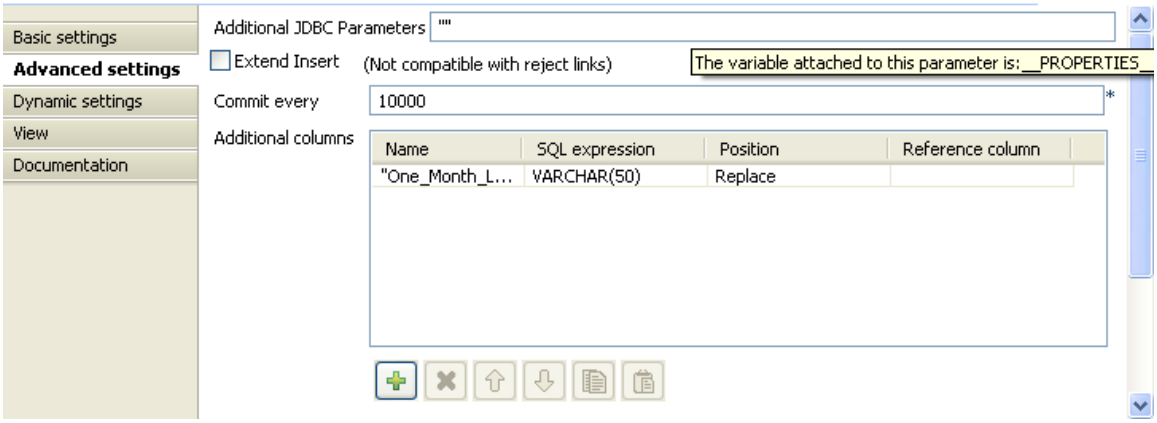
2. Fill in the database connection details.

3. Click the **[...]** button next to the **Table** field and select the table to be altered, *Dates* in this scenario.

4. On the **Action on table** list, select **Drop table if exists and create**, select *Insert* on the **Action on data** list.

5. If needed, click **Sync columns** to synchronize with the columns coming from the **tMap** component.

6. Click the **Advanced settings** tab to display the corresponding view and set the advanced parameters.



7. In the **Additional Columns** area, set the alteration to be performed on columns.

   In this scenario, the *One_month_later* column replaces *random_date_1*. Also, the data itself gets altered using an SQL expression that adds one month to the randomly picked-up date of the *random_date_1* column. ex: 2007-08-12 becomes 2007-09-12.

   -Enter *One_Month_Later* in the **Name** cell.

   -In the **SQL expression** cell, enter the relevant addition script to be performed, `"adddate(Random_date, interval 1 month)"` in this scenario.

   -Select **Replace** on the **Position** list.

   -Enter *Random_date1* on the **Reference column** list.

For this job we duplicated the *random_date_1* column in the DB table before replacing one instance of it with the *One_Month_Later* column. The aim of this workaround was to be able to view upfront the modification performed.

Job execution

Save your Job and press **F6** to execute it.

The new *One_month_later* column replaces the *random_date1* column in the DB table and adds one month to each of the randomly generated dates.

Random_date	One_Month_Later
2007-03-13 21:52:09	2007-04-13 21:52:09
2008-11-02 15:33:29	2008-12-02 15:33:29
2008-05-17 18:47:15	2008-06-17 18:47:15
2007-09-06 17:44:36	2007-10-06 17:44:36
2008-11-17 11:46:36	2008-12-17 11:46:36
2007-05-19 00:46:58	2007-06-19 00:46:58
2008-04-20 15:36:00	2008-05-20 15:36:00
2007-07-24 21:06:08	2007-08-24 21:06:08
2008-04-03 13:24:17	2008-05-03 13:24:17
2007-06-29 13:22:23	2007-07-29 13:22:23

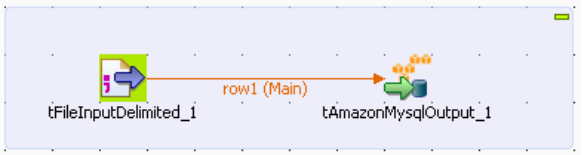
Related topic: see section *tDBOutput properties*.

Scenario 2: Updating data in a database table

This Java scenario describes a two-component Job that updates data in a MySQL table according to that in a delimited file.

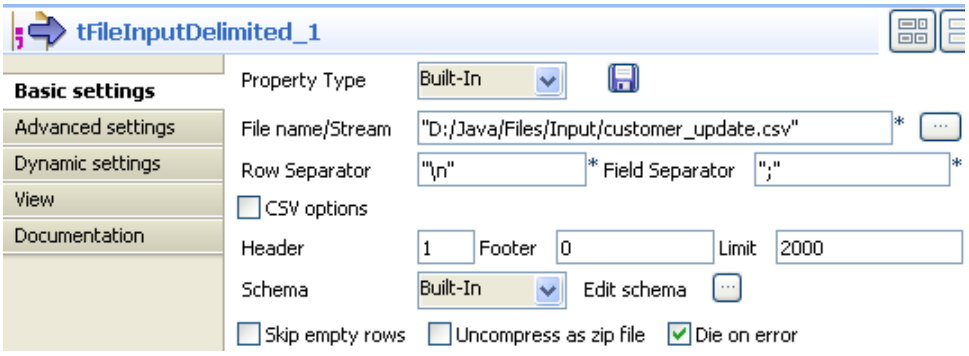
Setting up the Job

- Drop **tFileInputDelimited** and **tAmazonMysqlOutput** from the **Palette** onto the design workspace. Connect the two components together using a **Row Main** link.



Configuring the input component

1. Double-click **tFileInputDelimited** to display its **Basic settings** view and define the component properties.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

2. Fill in the database connection details.

3. In the **File Name** field, click the **[...]** button and browse to the source delimited file that contains the modifications to propagate in the MySQL table.

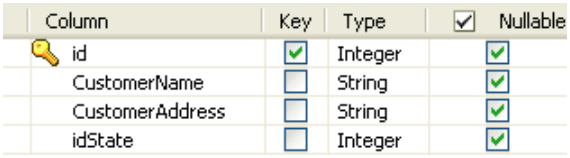
   In this example, we use the *customer_update* file that holds four columns: *id*, *CustomerName*, *CustomerAddress* and *idState*. Some of the data in these four columns is different from that in the MySQL table.

id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Marks, and Kaplan Ltd.	1949 Cloverdale Rd.	9

4. Define the row and field separators used in the source file in the corresponding fields. If needed, set **Header**, **Footer** and **Limit**.

   In this example, **Header** is set to 1 since the first row holds the names of columns, therefore it should be ignored. Also, the number of processed lines is limited to 2000.

5. Click the **[...]** button next to **Edit Schema** to open a dialog box where you can describe the data structure of the source delimited file that you want to pass to the component that follows.

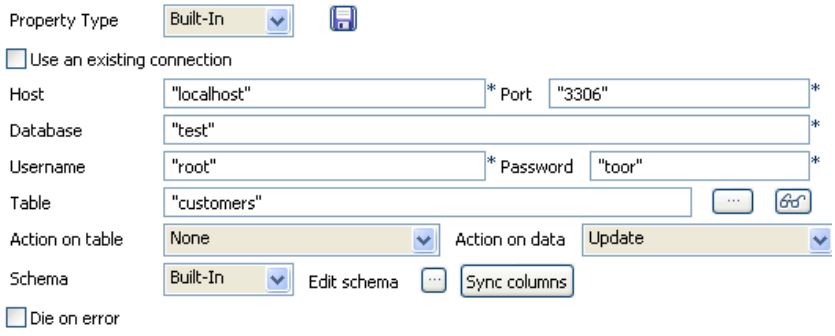


6. Select the **Key** check box(es) next to the column name(s) you want to define as key column(s).

   It is necessary to define at least one column as a key column for the Job to be executed correctly. Otherwise, the Job is automatically interrupted and an error message displays on the console.

Configuring the output component

1. In the design workspace, double-click **tAmazonMysqlOutput** to open its **Basic settings** view where you can define its properties.



2. Click **Sync columns** to retrieve the schema of the preceding component. If needed, click the **[...]** button next to **Edit schema** to open a dialog box where you can check the retrieved schema.

3. Fill in the database connection information in the corresponding fields.

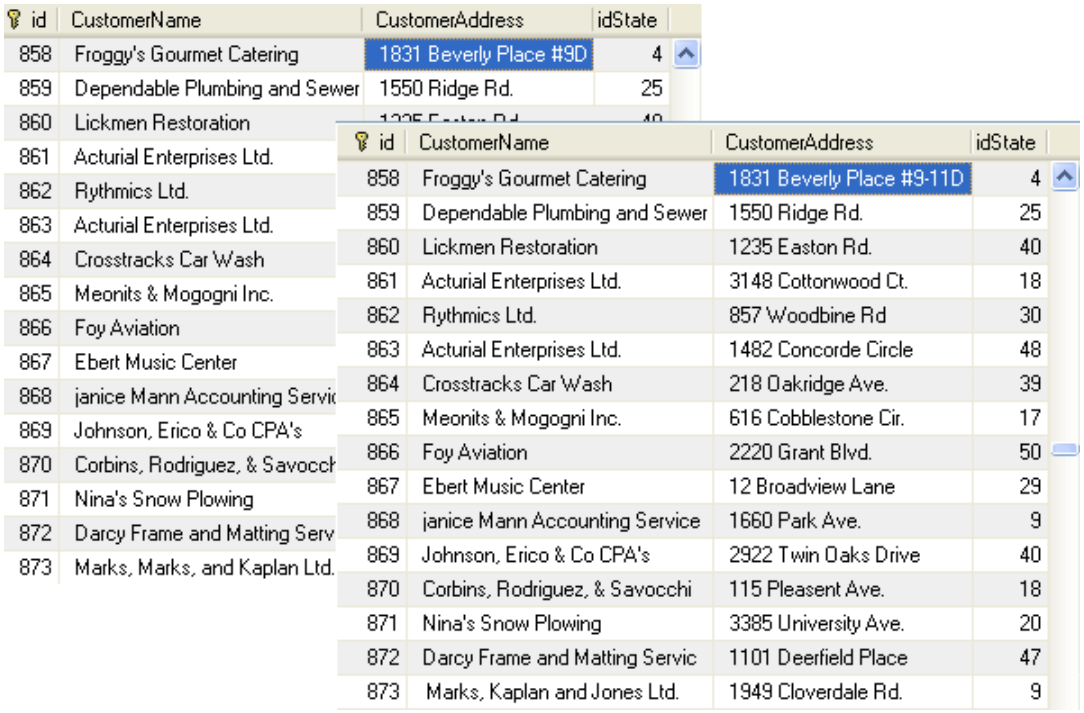
4. In the **Table** field, enter the name of the table to update.

5. From the **Action on table** list, select the operation you want to perform, **None** in this example since the table already exists.

6. From the **Action on data** list, select the operation you want to perform on the data, **Update** in this example.

Job execution

Save your Job and press **F6** to execute it.



id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration		
861	Acturial Enterprises Ltd.		
862	Rythmics Ltd.		
863	Acturial Enterprises Ltd.		
864	Crosstracks Car Wash		
865	Meonits & Mogogni Inc.		
866	Foy Aviation		
867	Ebert Music Center		
868	janice Mann Accounting Servi		
869	Johnson, Erico & Co CPA's		
870	Corbins, Rodriguez, & Savocch		
871	Nina's Snow Plowing		
872	Darcy Frame and Matting Serv		
873	Marks, Marks, and Kaplan Ltd.		

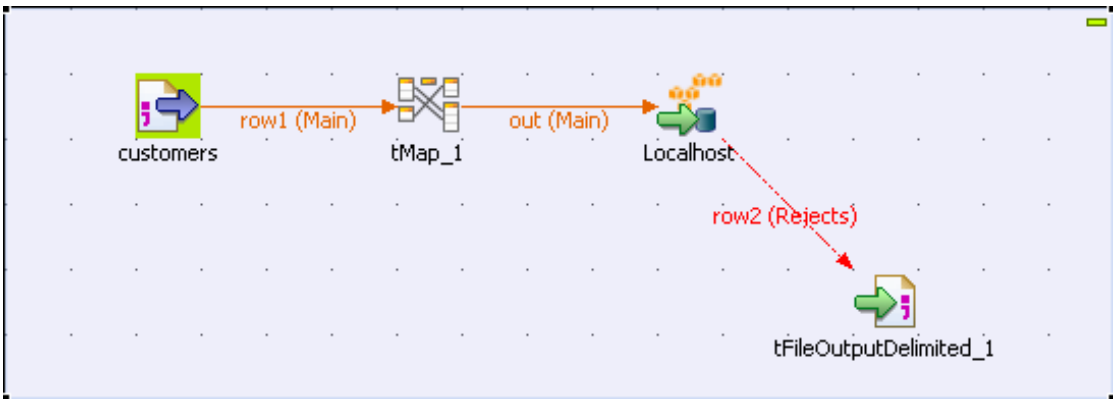
id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9-11D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 Cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Kaplan and Jones Ltd.	1949 Cloverdale Rd.	9

Using your DB browser, you can verify if the MySQL table, *customers*, has been modified according to the delimited file.

In the above example, the database table has always the four columns *id*, *CustomerName*, *CustomerAddress* and *idState*, but certain fields have been modified according to the data in the delimited file used.

Scenario 3: Retrieve data in error with a Reject link

This scenario describes a four-component Job that carries out migration from a customer file to a MySQL database table and redirects data in error towards a CSV file using a **Reject** link.

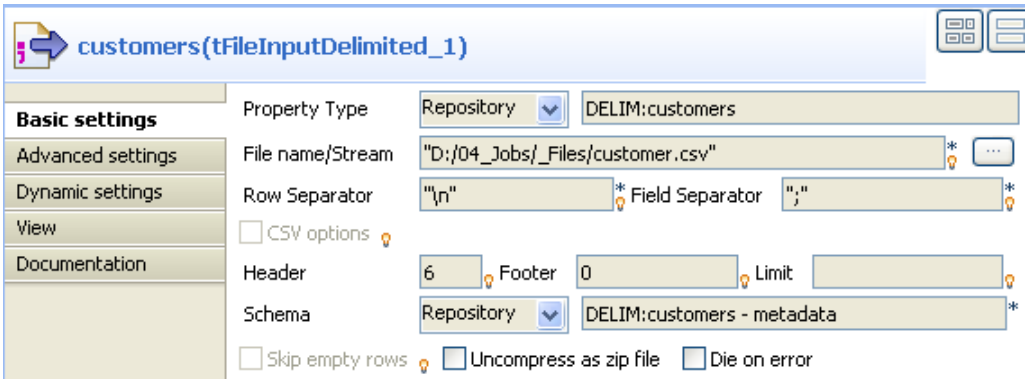


Setting up the Job

1. Drop a **tFileInputDelimited** component from the family **File** > **Input**, in the **Palette**, and fill in its properties manually in the **Component** tab.
2. From the **Palette**, drop a **tMap** from the **Processing** family onto the workspace.
3. Drop a **tAmazonMysqlOutput** from the **Databases** family in the **Palette** and fill in its properties manually in the **Component** tab.
4. From the **Palette**, select a **tFileOutputDelimited** from the **File** > **Output** family, and drop it onto the workspace.
5. Link the **customers** component to the **tMap** component, and the **tMap** and **Localhost** with a **Row Main** link. Name this second link *out*.
6. Link the **Localhost** to the **tFileOutputDelimited** using a **Row** > **Reject** link.

Configuring the input component

1. Double-click the **customers** component to display the **Component** view.



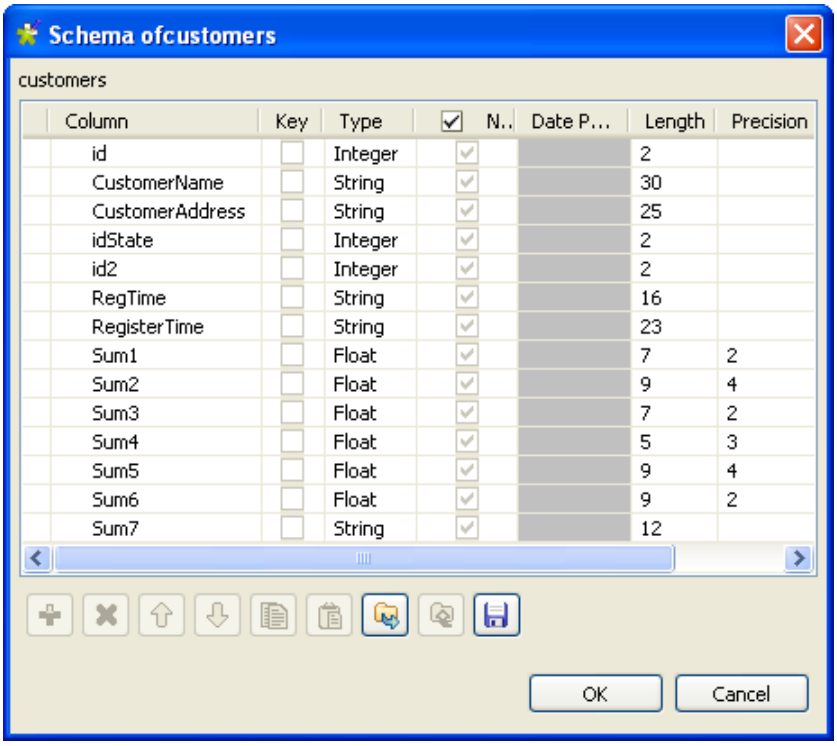
*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

2. Click the **[...]** button next to the **File Name** field, and fill in the path and the name of the file you want to use.
3. In the **Row** and **Field Separator** fields, type in between inverted commas the row and field separator used in the file.

4. In the **Header**, **Footer** and **Limit** fields, type in the number of headers and footers to ignore, and the number of rows to which processing should be limited.

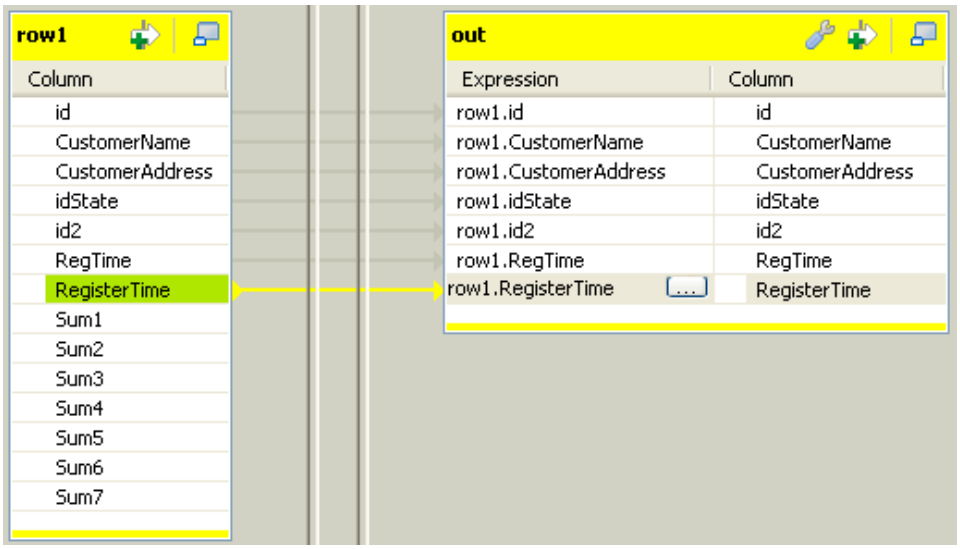
5. Click the **[...]** button next to the **Edit schema** field, and set the schema manually.

   The schema is as follows:

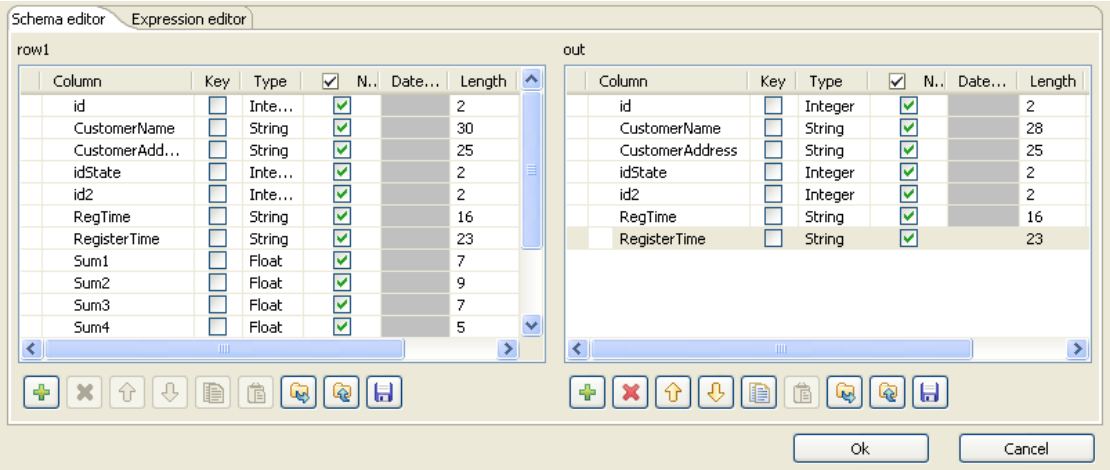


Configuring the tMap component

1. Double-click the **tMap** component to open its editor.

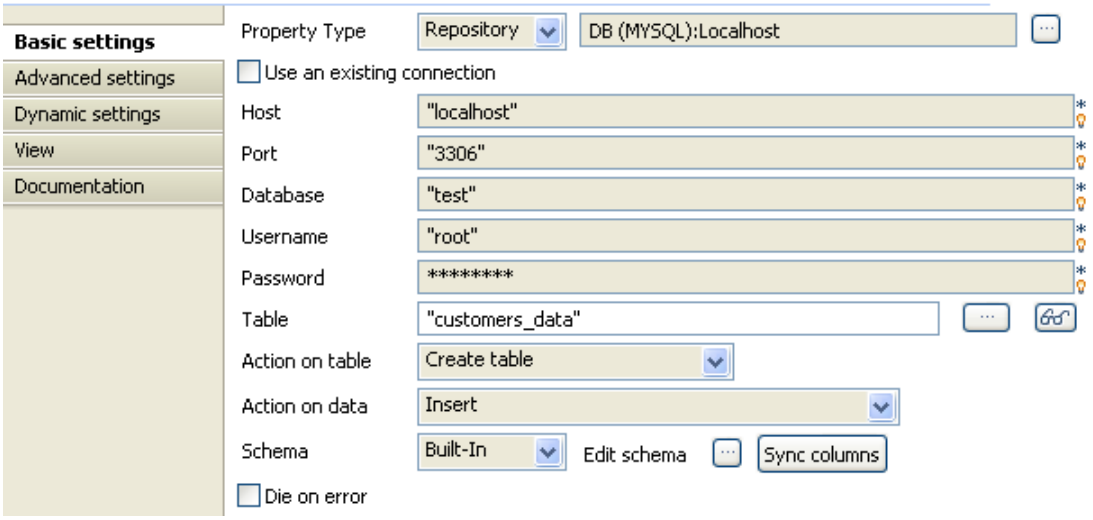


2. Select the *id*, *CustomerName*, *CustomerAddress*, *idSate*, *id2*, *RegTime* and *RegisterTime* columns on the table on the left and drop them on the **out** table, on the right.



3. In the **Schema editor** area, at the bottom of the **tMap** editor, in the right table, change the length of the **CustomerName** column to *28* to create an error. Thus, any data for which the length is greater than 28 will create errors, retrieved with the **Reject** link.

4. Click **OK**. In the workspace, double-click the output **Localhost** component to display its **Component** view.



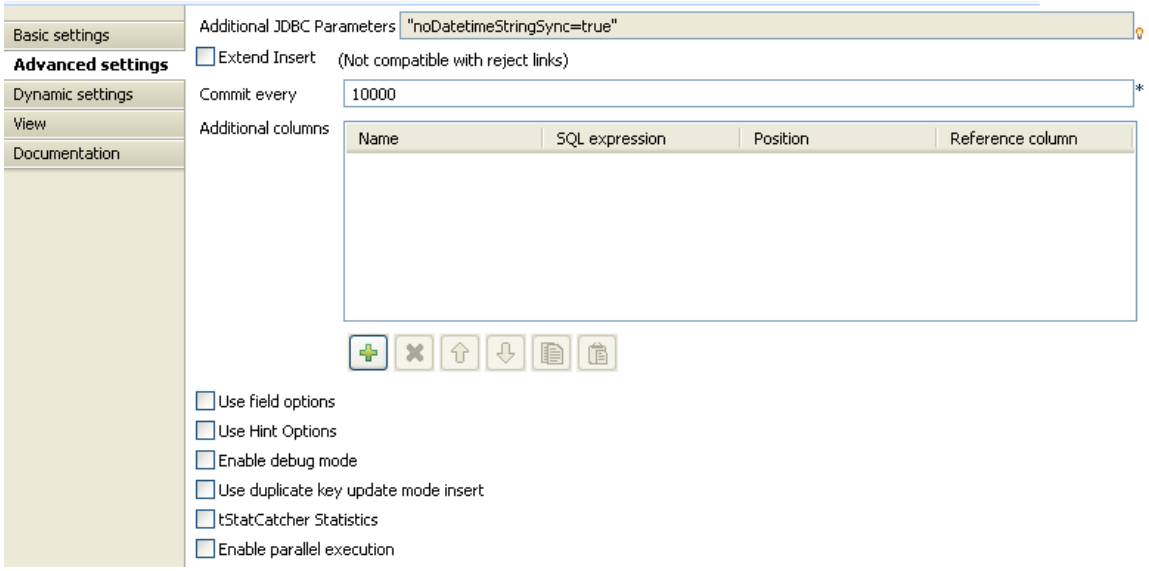
*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

5. Fill in the database connection details in the corresponding fields.

6. In the **Table** field, type in the name of the table to be created. In this scenario, we call it *customers_data*. In the **Action on data** list, select the **Create table** option. Click the **Sync columns** button to retrieve the schema from the previous component.

   Make sure the **Die on error** check box isn't selected, so that the Job can be executed despite the error you just created.

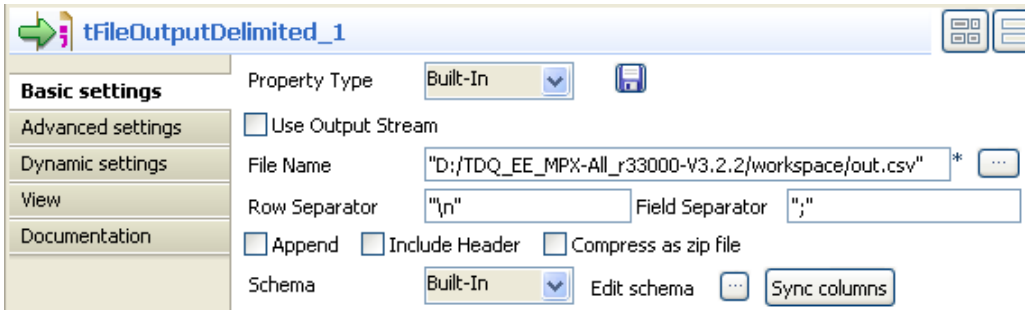
7. Click the **Advanced settings** tab of the **Component** view to set the advanced parameters of the component.



8. Deselect the **Extend Insert** check box which enables you to insert rows in batch, because this option is not compatible with the **Reject** link.

Configuring the output component

1. Double-click the **tFileOutputDelimited** component to set its properties in the **Component** view.



2. Click the **[...]** button next to the **File Name** field to fill in the path and name of the output file. Click the **Sync columns** button to retrieve the schema of the previous component.

Job execution

Save your Job and press **F6** to execute it.

1	Griffith Paving	talend@apres	7	41	##########	2001-01-17 06:26	Data truncation: Data too long
5	Terrinni & Sor	770 Exmoor F	5	9	##########	1982-04-19 10:26	Data truncation: Data too long
10	Elle Hypnosis	2032 Northbro	1	7	##########	1975-06-10 20:20	Data truncation: Data too long
15	Darcy Frame	1633 McGove	21	28	##########	2001-01-07 20:40	Data truncation: Data too long
19	Salt & Peppe	965 Marion P	44	33	##########	2006-01-24 10:53	Data truncation: Data too long
31	Arthur M. Fei	1220 Heather	37	24	##########	1974-07-24 20:48	Data truncation: Data too long
36	Got It All Ele	629 Kincaid A	22	28	##########	1993-10-22 15:26	Data truncation: Data too long
53	Crossroads C	2267 Anderso	5	14	##########	1977-10-24 17:53	Data truncation: Data too long
54	Jay's Cabinet	2024 St.John	48	48	2006-02-29 0(	2001-01-10 07:48	Data truncation: Data too long
57	Lee Stairwork	2644 Princeto	41	46	##########	1992-04-15 09:41	Data truncation: Data too long
58	Lambert & Lo	662 Lyons Ci	45	17	##########	2002-11-22 04:26	Data truncation: Data too long
70	janice Mann A	3181 Barkwo	28	35	##########	1983-04-20 15:20	Data truncation: Data too long
72	Michael Mont	1077 Court A	9	9	##########	2000-10-07 00:20	Data truncation: Data too long
83	Dean Barnett	2922 Twin Oa	18	46	##########	2006-03-23 10:14	Data truncation: Data too long
96	Doerner's Aw	616 Cobblest	26	28	##########	1993-05-03 04:33	Data truncation: Data too long
##	Auto Interior (	844 Spruce S	48	40	##########	2006-04-25 13:14	Data truncation: Data too long
##	First National	465 Stratford	5	41	##########	2003-09-23 01:54	Data truncation: Data too long
##	Terrinni & Sor	1859 Green E	12	44	##########	1986-12-26 05:48	Data truncation: Data too long

The data in error are sent to the delimited file, as well as the error type met. Here, we have: **Data truncation**.

tAmazonMysqlRollback

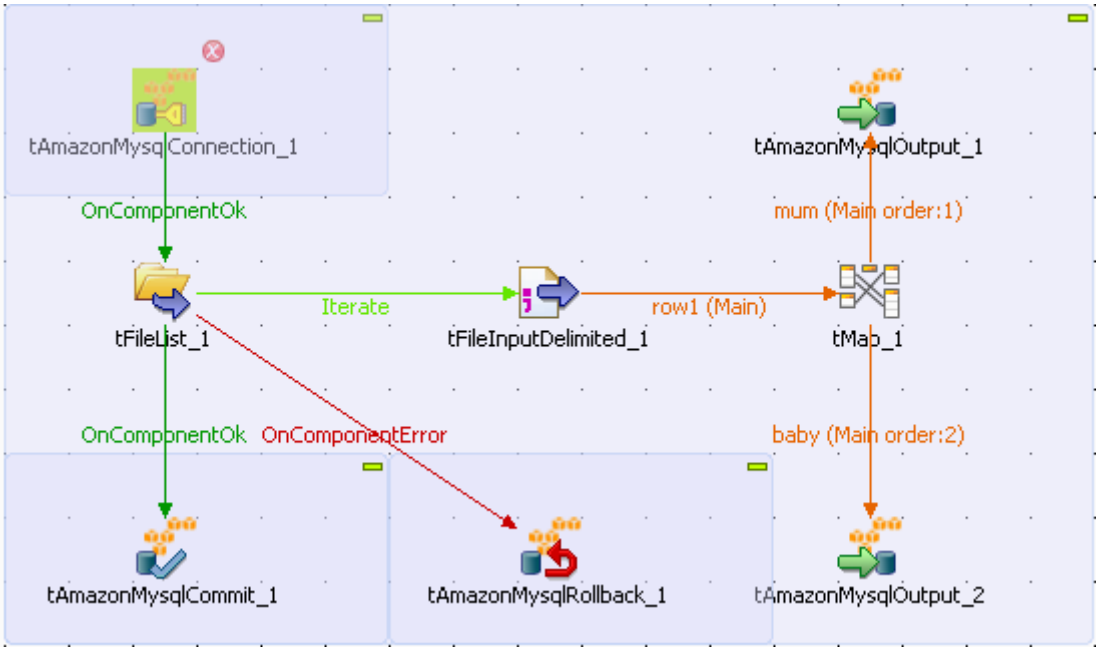
tAmazonMysqlRollback properties

This component is closely related to **tAmazonMysqlCommit** and **tAmazonMysqlConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Cloud/AmazonRDS/Mysql	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tAmazonMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonMysql components, especially with **tAmazonMysqlConnection** and **tAmazonMysqlCommit** components.	
Limitation	n/a	

Scenario: Rollback from inserting data in mother/daughter tables

Based on section *Scenario: Inserting data in mother/daughter tables*, insert a rollback function in order to prevent unwanted commit.



1. Drag and drop a **tAmazonMysqlRollback** to the design workspace and connect it to the Start component.

2. Set the Rollback unique field on the relevant DB connection.

This complementary element to the Job ensures that the transaction will not be partly committed.

tAmazonMysqlRow

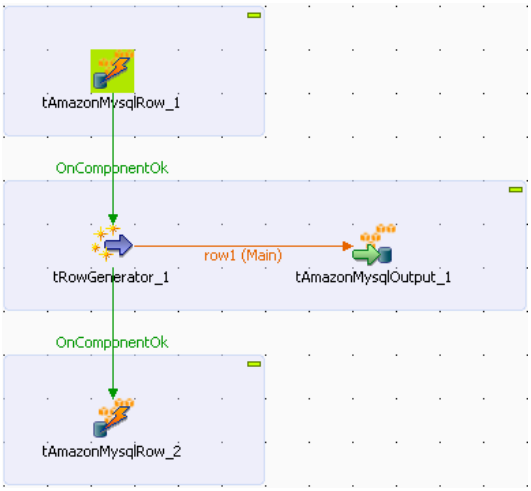
tAmazonMysqlRow properties

Component family	Cloud/Amazon/MySQL	
Function	**tAmazonMysqlRow** is the specific component for this database query. It executes the SQL query stated in the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAmazonMysqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	MySQL 5 is available.
	Use an existing connection	Select this check box and click the relevant **tAmazonMysqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder

	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the **Query** field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query in a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Scenario 1: Removing and regenerating a MySQL table index

This scenario describes a four-component job that removes a table index, applies a select insert action onto a table then regenerates the index.



Setting up the Job

1. Select and drop the following components onto the design workspace: **tAmazonMysqlRow** (x2), **tRowGenerator**, and **tAmazonMysqlOutput**.
2. Connect **tRowGenerator** to **tAmazonMysqlOutput**.
3. Using a **OnComponentOk** connections, link the first **tAmazonMysqlRow** to **tRowGenerator** and **tRowGenerator** to the second **tAmazonMysqlRow**.

Configuring the tAmazonMysqlRow component

1. Select the **tAmazonMysqlRow** to fill in the DB **Basic settings**.
2. Fill in the database connection details in the corresponding fields.
3. Propagate the properties and schema details onto the other components of the Job.
4. Type in the following SQL statement to alter the database entries: drop index <index_name> on <table_name>
5. Select the second **tAmazonMysqlRow** component, check the DB properties and schema.
6. Type in the SQL statement to recreate an index on the table using the following statement: create index <index_name> on <table_name> (<column_name>)

   The **tRowGenerator** component is used to generate automatically the columns to be added to the DB output table defined.

Configuring the output component

1. Select the **tAmazonMysqlOutput** component and fill in the DB connection properties>. The table to be fed is named: *comprehensive*.
2. The schema should be automatically inherited from the data flow coming from the **tRowGenerator**. Edit the schema to check its structure and check that it corresponds to the schema expected on the DB table specified.
3. The **Action on table** is *None* and the **Action on data** is *Insert*.

Job execution

Press **F6** to run the job.

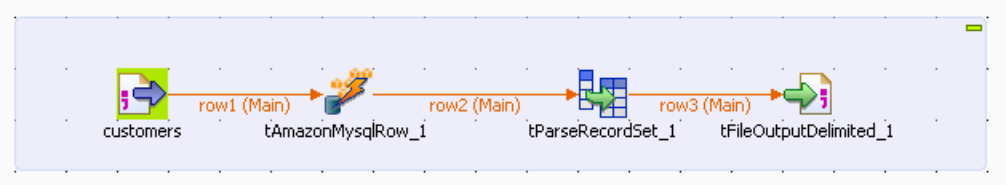
If you manage to watch the action on DB data, you can notice that the index is dropped at the start of the job and recreated at the end of the insert action.

Related topics: section *tDBSQLRow properties*.

Scenario 2: Using PreparedStatement objects to query data

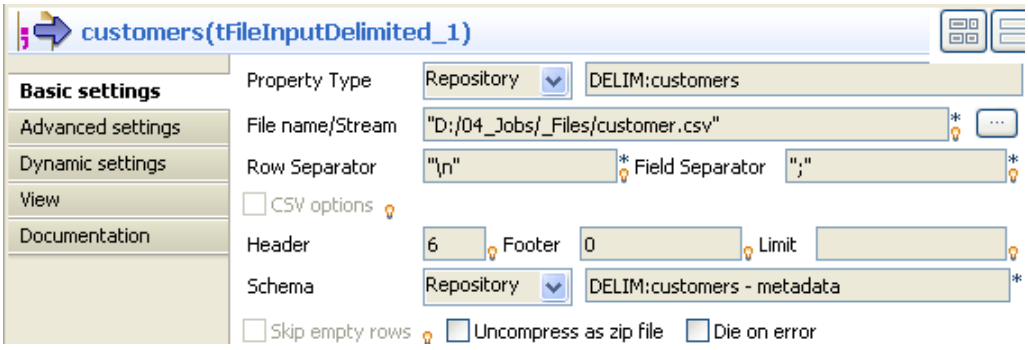
This scenario describes a four component job which allows you to link a table column with a client file. The MySQL table contains a list of all the American States along with the State ID, while the file contains the customer information including the ID of the State in which they live. We want to retrieve the name of the State for each client, using an SQL query. In order to process a large volume of data quickly, we use a PreparedStatement object which means that the query is executed only once rather than against each row in turn. Then each row is sentas a parameter.

For this scenario, we use a file and a database for which we have already stored the connection and properties in the Rerpository metadata. For further information concerning the creation of metadata in delimited files, the creation of database connection metadata and the usage of metadata, see *Talend Open Studio for Big Data User Guide*.



Configuring the input component

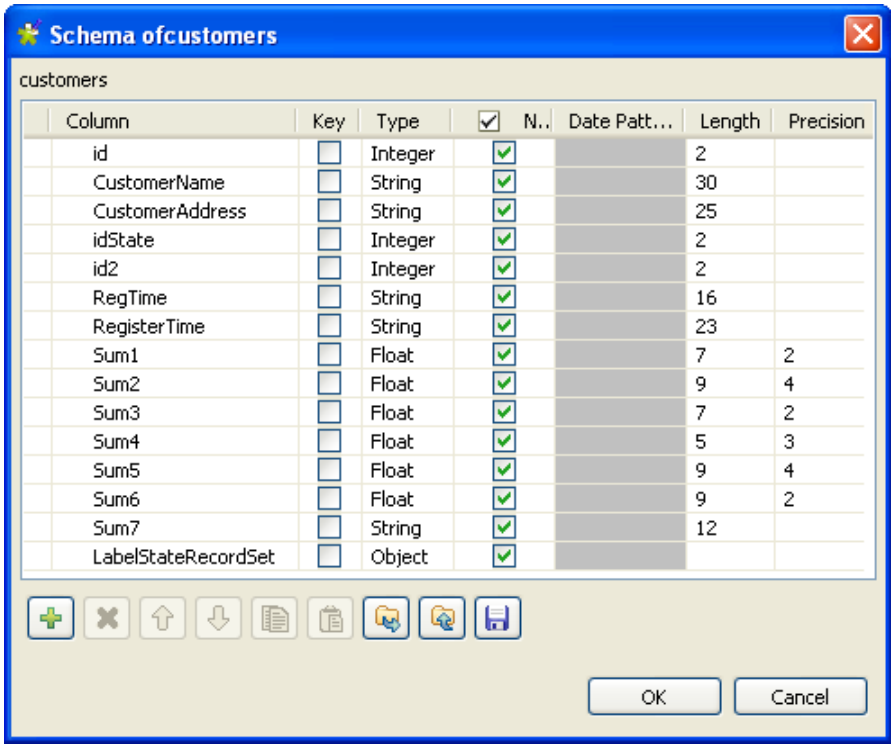
1. Drop a **tFileInputDelimited** component from the **Palette** onto the design workspace, and double-click it to open its **Basic settings** view.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

2. Define the path to the input file, the row speparator, the field separator, the header, and the footer in the corresponding fields.

3. Click on **[...]** next to the **Edit schema** field to add a column into which the name of the State will be inserted.

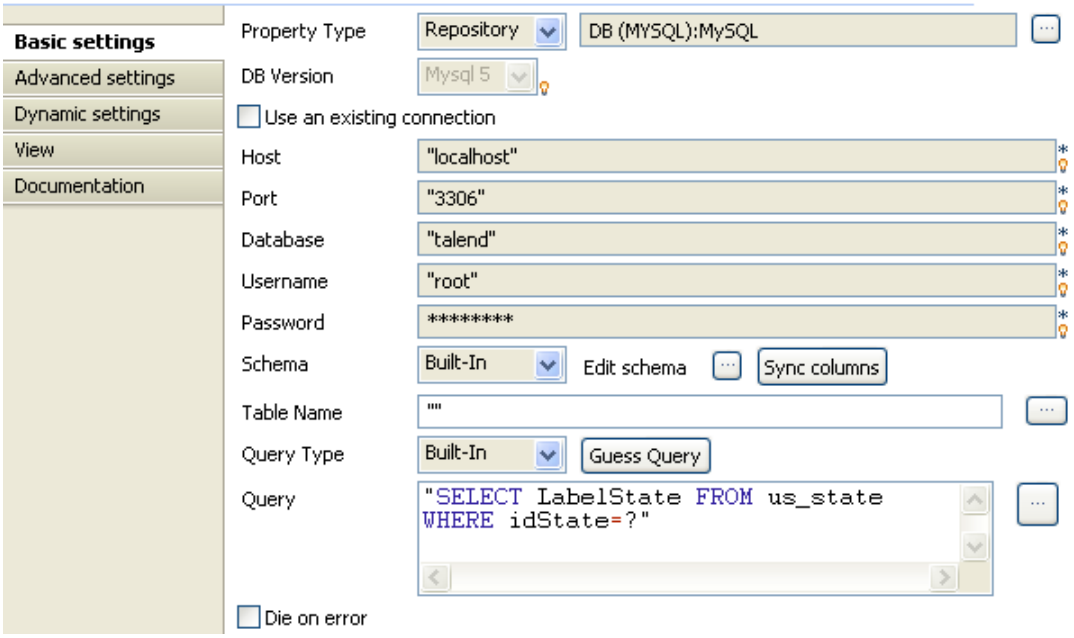


4. Click on the **[+]** button to add a column to the schema. Rename this column *LabelStateRecordSet* and select **Object** from the **Type** list. Click **OK** to save your modifications.

5. From the **Palette**, select the **tAmazonMysqlRow**, **tParseRecordSet** and **tFileOutputDelimited** components and drop them onto the workspace. Connect the four components using **Row** > **Main** type links.

Setting up the DB connection

1. Double-click **tAmazonMysqlRow** to set its properties in the **Basic settings** tab of the **Component** view.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

2. Fill in the database connection details in the corresponding fields.

3. Click the **Sync columns** button to retrieve the schemma from the preceding component.

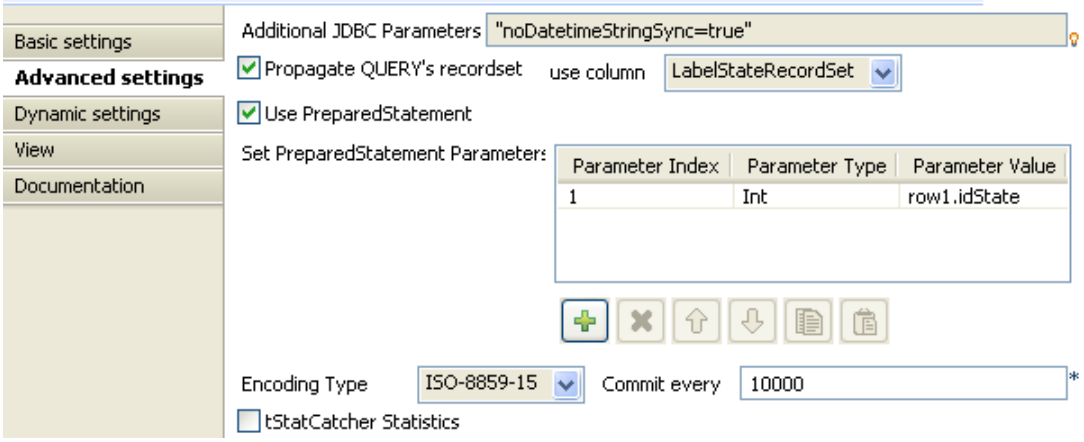
4. In the **Query** field, enter the SQL query you want to use. Here, we want to retrieve the names of the American States from the *LabelState* column of the MySQL table, *us_state*:

```
"SELECT LabelState
FROM us_state WHERE idState=?"
```

The question mark, "?", represents the parameter to be set in the **Advanced settings** tab.

Configuring the Advanced settings of tAmazonMysqlRow

1. Click **Advanced settings** to set the component's advanced properties.



2. Select the **Propagate QUERY's recordset** check box and select the *LabelStateRecordSet* column from the **use column** list to insert the query results in that column.

3. Select the **Use PreparedStatement** check box and define the parameter used in the query in the **Set PreparedStatement Parameters** table. Click on the **[+]** button to add a parameter.

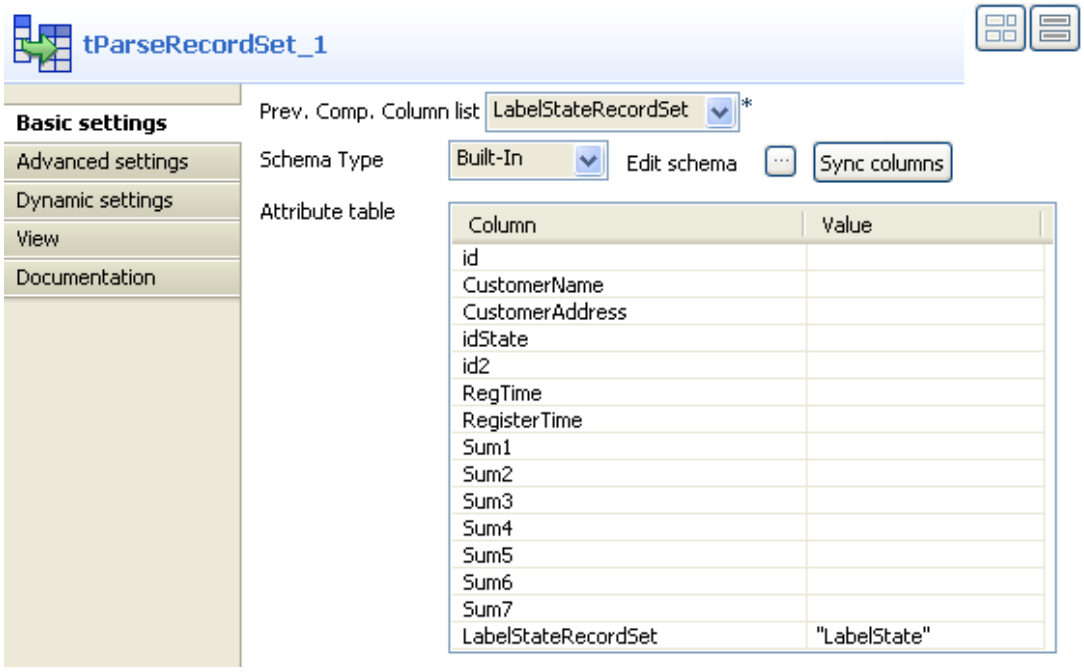
4. In the **Parameter Index** cell, enter the parameter position in the SQL instruction. Enter "*1*" as we are only using one parameter in this example.

5. In the **Parameter Type** cell, enter the type of parameter. Here, the parameter is a whole number, hence, select **Int** from the list.

6. In the **Parameter Value** cell, enter the parameter value. Here, we want to retrieve the name of the State based on the State ID for every client in the input file. Hence, enter "`row1.idState`".

Configuring the tParseRecordSet component

1. Double-click **tParseRecordSet** to set its properties in the **Basic settings** tab of the **Component** view.



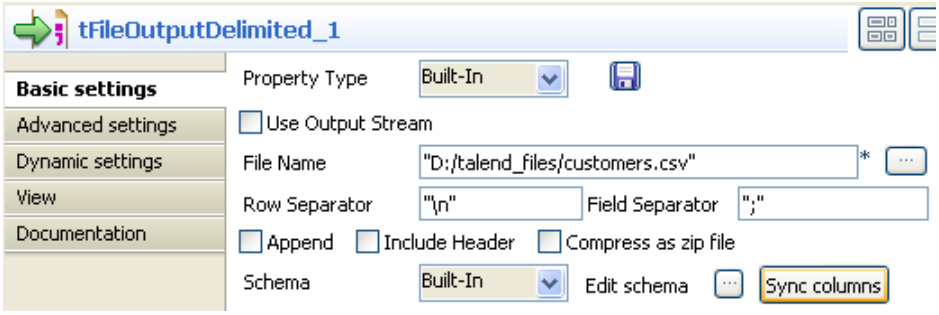
2. From the **Prev. Comp. Column list**, select the preceding components column for analysis. In this example, select *LabelStateRecordSet*.

3. Click on the **Sync columns** button to retrieve the schema from the preceding component. The **Attribute table** is automatically completed with the schema columns.

4. In the **Attribute table**, in the **Value** field which corresponds to the *LabelStateRecordSet*, enter the name of the column containing the State names to be retrieved and matched with each client, within double quotation marks. In this example, enter "*LabelState*".

Configuring the output component

1. Double-click **tFileOutputDelimited** to set its properties in the **Basic settings** tab of the **Component** view.



2. In the **File Name** field, enter the access path and name of the output file. Click **Sync columns** to retrieve the schema from the preceding component.

Job execution

Save your Job and press **F6** to run it.

customers.csv

	A	B	C	D	E	F	G	H	O
1	1	Griffith Paving	talend@apres	7	41	03/11/2006 09:20	2001-01-17 06:	67852.0	Connecticut
2	2	Bill's Dive Sh	511 Maple Av	35	5	19/11/2004 15:48	2002-06-07 09:	88792.0	Ohio
3	3	Childress Chi	662 Lyons Ci	1	28	16/02/2005 08:27	1990-04-01 21:	35340.0	Alabama
4	4	Facelift Kitch	220 Vine Ave	41	15	22/08/2002 09:55	1972-04-23 18:	6097.0	South Dakota
5	5	Terrinni & So	770 Exmoor F	5	9	28/06/2001 09:15	1982-04-19 10:	5146.0	California
6	6	Kermit the Pe	1860 Parksid	28	15	17/08/2003 10:07	2006-05-27 17:	16087.0	Nevada
7	7	Tub's Furnitur	807 Old Trail	15	9	27/08/2000 03:13	1970-03-27 23:	53216.0	Iowa
8	8	Toggle & Mye	618 Sheriden	9	15	24/03/2006 23:07	2005-08-02 01:	74168.0	Florida
9	9	Childress Chi	788 Tennyson	12	33	10/09/2001 06:33	1994-05-03 11:	92176.0	Idaho
10	10	Elle Hypnosis	2032 Northbr	1	7	11/01/1977 03:07	1975-06-10 20:	48498.0	Alabama

A column containing the name of the American State corrresponding to each client is added to the file.

Related scenarios

For a related scenario, see:

- section *Scenario 3: Combining two flows for selective output*

tAmazonOracleClose

tAmazonOracleClose properties

Function	**tAmazonOracleClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tAmazonOracleConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonOracle components, especially with **tAmazonOracleConnection** and **tAmazonOracleCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tAmazonOracleCommit

tAmazonOracleCommit Properties

This component is closely related to **tAmazonOracleConnection** and **tAmazonOracleRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Cloud/AmazonRDS/Oracle	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tAmazonOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tAmazonOracleCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonOracle components, especially with **tAmazonOracleConnection** and **tAmazonOracleRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAmazonOracleConnection** and **tAmazonOracleRollback**. It usually doesn't make much sense to use one of these without using a **tAmazonOracleConnection** component to open a connection for the current transaction.

For **tAmazonOracleCommit** related scenario, see section *tMysqlConnection*

tAmazonOracleConnection

tAmazonOracleConnection Properties

This component is closely related to **tAmazonOracleCommit** and **tAmazonOracleRollback**. It usually doesn't make much sense to use one of these without using a **tAmazonOracleConnection** component to open a connection for the current transaction.

Component family	Cloud/AmazonRDS/Oracle	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Connection type	Drop-down list of available drivers: **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Oracle 11-5 is available.
	Use tns file	Select this check box to use the metadata of a context included in a tns file. One tns file may have many contexts. **TNS File**: Enter the path to the tns file manually or browse to the file by clicking the three-dot button next to the filed. **Select a DB Connection in Tns File**: Click the three-dot button to display all the contexts held in the tns file and select the desired one.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating. You can set the encoding parameters through this field.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.

Usage	This component is to be used along with AmazonOracle components, especially with **tAmazonOracleCommit** and **tAmazonOracleRollback** components.
Limitation	n/a

Related scenario

This component is closely related to **tAmazonOracleCommit** and **tAmazonOracleRollback**. It usually doesn't make much sense to use one of these without using a **tAmazonOracleConnection** component to open a connection for the current transaction.

For **tAmazonOracleConnection** related scenario, see section *tMysqlConnection*

tAmazonOracleInput

tAmazonOracleInput properties

Component family	Cloud/AmazonRDS/Oracle	
Function	**tAmazonOracleInput** reads a database and extracts fields based on a query.	
Purpose	**tAmazonOracleInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Connection type	Drop-down list of available drivers: **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Select the Oracle version in use.
	Use an existing connection	Select this check box when using a configured **tAmazonOracleConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Oracle schema	Oracle schema name.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Database table name.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
Usage	This component covers all possible SQL queries for Oracle databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tAmazonOracleOutput

tAmazonOracleOutput properties

Component family	Cloud/AmazonRDS/Oracle	
Function	**tAmazonOracleOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAmazonOracleOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a **tAmazonOracleConnection** component. When you deselect it, a check box appears (selected by default and followed by a field) in the **Advanced settings**, **Batch Size**, which enables you to define the number of lines in each processed batch. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers: **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Select the Oracle version in use.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Oracle schema	Name of the Oracle schema.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again.

		Create a table: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Override any existing NLS_LANG environment variable	Select this check box to override variables already set for a NLS language environment.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.

		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Use Hint Options	Select this check box to activate the hint configuration area which helps you optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */.` - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	Convert columns and table to uppercase	Select this check box to set the names of columns and table in upper case.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch. *This option is available only when you do not **Use an existing connection** in **Basic settings**.*
	Support null in "SQL WHERE" statement	Select this check box to validate null in "SQL WHERE" statement.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Oracle database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For such an example, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tAmazonOracleOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tAmazonOracleRollback

tAmazonOracleRollback properties

This component is closely related to **tAmazonOracleCommit** and **tAmazonOracleConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Cloud/AmazonRDS/Oracle	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tAmazonOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AmazonOracle components, especially with **tAmazonOracleConnection** and **tAmazonOracleCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAmazonOracleConnection** and **tAmazonOracleCommit**. It usually doesn't make much sense to use one of these without using a **tAmazonOracleConnection** component to open a connection for the current transaction.

For **tAmazonOracleRollback** related scenario, see section *tMysqlRollback*.

tAmazonOracleRow

tAmazonOracleRow properties

Component family	Cloud/AmazonRDS/Oracle	
Function	**tAmazonOracleRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAmazonOracleRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tAmazonOracleConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tMarketoInput

tMarketoInput belongs to two component families: Business and Cloud. For more information on it, see section *tMarketoInput*.

tMarketoListOperation

tMarketoListOperation belongs to two component families: Business and Cloud. For more information on it, see section *tMarketoListOperation*.

tMarketoOutput

tMarketoOutput belongs to two component families: Business and Cloud. For more information on it, see section *tMarketoOutput*.

tSalesforceBulkExec

tSalesforceBulkExec belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceBulkExec*.

tSalesforceConnection

tSalesforceConnection belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceConnection*.

tSalesforceGetDeleted

tSalesforceGetDeleted belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceGetDeleted*.

tSalesforceGetServerTimestamp

tDB2SCD belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceGetServerTimestamp*.

tSalesforceGetUpdated

tSalesforceGetUpdated belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceGetUpdated*.

tSalesforceInput

tSalesforceInput belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceInput*.

tSalesforceOutput

tSalesforceOutput belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceOutput*.

tSalesforceOutputBulk

tSalesforceOutputBulk belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceOutputBulk*.

tSalesforceOutputBulkExec

tSalesforceOutputBulkExec belongs to two component families: Business and Cloud. For more information on it, see section *tSalesforceOutputBulkExec*.

tSugarCRMInput

tSugarCRMInput belongs to two component families: Business and Cloud. For more information on it, see section *tSugarCRMInput*.

tSugarCRMOutput

tSugarCRMOutput belongs to two component families: Business and Cloud. For more information on it, see section *tSugarCRMOutput*.

Custom Code components

This chapter details the major components which belong to the **Custom Code** family in the **Palette** of the **Integration** perspective of the **Talend** Studio.

The **Custom Code** components enable you to create codes for specific needs, quickly and efficiently.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tGroovy

tGroovy properties

Component Family	Custom Code	
Function	**tGroovy** allows you to enter customized code which you can integrate in the Talend programme. The code is run only once.	
Purpose	**tGroovy** broadens the functionality if the Talend Job, using the Groovy language which is a simplified Java syntax.	
Basic settings	*Groovy Script*	Enter the Groovy code youo want to run.
	Variables	This table has two columns. **Name**: Name of the variable called in the code.. **Value**: Value associated with the variable.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at component level.
Usage	This component can be used alone or as a subjob along with one other component.	
Limitation	Knowledge of the Groovy language is required.	

Related Scenarios

- For a scenario using the Groovy code, see section *Scenario: Calling a file which contains Groovy code*.
- For a functional example, see section *Scenario: Printing out a variable content*

tGroovyFile

tGroovyFile properties

Component Family	Custom Code	
Function	**tGroovyFile** allows you to call an existing Groovy script.	
Purpose	**tGroovyFile** broadens the functionaility of Talend Jobs using the Groovy language which is a simplified Java syntax.	
Basic settings	*Groovy File*	Name and path of the file containing the Groovy code.
	Variables	This table contains two columns. **Name**: Name of the variable called in the code. **Value**: Value associated with this variable.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at component level.
Usage	This component can be used alone or as a sub-job along with another component.	
Limitation	Knowledge of the Groovy language is required.	

Scenario: Calling a file which contains Groovy code

This scenario uses **tGroovyFile**, on its own. The Job calls a file containing Groovy code in order to display the file information in the **Console**. Below, is an example of the information displayed:

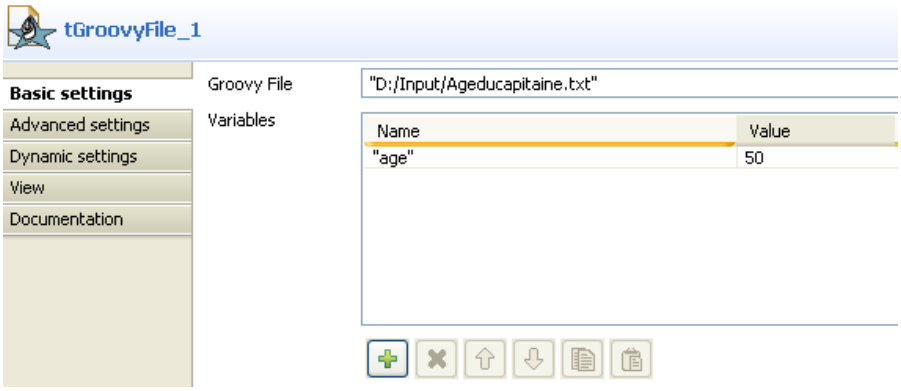
Setting up the Job

Open the **Custom_Code** folder in the **Palette** and drop a **tGroovyFile** component onto the workspace.

Configuring the tGroovyFile component

1. Double-click the component to display the **Component** view.
2. In the **Groovy File** field, enter the path to the file containing the Groovy code, or browse to the file in your directory.
3. In the **Variables** table, add a line by clicking the **[+]** button.

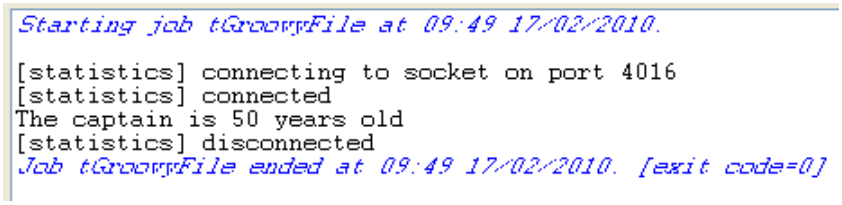
4. In the **Name** column, enter *"age"*, then in the Value column, enter *50, as in the screenshot*.



Job execution

Press **F6** to save and run the Job.

The **Console** displays the information contained in the input file, to which the variable result is added.



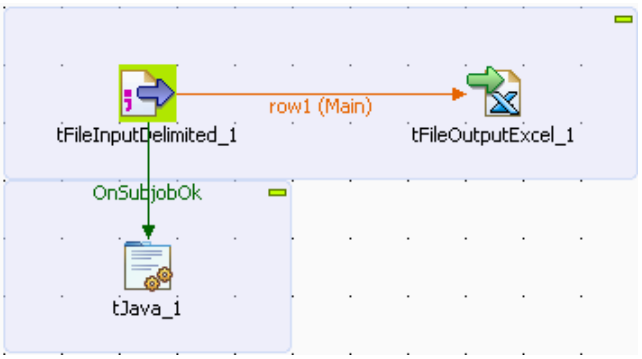
tJava

tJava properties

Component family	Custom Code	
Function	**tJava** enables you to enter personalized code in order to integrate it in **Talend** program. You can execute this code only once.	
Purpose	**tJava** makes it possible to extend the functionalities of a **Talend** Job through using Java commands.	
Basic settings	*Code*	Type in the Java code you want to execute according to the task you need to perform. For further information about Java functions syntax specific to **Talend**, see *Talend Open Studio for Big Data* Online Help (Help Contents > Developer Guide > API Reference). For a complete Java reference, check http://docs.oracle.com/javaee/6/api/
Advanced settings	*Import*	Enter the Java code that helps to import, if necessary, external libraries used in the **Main code** box of the **Basic settings** view.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as a one-component subjob.	
Limitation	You should know Java language.	

Scenario: Printing out a variable content

The following scenario is a simple demo of the extended application of the **tJava** component. The Job aims at printing out the number of lines being processed using a Java command and the global variable provided in *Talend Open Studio for Big Data*.



Setting up the Job

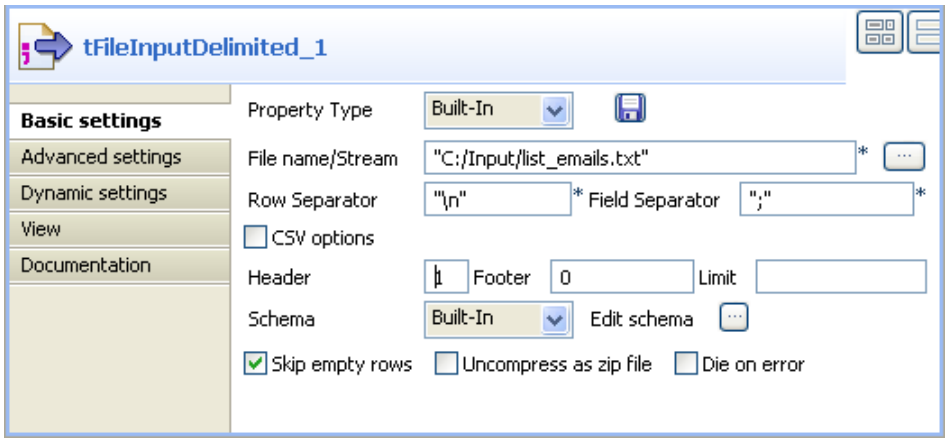
1. Select and drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tFileOutputExcel**, **tJava**.

2. Connect the **tFileInputDelimited** to the **tFileOutputExcel** using a **Row Main** connection. The content from a delimited txt file will be passed on through the connection to an xls-type of file without further transformation.

3. Then connect the **tFileInputDelimited** component to the **tJava** component using a **Trigger** > **On Subjob Ok** link. This link sets a sequence ordering **tJava** to be executed at the end of the main process.

Configuring the input component

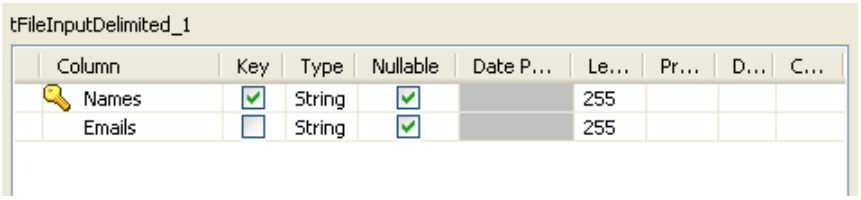
1. Set the **Basic settings** of the **tFileInputDelimited** component.



2. Define the path to the input file in the **File name** field.

   The input file used in this example is a simple text file made of two columns: *Names* and their respective *Emails*.

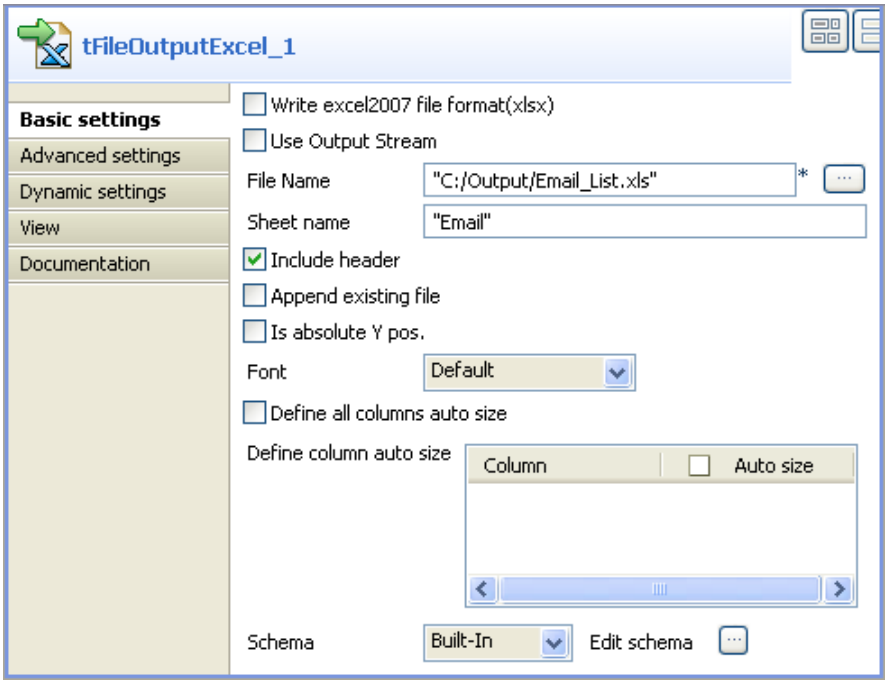
3. Click the **Edit Schema** button, and set the two-column schema. Then click **OK** to close the dialog box.



4. When prompted, click **OK** to accept the propagation, so that the **tFileOutputExcel** component gets automatically set with the input schema.

Configuring the output component

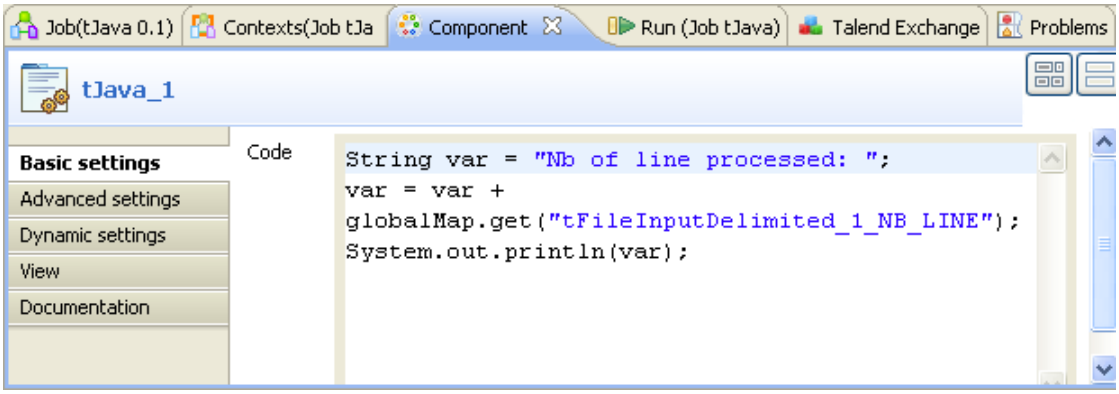
Set the output file to receive the input content without changes. If the file does not exist already, it will get created.



In this example, the **Sheet name** is *Email* and the **Include Header** box is selected.

Configuring the tJava component

1. Then select the **tJava** component to set the Java command to execute.



2. In the **Code** area, type in the following command:

```
String var = "Nb of line processed: ";
var = var + globalMap.get("tFileInputDelimited_1_NB_LINE");
System.out.println(var);
```

In this use case, we use the *NB_Line* variable. To access the global variable list, press Ctrl + Space bar on your keyboard and select the relevant global parameter.

Job execution

Save your Job and press **F6** to execute it.

```
Starting job JavaDb at 13:53 20/08/2007.
Nb of line processed: 4
Job JavaDb ended at 13:53 20/08/2007. [exit code=0]
```

The content gets passed on to the Excel file defined and the Number of lines processed are displayed on the **Run** console.

tJavaFlex

tJavaFlex properties

Component family	Custom Code	
Function	**tJavaFlex** enables you to enter personalized code in order to integrate it in **Talend** program. With **tJavaFlex**, you can enter the three java-code parts (start, main and end) that constitute a kind of component dedicated to do a desired operation.	
Objective	**tJava** makes it possible to extend the functionalities of a **Talend** Job through using Java commands.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Data Auto Propagate	Select this check box to automatically propagate the data to the component that follows.
	Start code	Enter the Java code that will be called during the initialization phase.
	Main code	Enter the Java code to be applied for each line in the data flow.
	End code	Enter the Java code that will be called during the closing phase.
Advanced settings	*Import*	Enter the Java code that helps to import, if necessary, external libraries used in the **Main code** box of the **Basic settings** view.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a job level as well as at each component level.
Usage	You can use this component as a start, intermediate or output component. You can as well use it as a one-component subjob.	
Limitation	You should know the Java language.	

Scenario 1: Generating data flow

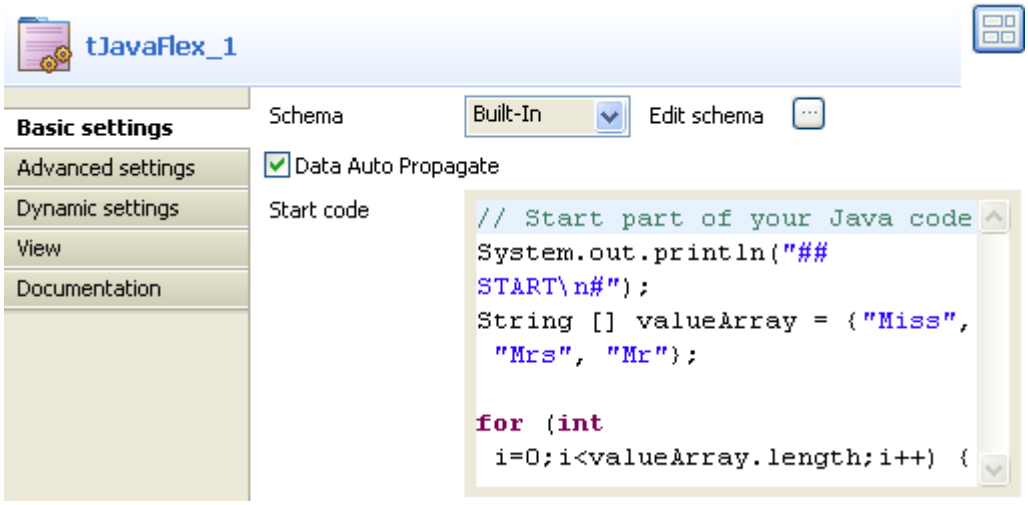
This scenario describes a two-components Job that generates a three-line data flow describing different personal titles (Miss, Mrs, and Mr) and displaying them on the console.

Setting up the Job

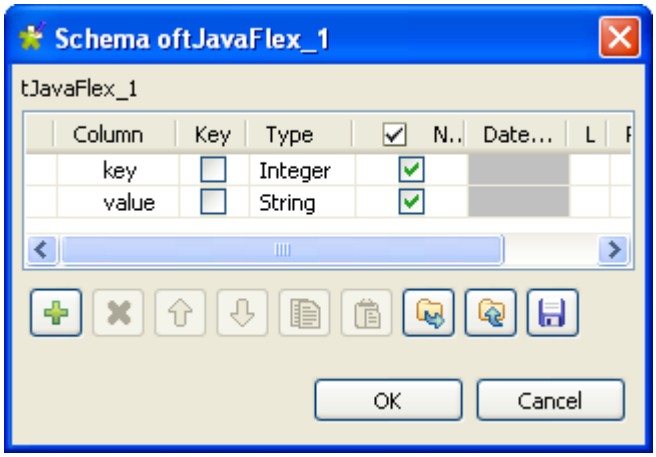
1. Drop **tJavaFlex** and **tLogRow** from the **Palette** onto the design workspace.
2. Connect the components together using a **Row** > **Main** link.

Configuring the tJavaFlex component

1. Double-click **tJavaFlex** to display its **Basic settings** view and define its properties.



2. Click the three-dot button next to **Edit schema** to open the corresponding dialog box where you can define the data structure to pass to the component that follows.



3. Click the **[+]** button to add two columns: *key* and *value* and then set their types to **Integer** and **String** respectively.
4. Click **OK** to validate your changes and close the dialog box.
5. In the **Basic settings** view of **tJavaFlex**, select the **Data Auto Propagate** check box to automatically propagate data to the component that follows.

   In this example, we do not want to do any transformation on the retrieved data.
6. In the **Start code** field, enter the code to be executed in the initialization phase.

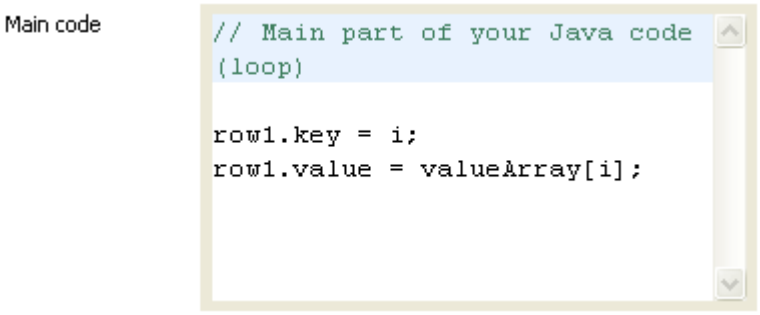
   In this example, the code indicates the initialization of **tJavaFlex** by displaying the START message and sets up the loop and the variables to be used afterwards in the Java code:

```
System.out.println("## START\n#");
String [] valueArray = {"Miss", "Mrs", "Mr"};

for (int i=0;i<valueArray.length;i++) {
```

7. In the **Main code** field, enter the code you want to apply on each of the data rows.

   In this example, we want to display each key with its value:

```
row1.key = i;
row1.value = valueArray[i];
```

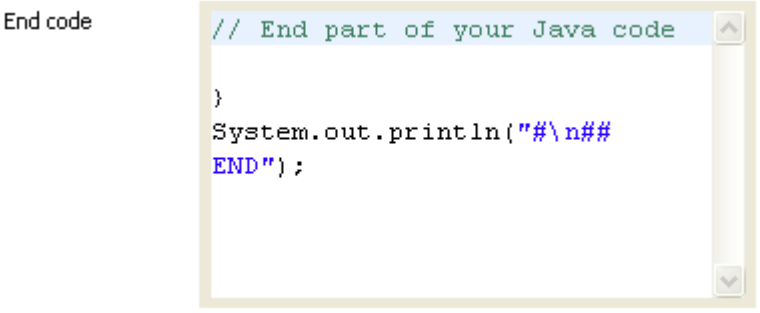


*In the **Main code** field, "row1" corresponds to the name of the link that comes out of **tJavaFlex**. If you rename this link, you have to modify the code of this field accordingly.*

8. In the **End code** field, enter the code that will be executed in the closing phase.

   In this example, the brace (curly bracket) closes the loop and the code indicates the end of the execution of **tJavaFlex** by displaying the END message:

```
}
System.out.println("#\n## END");
```



9. If needed, double-click **tLogRow** and in its **Basic settings** view, click the **[...]** button next to **Edit schema** to make sure that the schema has been correctly propagated.

Saving and executing the Job

1. Save your Job by pressing **Ctrl+S**.

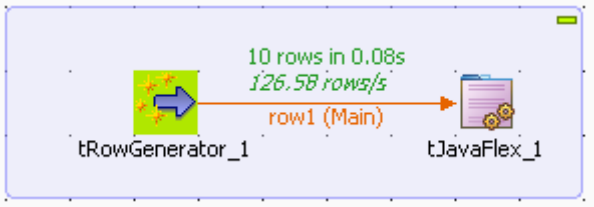
2. Execute the Job by pressing **F6** or clicking **Run** on the **Run** tab.

```
Starting job tJavaFlex_scenario1 at 14:49 02/09/2009.

## START
#
0|Miss
1|Mrs
2|Mr
#
## END
Job tJavaFlex_scenario1 ended at 14:49 02/09/2009. [exit code=0]
```

The three personal titles are displayed on the console along with their corresponding keys.

Scenario 2: Processing rows of data with tJavaFlex

This scenario describes a two-component Job that generates random data and then collects that data and does some transformation on it line by line using Java code through the **tJavaFlex** component.

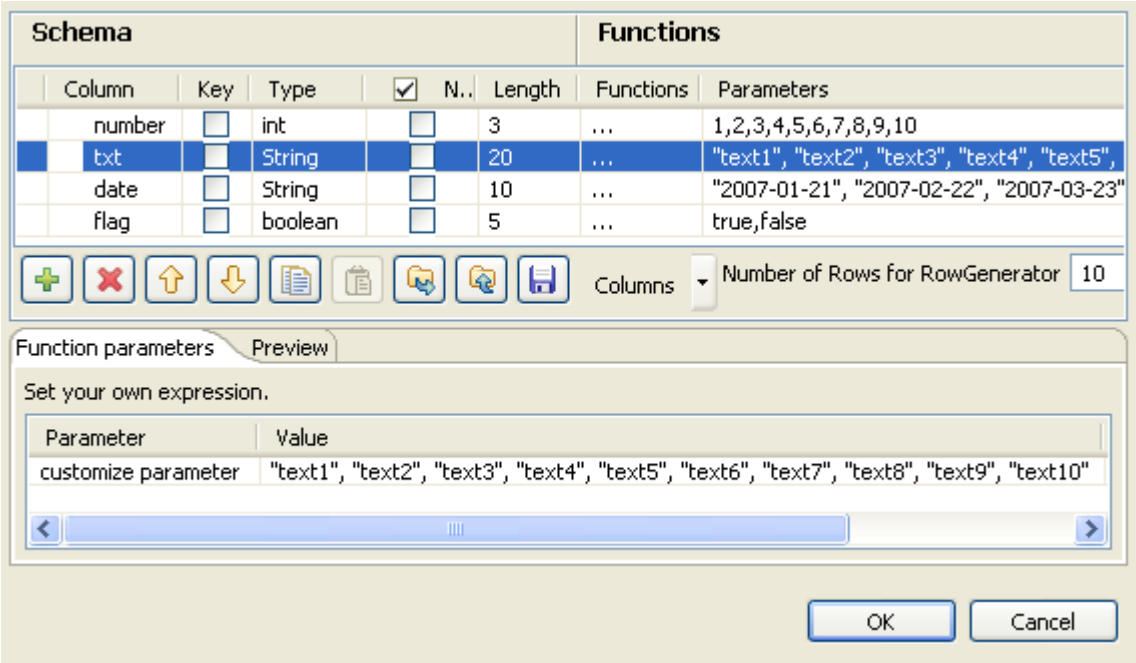


Setting up the Job

1. Drop **tRowGenerator** and **tJavaFlex** from the **Palette** onto the design workspace.
2. Connect the components together using a **Row Main** link.

Configuring the input component

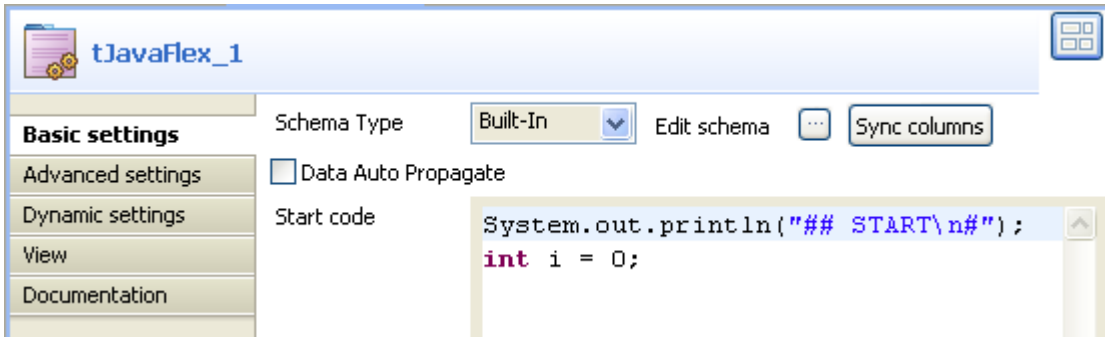
1. Double-click **tRowGenerator** to display its **Basic settings** view and the **[RowGenerator Editor]** dialog box where you can define the component properties.



2. Click the plus button to add four columns: *number*, *txt*, *date* and *flag*.
3. Define the schema and set the parameters to the four columns according to the above capture.
4. In the **Functions** column, select the three-dot function **[...]** for each of the defined columns.
5. In the **Parameters** column, enter 10 different parameters for each of the defined columns. These 10 parameters corresponds to the data that will be randomly generated when executing **tRowGenerator**.
6. Click **OK** to validate your changes and close the editor.

Configuring the tJavaFlex component

1. Double-click **tJavaFlex** to display its **Basic settings** view and define the components properties.



2. Click **Sync columns** to retrieve the schema from the preceding component.
3. In the **Start code** field, enter the code to be executed in the initialization phase.

   In this example, the code indicates the initialization of the **tJavaFlex** component by displaying the START message and defining the variable to be used afterwards in the Java code:

```
System.out.println("## START\n#");
```

```
int i = 0;
```

4. In the **Main code** field, enter the code to be applied on each line of data.

   In this example, we want to show the number of each line starting from 0 and then the number and the random text transformed to upper case and finally the random date set in the editor of **tRowGenerator**. Then, we create a condition to show if the status is **true** or **false** and we increment the number of the line:

```
System.out.print(" row" + i + ":");
System.out.print("# number:" + row1.number);
System.out.print (" | txt:" + row1.txt.toUpperCase());
System.out.print(" | date:" + row1.date);
if(row1.flag) System.out.println(" | flag: true");
else  System.out.println(" | flag: false");

i++;
```

Main code

```
System.out.print(" row" + i + ":");
System.out.print("# number:" +
 row1.number);
System.out.print (" | txt:" +
 row1.txt.toUpperCase());
System.out.print(" | date:" +
 row1.date);
if(row1.flag) System.out.println(" |
flag: true");
else  System.out.println(" | flag:
false");

i++;
```

*In the **Main code** field, "row1" corresponds to the name of the link that connects to **tJavaFlex**. If you rename this link, you have to modify the code.*

5. In the **End code** field, enter the code that will be executed in the closing phase.

   In this example, the code indicates the end of the execution of **tJavaFlex** by displaying the END message:

```
System.out.println("#\n## END");
```

End code

```
System.out.println("#\n## END");
```

Saving and executing the Job

1. Save your Job by pressing **Ctrl+S**.

2. Execute the Job by pressing **F6** or clicking **Run** on the **Run** tab.

```
Starting job tJavaFlex_scenario2 at 18:35 02/09/2009.

## START
#
 row0:# number:10 | txt:TEXT5 | date:2006-05-25 | flag: false
 row1:# number:7 | txt:TEXT7 | date:2006-06-26 | flag: true
 row2:# number:5 | txt:TEXT2 | date:2006-05-25 | flag: false
 row3:# number:6 | txt:TEXT1 | date:2005-08-28 | flag: true
 row4:# number:8 | txt:TEXT9 | date:2006-05-25 | flag: false
 row5:# number:9 | txt:TEXT1 | date:2007-01-21 | flag: false
 row6:# number:7 | txt:TEXT5 | date:2004-11-21 | flag: true
 row7:# number:9 | txt:TEXT4 | date:2005-08-29 | flag: true
 row8:# number:4 | txt:TEXT6 | date:2006-06-26 | flag: false
 row9:# number:3 | txt:TEXT10 | date:2006-05-25 | flag: false
#
## END
Job tJavaFlex_scenario2 ended at 18:35 02/09/2009. [exit
code=0]
```

The console displays the randomly generated data that was modified by the java command set through **tJavaFlex**.

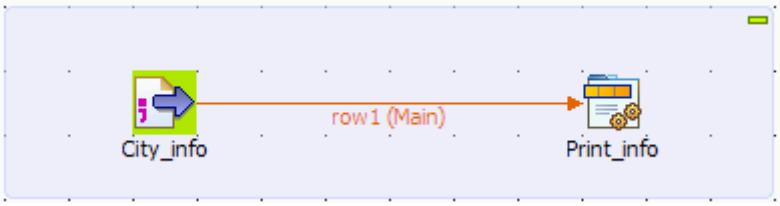
tJavaRow

tJavaRow properties

Component Family	Custom Code	
Function	**tJavaRow** allows you to enter customized code which you can integrate in a Talend programme. With **tJavaRow**, you can enter the Java code to be applied to each row of the flow.	
Purpose	**tJavaRow** allows you to broaden the functionality of **Talend** Jobs, using the Java language.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Code	Enter the Java code to be applied to each line of the data flow.
Advanced settings	*Import*	Enter the Java code required to import, if required, the external library used in the **Main code** field of the **Basic settings** tab.
	tStatCatcher Statistics	Select this check box to collect the log data at a component level..
Usage	This component is used as an intermediary between two other components. It must be linked to both an input and an output component.	
Limitation	Knowledge of Java language is necessary.	

Scenario: Transforming data line by line using tJavaRow

In this scenario, the information of a few cities read from an input delimited file is transformed using Java code through the **tJavaRow** component and printed on the console.



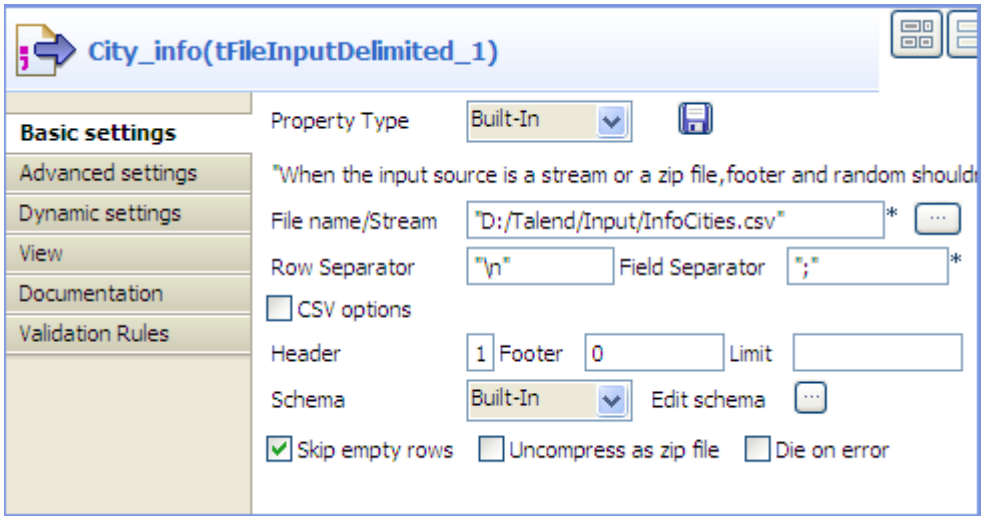
Setting up the Job

1. Drop a **tFileInputDelimited** component and a **tJavaRow** component from the **Palette** onto the design workspace, and label them to better identify their roles in the Job.

2. Connect the two components using a **Row** > **Main** connection.

Configuring the components

1. Double-click the **tFileInputDelimited** component to display its **Basic settings** view in the **Component** tab.

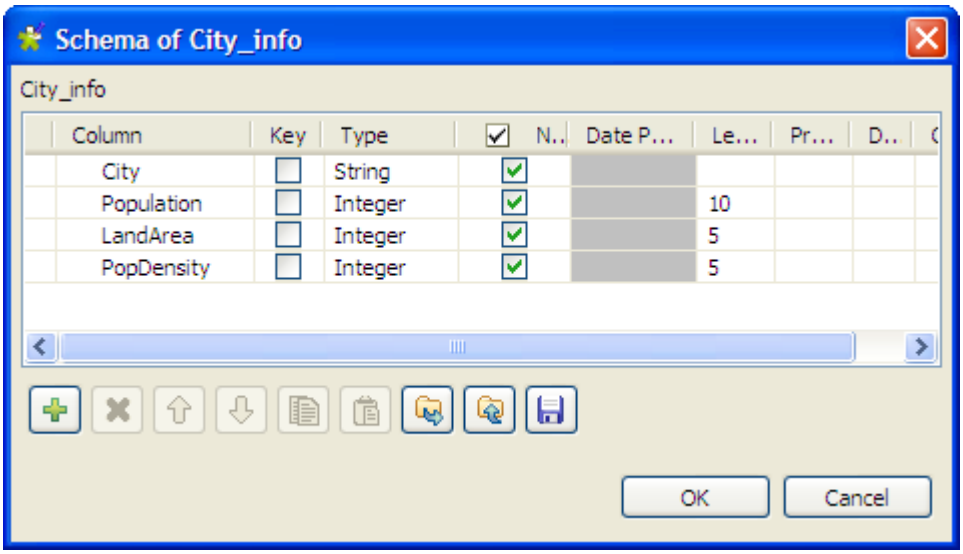


2. In the **File name/Stream** field, type in the path to the input file in double quotation marks, or browse to the path by clicking the **[...]** button, and define the first line of the file as the header.

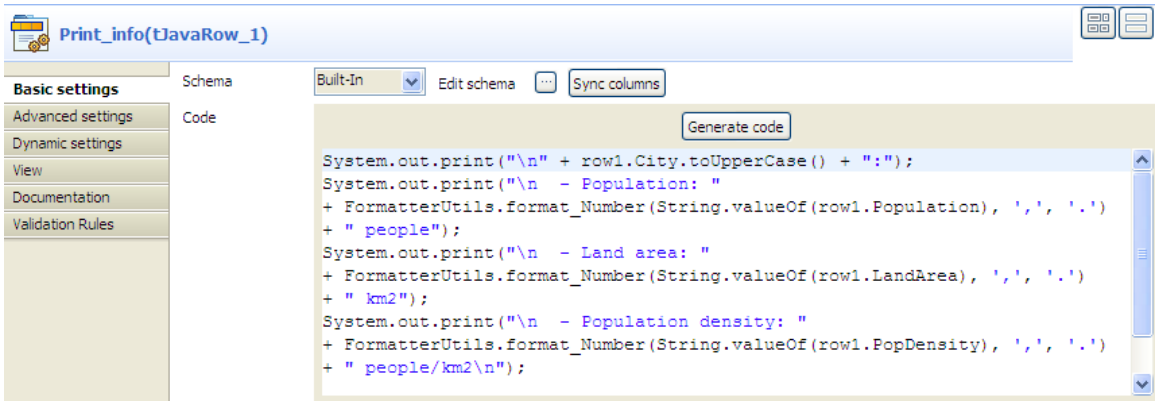
   In this example, the input file has the following content:

```
City;Population;LandArea;PopDensity
Beijing;10233000;1418;7620
Moscow;10452000;1081;9644
Seoul;10422000;605;17215
Tokyo;8731000;617;14151
New York;8310000;789;10452
```

3. Click the **[...]** button next to Edit schema to open the **[Schema]** dialog box, and define the data structure of the input file. Then, click **OK** to validate the schema setting and close the dialog box.



4. Double-click the **tJavaRow** component to display its **Basic settings** view in the **Component** tab.



5. Click **Sync columns** to make sure that the schema is correctly retrieved from the preceding component.

6. In the **Code** field, enter the code to be applied on each line of data based on the defined schema columns.

   In this example, we want to transform the city names to upper case, group digits of numbers larger than 1000 using the thousands separator for ease of reading, and print the data on the console:

```
System.out.print("\n" + row1.City.toUpperCase() + ":");
System.out.print("\n  - Population: "
+ FormatterUtils.format_Number(String.valueOf(row1.Population), ',', '.')
+ " people");
System.out.print("\n  - Land area: "
+ FormatterUtils.format_Number(String.valueOf(row1.LandArea), ',', '.')
+ " km2");
System.out.print("\n  - Population density: "
+ FormatterUtils.format_Number(String.valueOf(row1.PopDensity), ',', '.')
+ " people/km2\n");
```

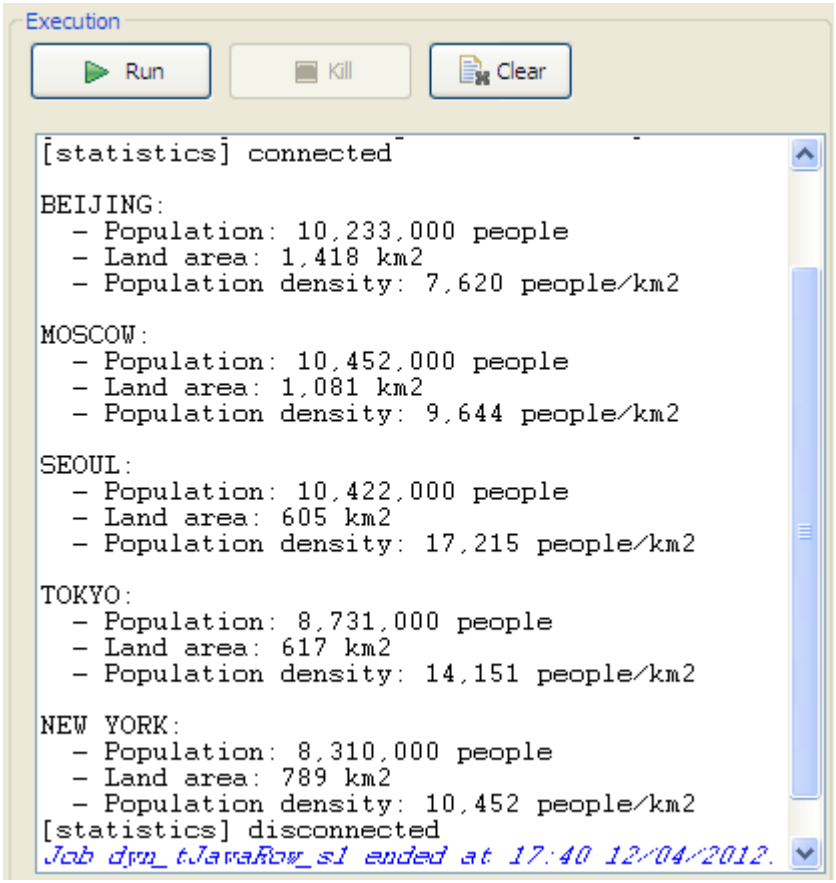
*In the **Code** field, "row1" refers to the name of the link that connects to **tJavaRow**. If you rename the link, you have to modify the code.*

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The city information is transformed by the Java code set through **tJavaRow** and displayed on the console.


Execution
Run
Kill
Clear
[statistics] connected
BEIJING:
- Population: 10,233,000 people
- Land area: 1,418 km2
- Population density: 7,620 people/km2
MOSCOW:
- Population: 10,452,000 people
- Land area: 1,081 km2
- Population density: 9,644 people/km2
SEOUL:
- Population: 10,422,000 people
- Land area: 605 km2
- Population density: 17,215 people/km2
TOKYO:
- Population: 8,731,000 people
- Land area: 617 km2
- Population density: 14,151 people/km2
NEW YORK:
- Population: 8,310,000 people
- Land area: 789 km2
- Population density: 10,452 people/km2
[statistics] disconnected
Job dpm_tJavaRow_s1 ended at 17:40 12/04/2012.

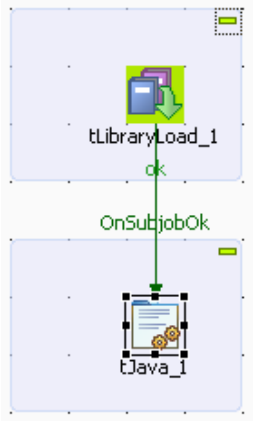
tLibraryLoad

tLibraryLoad properties

Famille de composant	Custom Code	
Function	**tLibraryLoad** allows you to import a library.	
Purpose	**tLibraryLoad** allows you to load useable Java libraries in a Job.	
Basic settings	*Library*	Select the library you want to import from the list, or click on the **[...]** button to browse to the library in your directory.
Advanced settings	*Dynamic Libs*	**Lib Paths**: Enter the access path to your library, between double quotation marks.
	Import	Enter the Java code required to import, if required, the external library used in the **Main code** field of the **Basic settings** tab.
	tStatCatcher Statistics	Select this check box to collect the log data at component level.
Usage	This component may be used alone, although it is more logical to use it as part of a Job.	
Limitation	n/a	

Scenario: Checking the format of an e-mail addressl

This scenario uses two components, a **tLibraryLoad** and a **tJava**. The goal of this scenario is to check the format of an e-mail address and verify whether the format is valid or not.



Setting up the Job

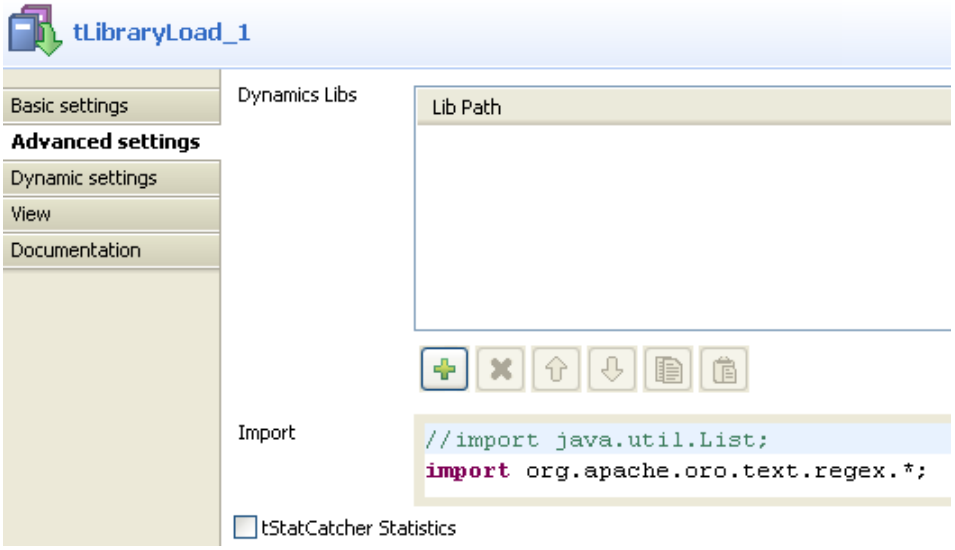
1. In the **Palette**, open the **Custom_Code** folder, and slide a **tLibraryLoad** and **tJava** component onto the workspace.

2. Connect **tLibraryLoad** to **tJava** using a **Trigger** > **OnSubjobOk** link.

Configuring the tLibraryLoad component

1. Double-click on **tLibraryLoad** to display its **Basic settings**. From the **Library** list, select *jakarta-oro-2.0.8.jar*.

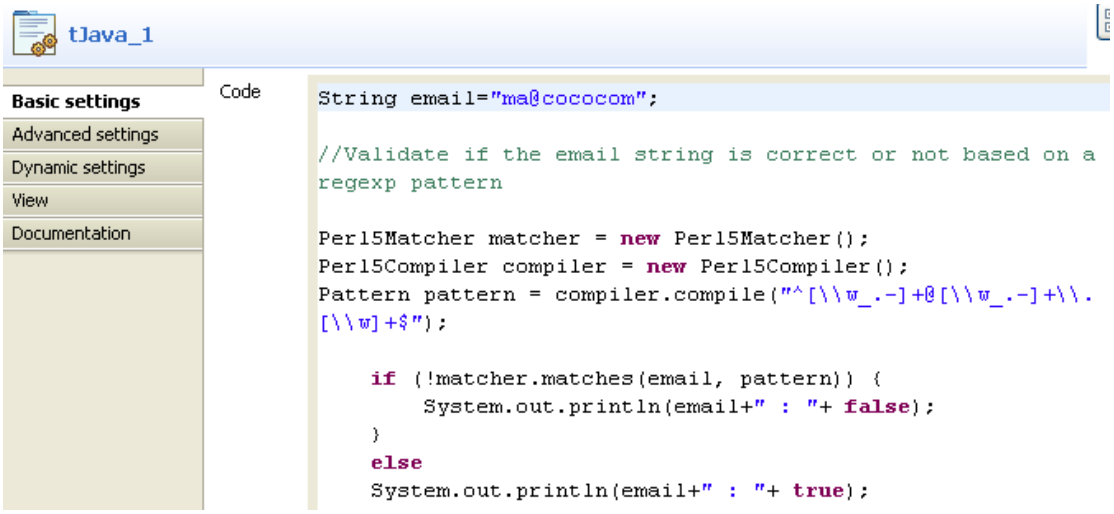
2. In the **Import** field of the **Advanced settings** tab, type *import org.apache.oro.text.regex.*;*



Configuring the tJava component

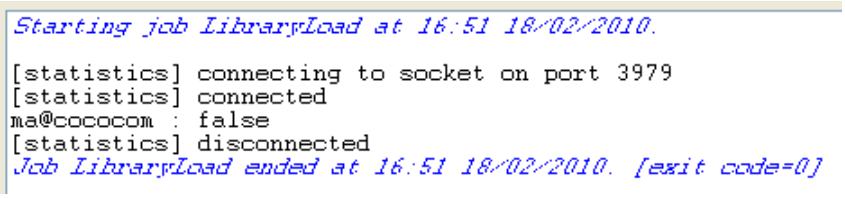
1. Double-click on **tJava** to display its **Component** view.

2. In the **Basic settings** tab, enter your code, as in the screenshot below. The code allows you to check whether the character string pertains to an e-mail address, based on the regular expression: `"^[\\w_.-]+@[\\w_.-]+\\.[\\w]+$"`.



Job execution

Press **F6** to save and run the Job.



The **Console** displays the boolean *false*. Hence, the e-mail address is not valid as the format is incorrect.

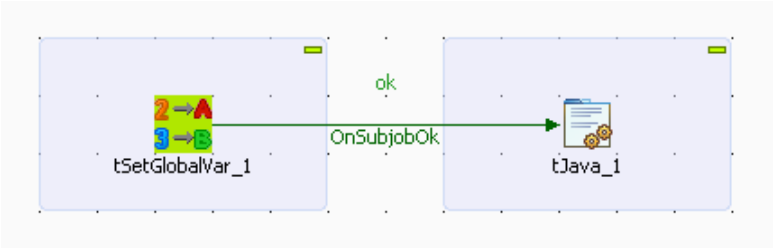
tSetGlobalVar

tSetGlobalVar properties

Component family	Custom Code	
Function	**tSetGlobalVar** allows you to define and set global variables in GUI.	
Purpose	**tSetGlobalVar** facilitates the process of defining global variables.	
Basic settings	*Variables*	This table contains two columns. **Key**: Name of the variable to be called in the code. **Value**: Value assigned to this variable.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as a one-component subjob.	
Limitation	Knowledge of Java language is required.	

Scenario: Printing out the content of a global variable

This scenario is a simple Job that prints out the value of a global variable defined in the **tSetGlobalVar** component.

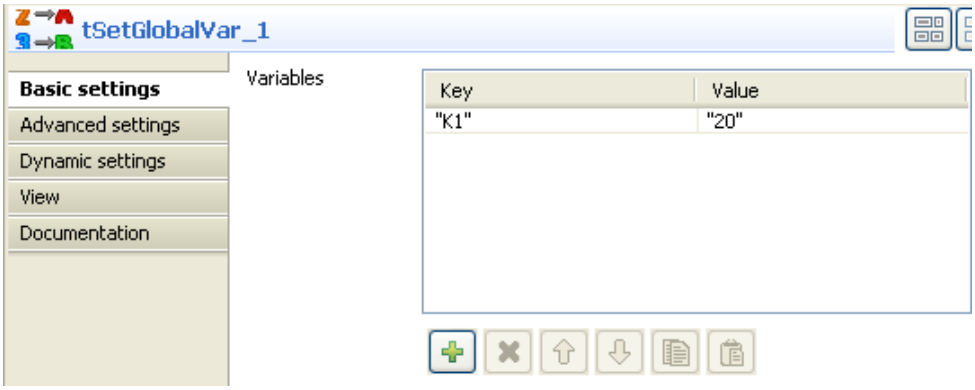


Setting up the Job

1. Drop the following components from the **Palette** onto the design workspace: **tSetGlobalVar** and **tJava**.
2. Connect the **tSetGlobalVar** component to the **tJava** component using a **Trigger** > **OnSubjobOk** connection.

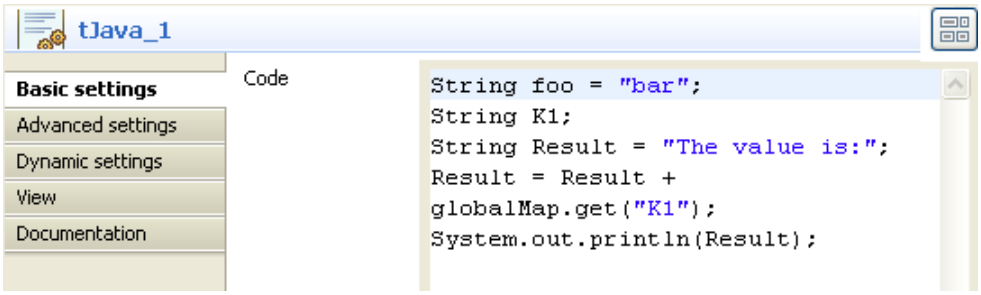
Configuring the tSetGlobalVar component

1. Double-click the **tSetGlobalVar** component to display its **Basic settings** view.



2. Click the plus button to add a line in the **Variables** table, and fill the **Key** and **Value** fields with *K1* and *20* respectively.

3. Then double-click the **tJava** component to display its **Basic settings** view.



4. In the **Code** area, type in the following lines:

```
String foo = "bar";
String K1;
String Result = "The value is:";
```

```
Result = Result + globalMap.get("K1");
```

```
System.out.println(Result);
```

In this use case, we use the *Result* variable. To access the global variable list, press **Ctrl + Space** bar on your keyboard and select the relevant global parameter.

Job execution

Save your Job and press **F6** to execute it.

The content of global variable *K1* is displayed on the console.

```
Starting job SetGlobalVar at 16:22 25/02/2011.

[statistics] connecting to socket on port 3489
[statistics] connected
The value is:20
[statistics] disconnected
Job SetGlobalVar ended at 16:22 25/02/2011.
[exit code=0]
```

Data Quality components

This chapter details the main components that you can find in the **Data Quality** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Data Quality family comprises dedicated components that help you improve the quality of your data. These components covers various needs such as narrow down filtering the unique row, calculating CRC, finding data based on fuzzy matching, and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAddCRCRow

tAddCRCRow properties

Component family	Data Quality	
Function	**tAddCRCRow** calculates a surrogate key based on one or several columns and adds it to the defined schema.	
Purpose	Providing a unique ID helps improving the quality of processed data.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. In this component, a new CRC column is automatically added.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Implication	Select the check box facing the relevant columns to be used for the surrogate key checksum.
Advanced Settings	*CRC type*	Select a CRC type in the list. The longer the CRC, the least overlap you will have.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is an intermediary step. It requires an input flow as well as an output.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Adding a surrogate key to a file

This scenario describes a Job adding a surrogate key to a delimited file schema.

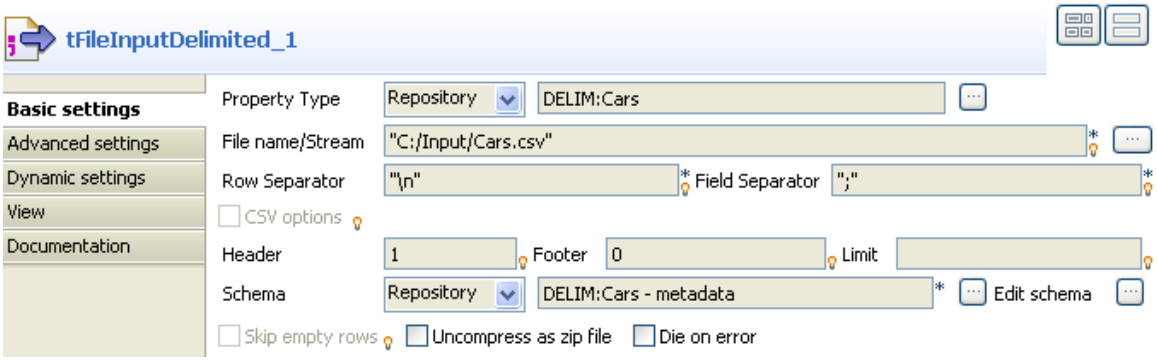
Setting up the Job

1. Drop the following components: **tFileInputDelimited**, **tAddCRCRow** and **tLogRow**.
2. Connect them using a **Main row** connection.

Configuring the input component

*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

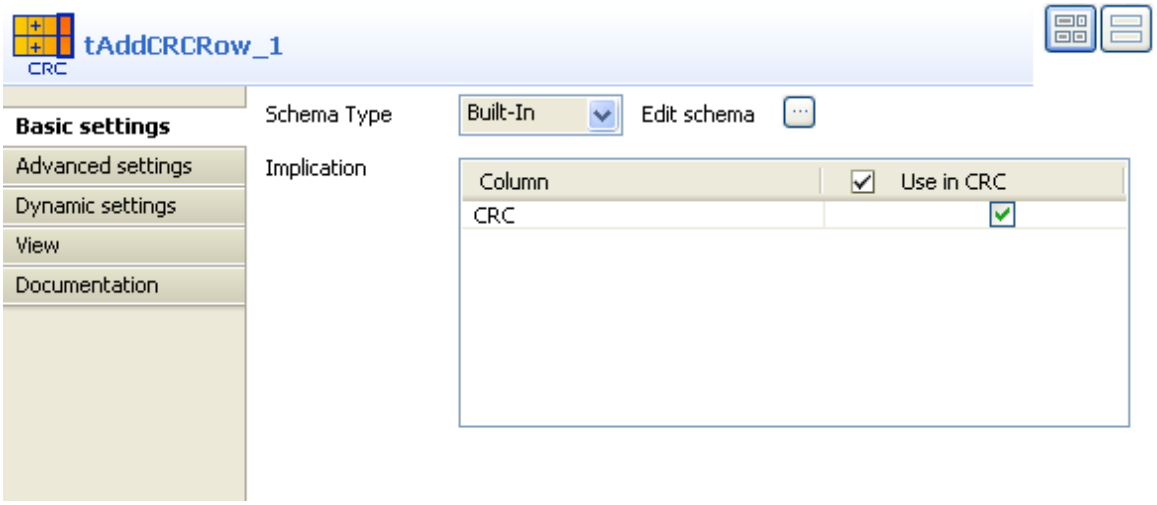
1. In the **tFileInputDelimited Component** view, set the **File Name** path and all related properties in case these are not stored in the **Repository**.



2. Create the schema through the **Edit Schema** button. Remember to set the data type column and for more information on the Date pattern to be filled in, visit *http://docs.oracle.com/javase/6/docs/api/index.html*.

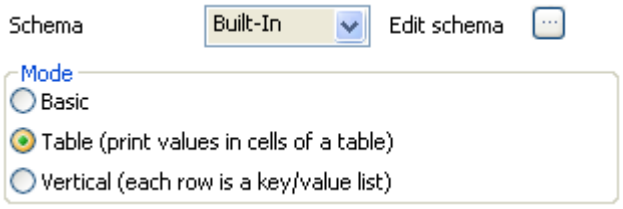
Configuring the tAddCRCRow component

1. In the **tAddCRCRow Component** view, select the check boxes of the input flow columns to be used to calculate the CRC.



Notice that a CRC column (read-only) has been added at the end of the schema.

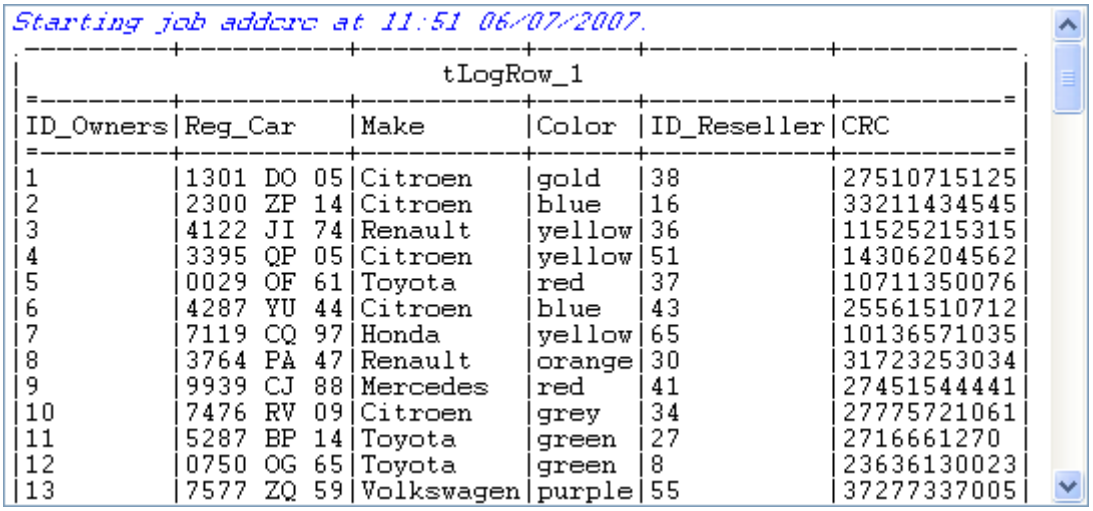
2. Select **CRC32** as **CRC Type** to get a longer surrogate key.



3. In the **Basic settings** view of **tLogRow**, select the **Print values in cells of a table** option to display the output data in a table on the Console.

Job execution

Then save your Job and press **F6** to execute it.



```
Starting job addcrc at 11:51 06/07/2007.
.---------+----------+----------+------+-----------+-----------.
|                              tLogRow_1                         |
|=---------+----------+----------+------+-----------+-----------=|
|ID_Owners|Reg_Car    |Make      |Color |ID_Reseller|CRC         |
|=---------+----------+----------+------+-----------+-----------=|
|1         |1301 DO 05|Citroen   |gold  |38         |27510715125|
|2         |2300 ZP 14|Citroen   |blue  |16         |33211434545|
|3         |4122 JI 74|Renault   |yellow|36         |11525215315|
|4         |3395 QP 05|Citroen   |yellow|51         |14306204562|
|5         |0029 OF 61|Toyota    |red   |37         |10711350076|
|6         |4287 YU 44|Citroen   |blue  |43         |25561510712|
|7         |7119 CQ 97|Honda     |yellow|65         |10136571035|
|8         |3764 PA 47|Renault   |orange|30         |31723253034|
|9         |9939 CJ 88|Mercedes  |red   |41         |27451544441|
|10        |7476 RV 09|Citroen   |grey  |34         |27775721061|
|11        |5287 BP 14|Toyota    |green |27         |2716661270 |
|12        |0750 OG 65|Toyota    |green |8          |23636130023|
|13        |7577 ZQ 59|Volkswagen|purple|55         |37277337005|
```

An additional CRC Column has been added to the schema calculated on all previously selected columns (in this case all columns of the schema).

tChangeFileEncoding

tChangeFileEncoding component belongs to two component families: Data Quality and File. For more information about **tChangeFileEncoding**, see section *tChangeFileEncoding*.

tExtractRegexFields

tExtractRegexFields belongs to two component families: Data Quality and Processing. For more information on **tExtractRegexFields**, see section *tExtractRegexFields*.

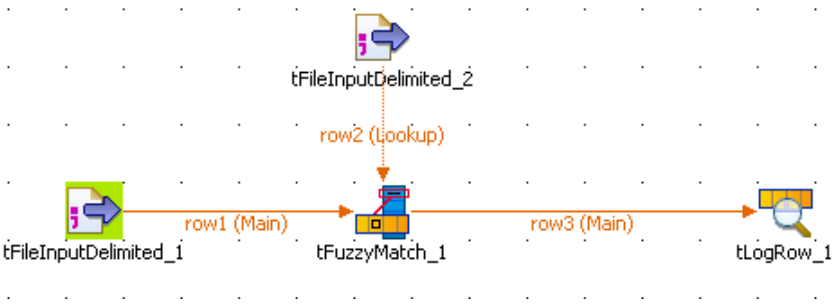
tFuzzyMatch

tFuzzyMatch properties

Component family	Data Quality	
Function	Compares a column from the main flow with a reference column from the lookup flow and outputs the main flow data displaying the distance	
Purpose	Helps ensuring the data quality of any source data against a reference data source.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Two read-only columns, Value and Match are added to the output schema automatically. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Matching type	Select the relevant matching algorithm among: **Levenshtein**: Based on the edit distance theory. It calculates the number of insertion, deletion or substitution required for an entry to match the reference entry. **Metaphone**: Based on a phonetic algorithm for indexing entries by their pronunciation. It first loads the phonetics of all entries of the lookup reference and checks all entries of the main flow against the entries of the reference flow. **Double Metaphone**: a new version of the Metaphone phonetic algorithm, that produces more accurate results than the original algorithm. It can return both a primary and a secondary code for a string. This accounts for some ambiguous cases as well as for multiple variants of surnames with common ancestry.
	Min distance	(Levenshtein only) Set the minimum number of changes allowed to match the reference. If set to 0, only perfect matches are returned.
	Max distance	(Levenshtein only) Set the maximum number of changes allowed to match the reference.
	Matching column	Select the column of the main flow that needs to be checked against the reference (lookup) key column
	Unique matching	Select this check box if you want to get the best match possible, in case several matches are available.
	Matching item separator	In case several matches are available, all of them are displayed unless the unique match box is selected. Define the delimiter between all matches.
Usage	This component is not startable (green background) and it requires two input components and an output component.	

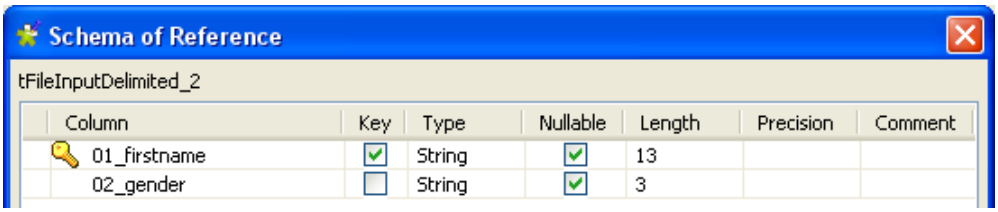
Scenario 1: Levenshtein distance of 0 in first names

This scenario describes a four-component Job aiming at checking the edit distance between the *First Name* column of an input file with the data of the reference input file. The output of this Levenshtein type check is displayed along with the content of the main flow on a table

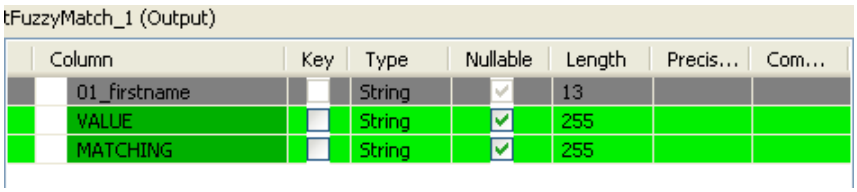


- Drag and drop the following components from the **Palette** to the design workspace: **tFileInputDelimited** (x2), **tFuzzyMatch**, **tFileOutputDelimited**.
- Define the first **tFileInputDelimited** Basic settings. Browse the system to the input file to be analyzed and most importantly set the schema to be used for the flow to be checked.
- In the schema, set the **Type** of data in the Java version, especially if you are in **Built-in** mode.
- Link the defined input to the **tFuzzyMatch** using a **Main** row link.
- Define the second **tFileInputDelimited** component the same way.

Make sure the reference column is set as key column in the schema of the lookup flow.

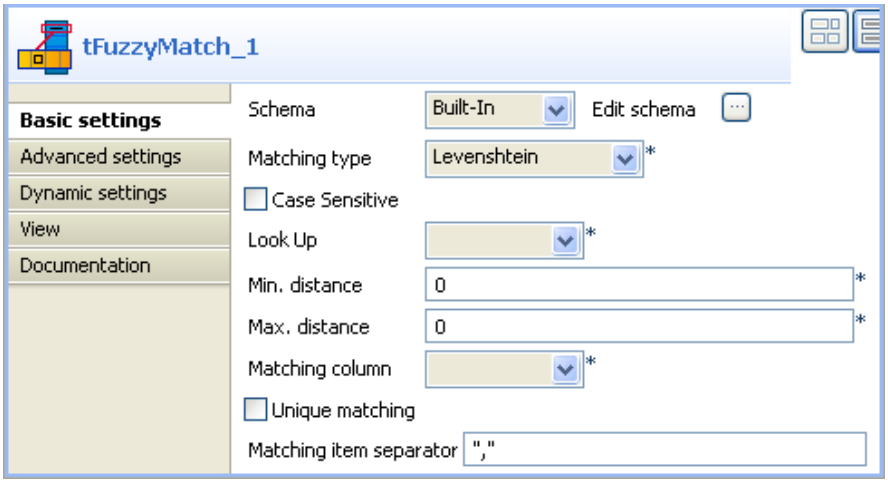


- Then connect the second input component to the **tFuzzyMatch** using a main row (which displays as a **Lookup** row on the design workspace).
- Select the **tFuzzyMatch** Basic settings.
- The **Schema** should match the **Main** input flow schema in order for the main flow to be checked against the reference.



- Note that two columns, **Value** and **Matching**, are added to the output schema. These are standard matching information and are read-only.
- Select the method to be used to check the incoming data. In this scenario, *Levenshtein* is the **Matching type** to be used.

- Then set the distance. In this method, the distance is the number of char changes (insertion, deletion or substitution) that needs to be carried out in order for the entry to fully match the reference.



- In this use case, we want the distance be of 0 for the min. or for the max. This means only the exact matches will be output.
- Also, clear the **Case sensitive** check box.
- And select the column of the main flow schema that will be selected. In this example, the first name.
- No need to select the **Unique matching** check box nor hence the separator.
- Link the **tFuzzyMatch** to the standard output **tLogRow**. No other parameters than the display delimiter is to be set for this scenario.
- Save the Job and press **F6** to execute the Job.

```
audra (1)||
audra (2)||
audrea||
audrey||
august (1)||
august (2)||
augusta||
auguste|0|auguste
augustijn||
augustin|0|augustin
augustine||
augusto||
augusts||
augustus||
aukusti||
```

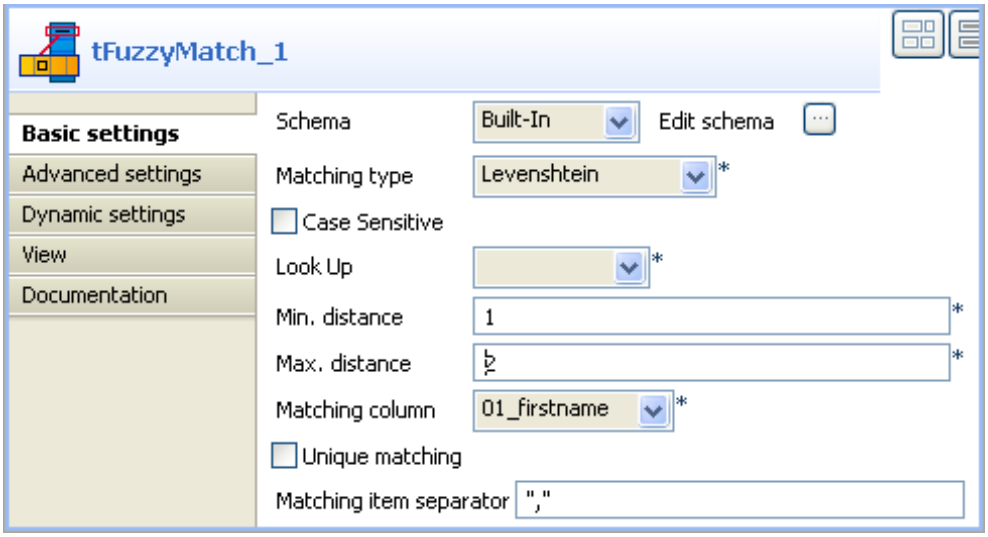
As the edit distance has been set to 0 (min and max), the output shows the result of a regular join between the main flow and the lookup (reference) flow, hence only full matches with Value of 0 are displayed.

A more obvious example is with a minimum distance of 1 and a max. distance of 2, see section *Scenario 2: Levenshtein distance of 1 or 2 in first names*

Scenario 2: Levenshtein distance of 1 or 2 in first names

This scenario is based on the scenario 1 described above. Only the min and max distance settings in **tFuzzyMatch** component get modified, which will change the output displayed.

- In the **Component** view of the **tFuzzyMatch**, change the min distance from 0 to 1. This excludes straight away the exact matches (which would show a distance of 0).
- Change also the max distance to 2 as the max distance cannot be lower than the min distance. The output will provide all matching entries showing a discrepancy of 2 characters at most.



- No other change of the setting is required.
- Make sure the **Matching item separator** is defined, as several references might be matching the main flow entry.
- Save the new Job and press **F6** to run it.

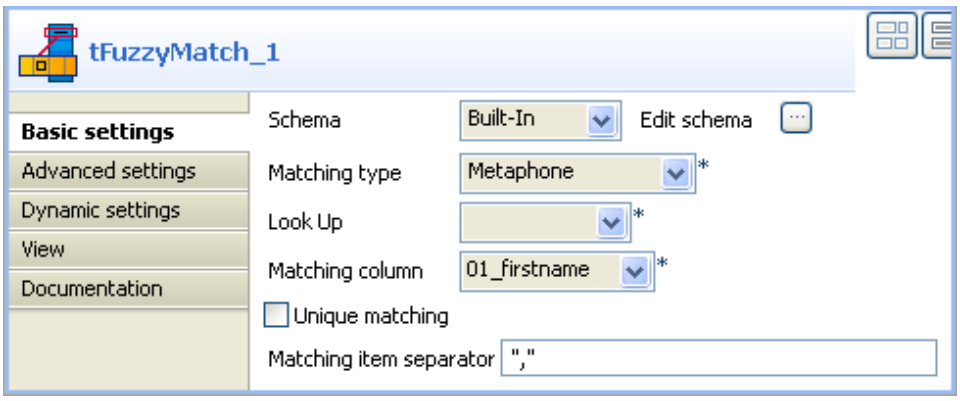
```
audrea|2|aude
audrey|2|aude
august (1)||
august (2)||
augusta|1|auguste
auguste|2|augustin
augustijn|1|augustin
augustin|2|auguste
augustine|1|augustin
augusto|1|auguste
augusts|1|auguste
augustus|2|auguste,augustin
aukusti|2|auguste,augustin
aulay||
aulus|2|jules
```

As the edit distance has been set to 2, some entries of the main flow match several reference entries.

You can also use another method, the metaphone, to assess the distance between the main flow and the reference,

Scenario 3: Metaphonic distance in first name

This scenario is based on the scenario 1 described above.



- Change the **Matching type** to **Metaphone**. There is no min nor max distance to set as the matching method is based on the discrepancies with the phonetics of the reference.
- Save the Job and press **F6**. The phonetics value is displayed along with the possible matches.

```
audrey||
august (1)|AKST|auguste
august (2)|AKST|auguste
augusta|AKST|auguste
auguste|AKST|auguste
augustijn||
augustin|AKSTN|augustin
augustine|AKSTN|augustin
augusto|AKST|auguste
augusts||
augustus||
aukusti|AKST|auguste
aulay||
aulus||
aune||
```

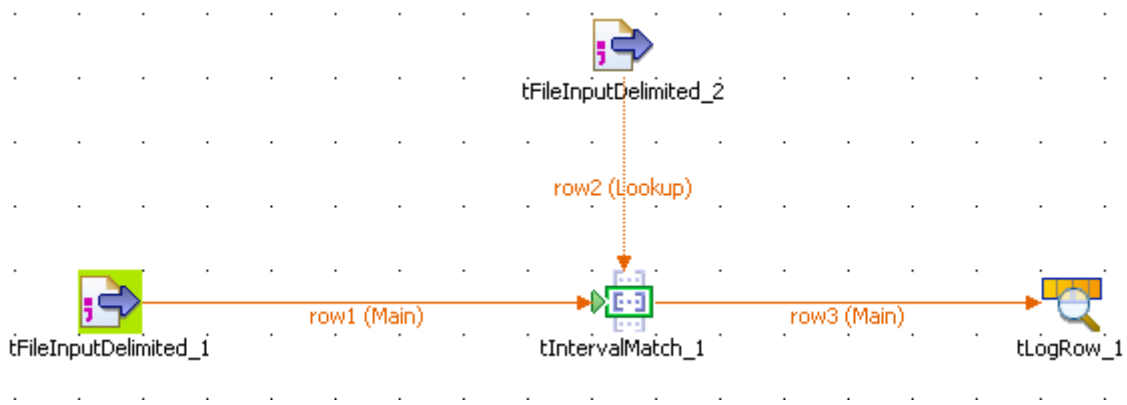
tIntervalMatch

tIntervalMatch properties

Component family	Data Quality	
Function	**tIntervalMatch** receives a main flow and aggregates it based on join to a lookup flow (Java). Then it matches a specified value to a range of values and returns related information.	
Purpose	Helps to return a value based on a Join relation.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Search column	Select the main flow column containing the values to be matched to a range of values
	Column (LOOKUP)	Select the lookup flow column containing the values to be returned when the Join is ok.
	Lookup Column min/ bounds strictly (min)	Select the column containing the min value of the tange. Select the check box if the boundary is strict.
	Lookup Column max/ bounds strictly (max)	Select the column containing the max value of the tange. Select the check box if the boundary is strict.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

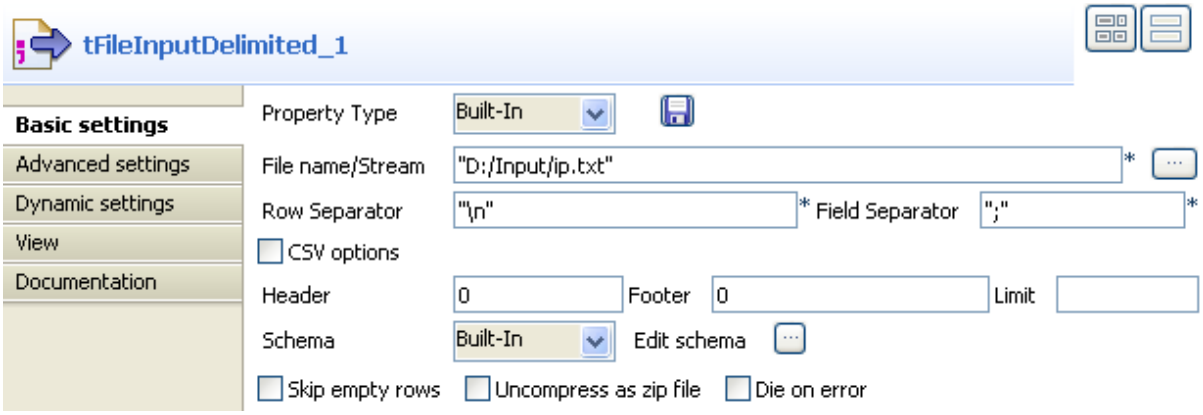
Scenario: Identifying Ip country

In this Job, a incoming main flow provides 2 columns: *Documents* and *IP* dummy values. A second file used as lookup flow in Java contains a list of sorted IP ranges and their corresponding country. This Job aims at retrieving each document's country from their IP value, in other words, creating a Join between the main flow and the lookup flow.

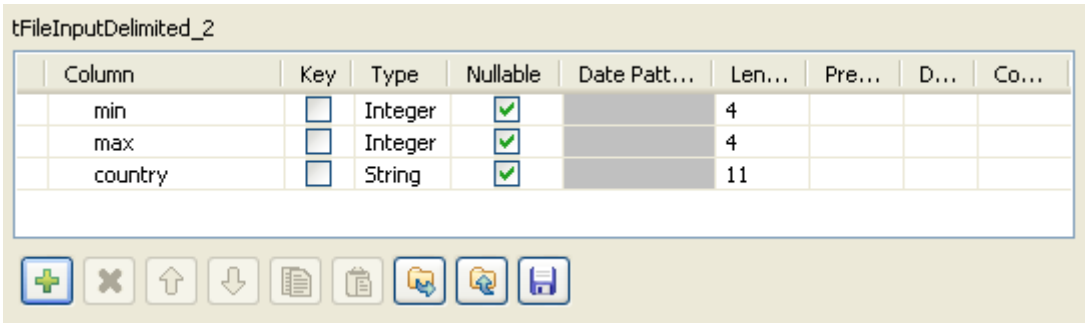


The Job requires one extra **tFileInputDelimited**, a **tIntervalMatch** and a **tLogRow**.

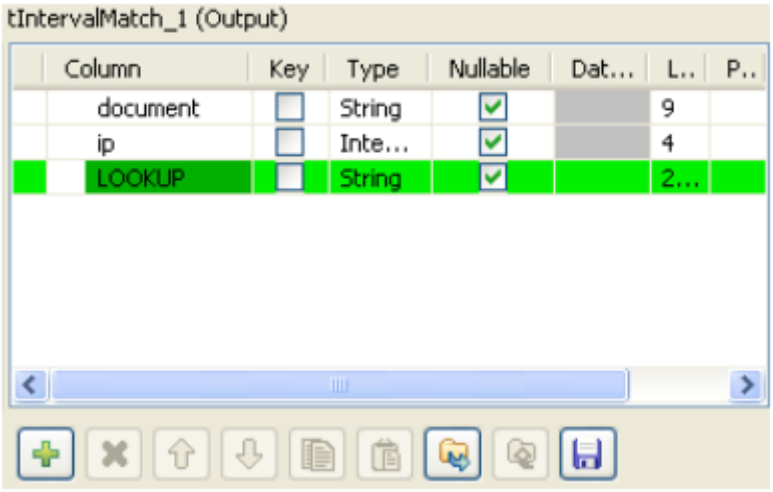
- Drop the components onto the design workspace.
- Set the basic settings of the **tFileInputDelimited** component.



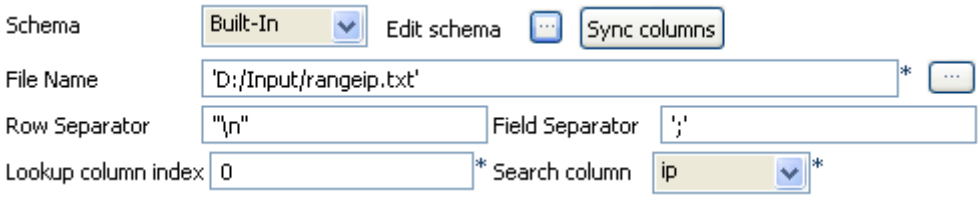
- The schema is made of two columns, respectively *Document* and *IP*
- Set the **Type** column on **String** for the *Document* column and **Integer** for the *IP* column.
- Set now the second **tFileInputDelimited** properties.



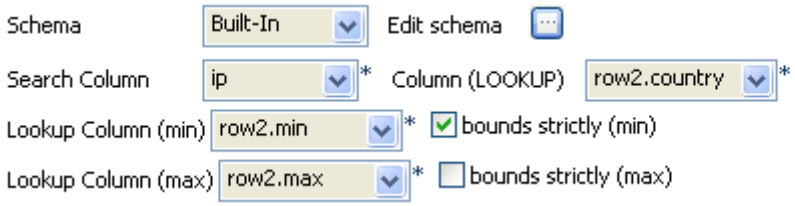
- Don’t forget to define the **Type** of data.
- Propagate the schema from the incoming main flow to the **tIntervalMatch** component.



- Note that the output schema from the **tIntervalMatch** component is read-only and is made of the input schema plus an extra Lookup column which will output the requested lookup data.
- Set the other properties of the **tIntervalMatch** component.



- Set the tIntervalMatch other properties such as the min and max column corresponding to the range bounds.



- In the **Column Lookup** field, select the column where are the values to be returned.
- In the **Search column** field, select the main flow column containing the values to be matched to the range values.
- The **tLogRow** component does not require any specific setting for this Job.

Following result is displayed:

```
Starting job intervalMatch at 17:05 01/02/2008.
document1|1200|DEUTSCHLAND
document2|4500|
document3|500|FRANCE
document1|2201|ITALIA
Job intervalMatch ended at 17:05 01/02/2008. [exit c
```

Only requested values (country) are included in the output.

tReplaceList

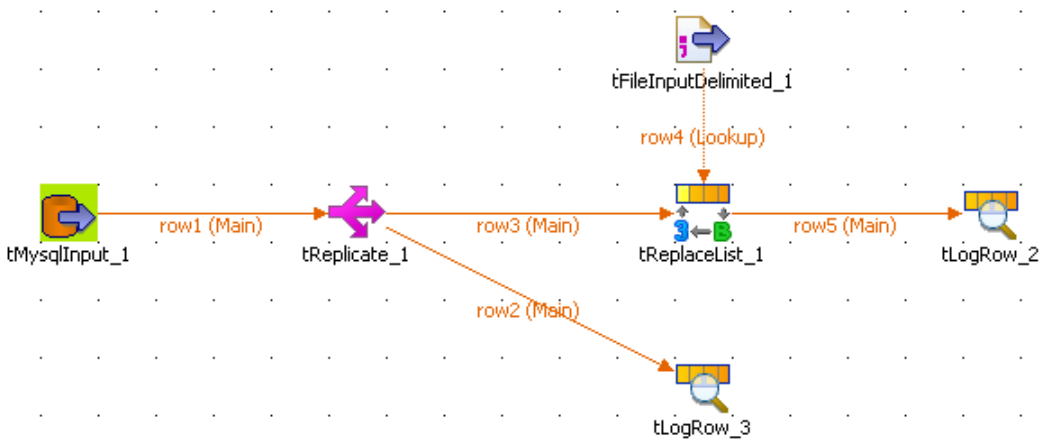
tReplaceList Properties

Component family	Data Quality	
Function	Carries out a Search and Replace operation in the input columns defined based on an external lookup.	
Purpose	Helps to cleanse all files before further processing.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Two read-only columns, Value and Match are added to the output schema automatically. *The data Type defined in the schemas must be consistent, ie., an integer can only be replaced by another integer using an integer as a look up field. Values of one type cannot be replaced by values of another type.*
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Lookup search column	Type in the position number of the column to be searched in the lookup schema. **0**: first column read **1**: second column read **n**: position number of the column in the schema read. *In order to ensure the uniqueness of values being searched, make sure this column is marked as Key in your lookup schema.*
	Lookup replacement column	Type in the position number of the column where the replacement values are stored. **0**: first column read **1**: second column read **n**: position number of the column in the schema read
	Column options	Select the columns of the main flow where the replacement is to be carried out.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	**tReplaceList** is an intermediary component. It requires an input flow and an output component.	

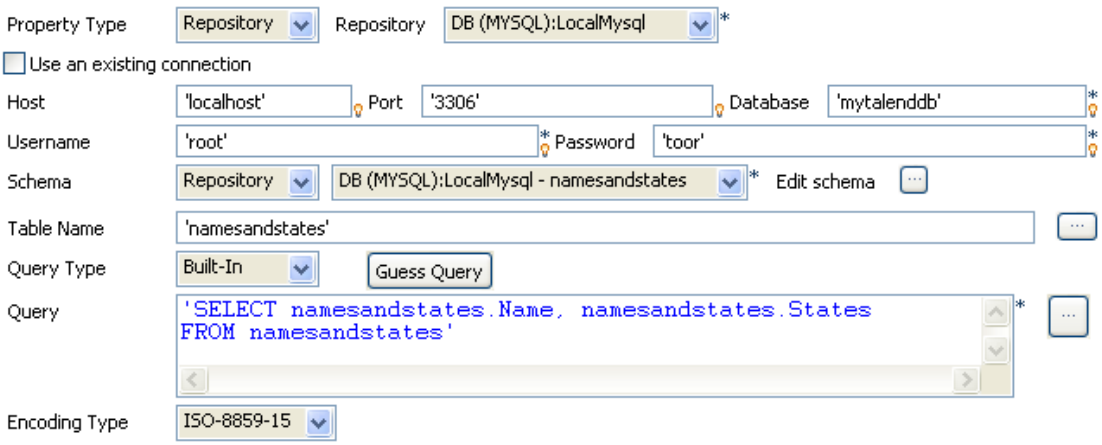
Scenario: Replacement from a reference file

If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available for the property and schema.

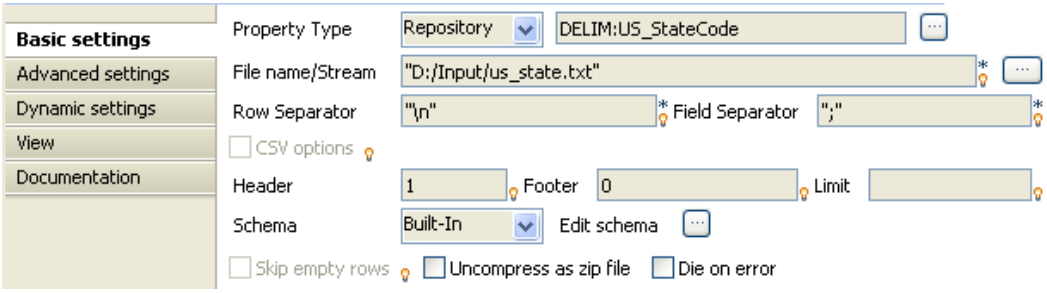
The following Job searches and replaces a list of countries with their corresponding codes. The relevant codes are taken from a reference file placed as lookup flow in the Job. The main flow is replicated and both outputs are displayed on the console, in order to show the main flow before and after replacement.



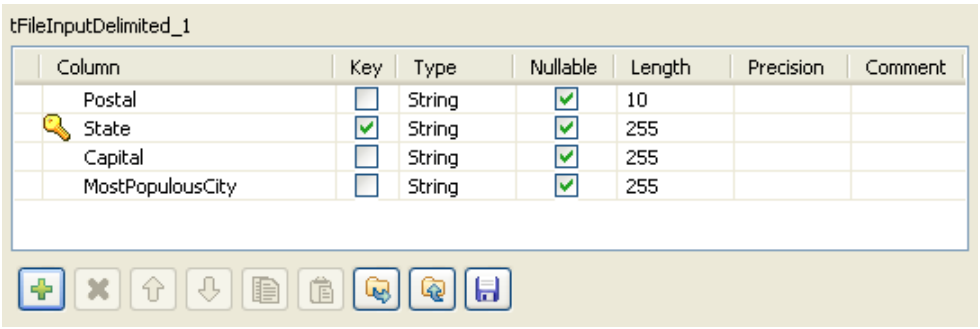
- Drop the following components from the **Palette** to the design workspace: **tMysqlInput**, **tFileInputDelimited**, **tReplicate**, **tReplaceList** and **tLogRow** (x2).
- Connect the components using **Main Row** connections via a right-click on each component. Notice that the main row coming from the reference flow (**tFileInputDelimited**) is called a lookup row.
- Select the **tMysqlInput** component and set the input flow parameters.



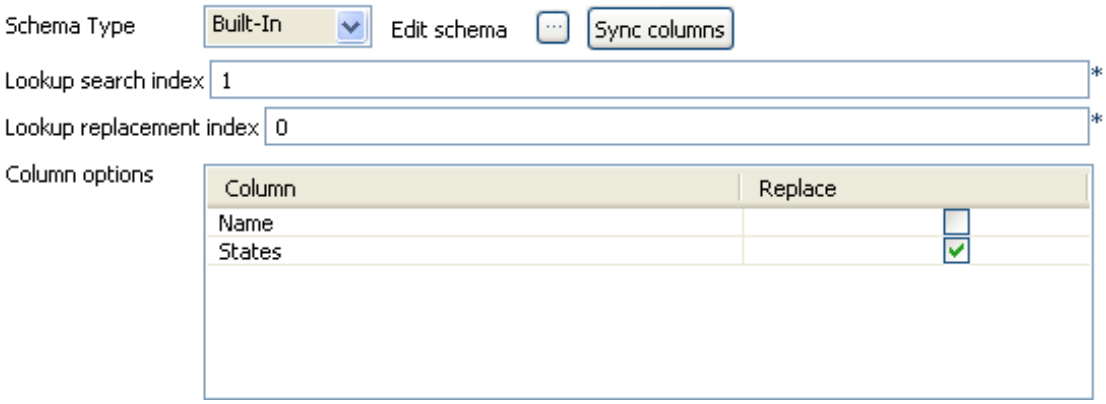
- The input schema is made of two columns: *Names*, *States*. The column States gathered the name of the United States of America which are to be replaced by their respective code.
- In the **Query** field, make sure the *State* column is included in the `Select` statement. In this use case, all columns are selected.
- Check the **tReplicate** component setting. The schema is simply duplicated into two identical flows, but no change to the schema can be made.
- Then double-click on the **tFileInputDelimited** component, to set the reference file.



- The file includes two columns: *Postal*, *State* where *Postal* provides the zipcode corresponding to the name given in the respective row of the *State* column.
- The fields are delimited by semicolons and rows are separated by carriage returns.
- Edit the lookup flow schema.



- Make sure the lookup search column (in this use case: *State*) is a key, in order to ensure the uniqueness of the values being searched.
- Select the **tReplaceList** and set the operation to carry out.
- The schema is retrieved from the previous component of the main flow.



- In **Lookup search index** field, type in the position index of the column being searched. In this use case, *State* is the second column of the lookup input file, therefore type in **1** in this field.
- In **Lookup replacement index** field, fill in the position number of the column containing the replacement values, in this example: *Postal* for the State codes.
- In the **Column options** table, select the *States* column as in this use case, the State names are to be replaced with their corresponding code.
- In both **tLogRow** components, select the **Print values in table cells** check box for a better readability of the outputs.

- Save the Job and press **F6** to execute it.

```
Starting job ReplaceList at 15:06 22/10/2007.
.-----------------------------------.
|               tLogRow_3           |
+--------------------+--------------+
| Name               | States       |
+--------------------+--------------+
| William Grant      | Iowa         |
| William Hoover     | New York     |
| Grover Lincoln     | North Dakota |
| Lyndon Jefferson   | Ohio         |
| Gerald Hayes       | Washington   |
| Benjamin Grant     | Maine        |
| George Pierce      | Connecticut  |
| Jimmy Reagan       | Alaska       |
| Martin Hayes       | Washington   |
| Franklin Jefferson | Iowa         |
| Andrew Nixon       | New Hampshire|
'--------------------+--------------'

.-----------------------------.
|           tLogRow_2         |
+--------------------+--------+
| Name               | States |
+--------------------+--------+
| William Grant      | IA     |
| William Hoover     | NY     |
| Grover Lincoln     | ND     |
| Lyndon Jefferson   | OH     |
| Gerald Hayes       | WA     |
| Benjamin Grant     | ME     |
| George Pierce      | CT     |
| Jimmy Reagan       | AK     |
| Martin Hayes       | WA     |
| Franklin Jefferson | IA     |
| Andrew Nixon       | NH     |
'--------------------+--------'

Job ReplaceList ended at 15:06 22/10/2007. [exit code=0]
```

The first flow output shows the States column with full state names as it comes from the main input flow.

The second flow output shows the States column after the State column names have been replaced with their respective codes.

tSchemaComplianceCheck

tSchemaComplianceCheck Properties

Component family	Data Quality	
Function	Validates all input rows against a reference schema or check types, nullability, length of rows against reference values. The validation can be carried out in full or partly.	
Purpose	Helps to ensure the data quality of any source data against a reference data source.	
Basic settings	*Base Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Describe the structure and nature of your data to be processed as it is. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Check all columns from schema	Select this option to carry out all checks on all columns against the base schema.
	Custom defined	Select this option to carry out particular checks on particular columns. When this option is selected, the **Checked Columns** table shows.
	Checked Columns	In this table, define what checks are to be carried out on which columns.
		Column: Displays the columns names.
		Type: Select the type of data each column is supposed to contain. This validation is mandatory for all columns.
		Date pattern: Define the expected date format for each column with the data type of *Date*.
		Nullable: Select the check box in an individual column to define the column to be nullable, that is, to allow empty rows in this column to go to the output flow regardless of the base schema definition. To define all columns to be nullable, select the check box in the table header.
		Undefined or empty: Select the check box in an individual column to reject empty rows in this column while the column is not nullable in the base schema definition. To carry out this verification on all the columns, select the check box in the table header.
		Max length: Select the check box in an individual column to verify the data length of the column against the length definition of the base schema. To carry out this verification on all the columns, select the check box in the table header.
	Use another schema for compliance check	Define a reference schema as you expect the data to be, in order to reject the non-compliant data. It can be restrictive on data type, null values, and/or length.
	Trim the excess content of column when length checking chosen and the length is greater than defined length	With any of the three modes of **tSchemaComplianceCheck**, select this check box to truncate the data that exceeds the length specified rather than reject it. This option is applicable only on data of **String** type.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.

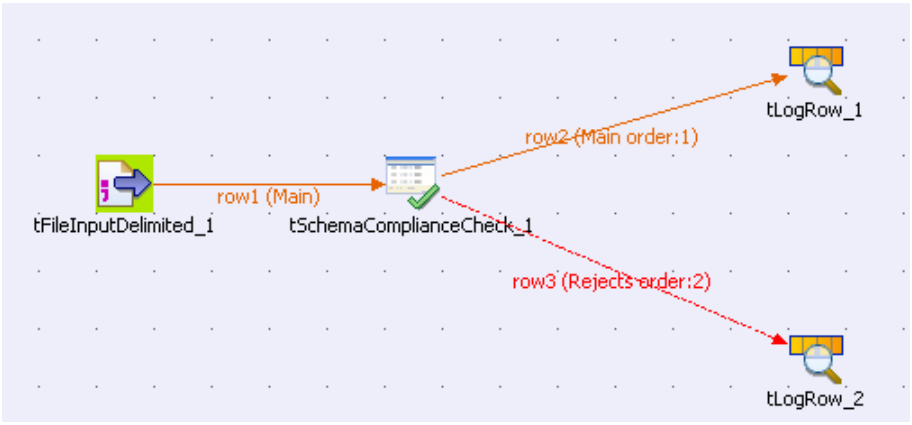
	Use Fastest Date Check	Select this check box to perform a fast date format check using the *TalendDate.isDate()* method of the *TalendDate* system routine if *Date pattern* is not defined. For more information about routines, see *Talend Open Studio for Big Data User Guide*.
	Treat all empty string as NULL	Select this check box to treat any empty fields in any columns as null values, instead of empty strings. By default, this check box is selected. When it is cleared, the **Choose Column(s)** table shows to let you select individual columns.
Usage	This component is an intermediary step in the flow allowing to exclude from the main flow the non-compliant data. This component cannot be a start component as it requires an input flow. It also requires at least one output component to gather the validated flow, and possibly a second output component for rejected data using Rejects link. For more information, see *Talend Open Studio for Big Data User Guide*.	

Scenario: Validating data against schema

This very basic scenario shows how to check the type, nullability and length of an incoming flow against a defined reference schema. The incoming flow comes from a simple CSV file that contains heterogeneous data including wrong data type, data exceeding the maximum length, wrong ID and null values in non-nullable columns, as shown below:

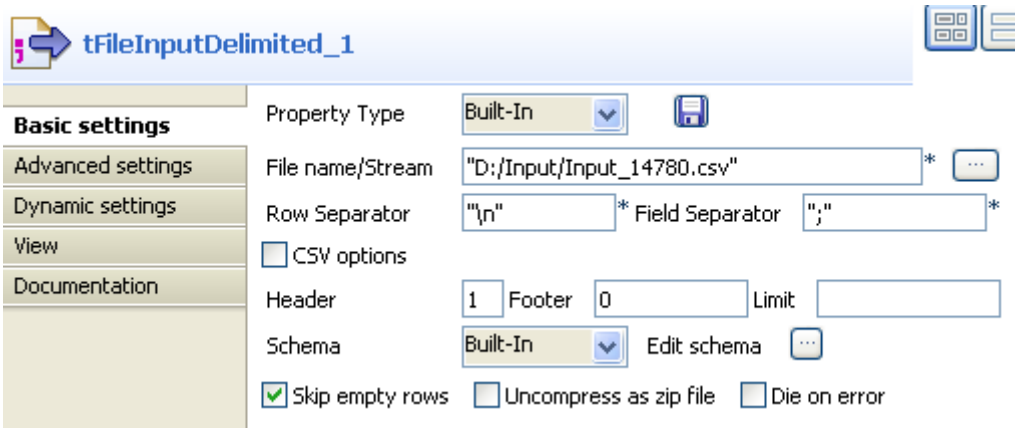
```
1  ID;Name;BirthDate;State;City
2  1;Dwight;06-04-2008;Delaware;Concord
3  2;Warren;25-10-2008;Montana
4  3;Benjamin;17-08-2008;Washington;Austin
5  4;Harry;14-04-2008;Kansas;Annapolis
6  5;Ulysses;2007-04-12;Michigan;Raleigh
7  6;James;19-08-2007;Delaware;Charleston
8  .7;Bill;20-04-2007;Illinois;Bismarck
9  8;Ulysses;04-12-2008;;Saint Paul
10 9;Thomas;09-05-2008;Maryland;Albany
11 10;Ronald;11-02-2008;Florida;Hartford
```

Upon validation, the valid rows and the rejected rows are displayed respectively in two tables on the **Run** console.

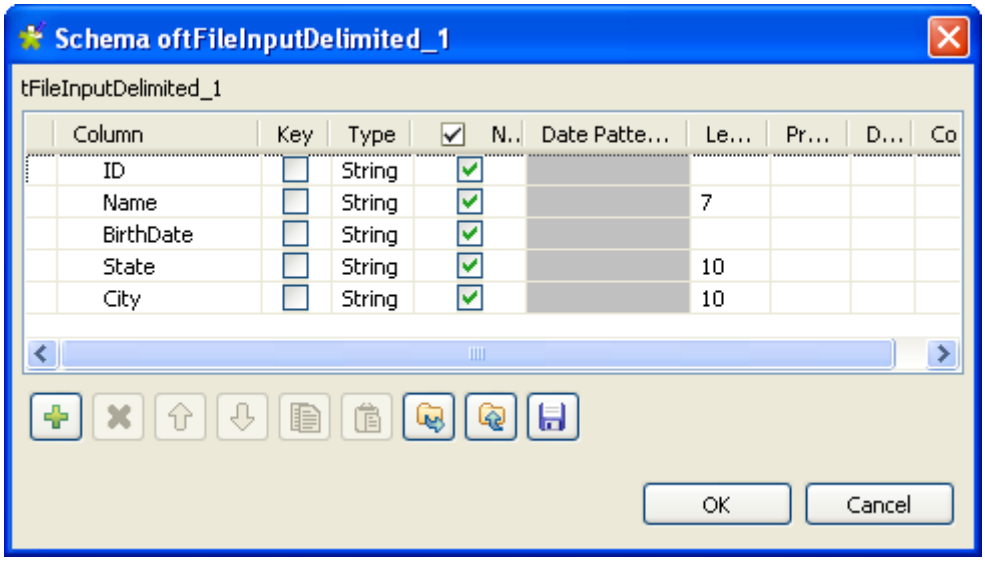


- Drop the following components: a **tFileInputDelimited**, a **tSchemaComplianceCheck**, and two **tLogRow** components from the **Palette** to the design workspace.
- Connect the **tFileInputDelimited** component to the **tSchemaComplianceCheck** component using a **Row** > **Main** connection.
- Connect the **tSchemaComplianceCheck** component to the first **tLogRow** component using a **Row > Main** connection. This output flow will gather the valid data.

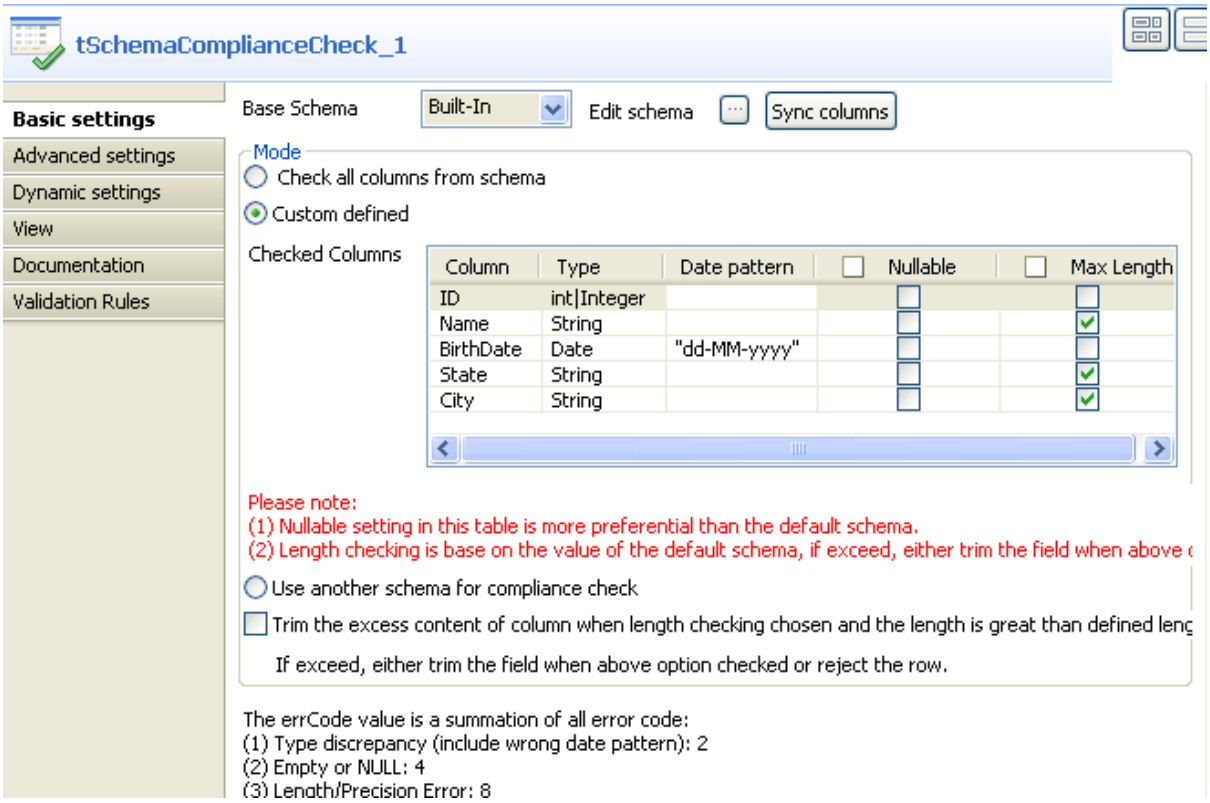
- Connect the **tSchemaComplianceCheck** component to the second **tLogRow** component using a **Row > Rejects** connection. This second output flow will gather the non-compliant data.
- Select the **Rejects** connection, and notice that the schema passed to the second **tLogRow** contains two more columns: *ErrorCode* and *ErrorMessage*. These two read-only columns provide information about the rejected data to ease error handling and troubleshooting if needed.
- Double-click the **tFileInputDelimited** component to display its **Basic settings** view.



- Fill in the **File name** field by browsing to the input file.
- Specify the header row. In this use case, the first row of the input file is the header row.
- Leave the other parameters as they are.
- Click **Edit schema** to describe the data structure of the input file. In this use case, the schema is made of five columns: *ID*, *Name*, *BirthDate*, *State*, and *City*.



- Leave the **Type** field as permissive as possible. You will define the actual type of the data in the **tSchemaComplianceCheck** component.
- Fill the **Length** field for the *Name*, *State* and *City* columns with *7*, *10* and *10* respectively.
- Click **OK** to propagate the schema and close the schema dialog box.
- Double-click the **tSchemaComplianceCheck** component to display its **Basic settings** view, wherein you will define most of the validation parameters.



- Select the **Custom defined** option in the **Mode** area to perform custom defined checks.

  In this example, we use the **Checked columns** table to set the validation parameters. However, you can also select the **Check all columns from schema** check box if you want to perform all the checks (type, nullability and length) on all the columns against the base schema, or select the **Use another schema for compliance check** option and define a new schema as the expected structure of the data.

- In the **Checked Columns** table, define the checks to be performed. In this use case:

  - The type of the *ID* column should be **Int**.

  - The length of the *Name*, *State* and *City* columns should be checked.

  - The type of the *BirthDate* column should be **Date**, and the expected date pattern is *dd-MM-yyyy*.

  - All the columns should be checked for null values, so clear the **Nullable** check box for all the columns.

To send rows containing fields exceeding the defined maximum length to the reject flow, make sure that the **Trim the excess content of column when length checking chosen and the length is greater than defined length** check box is cleared.

- In the **Advanced settings** view of the **tSchemaComplianceCheck** component, select the **Treat all empty string as NULL** option to sent any rows containing empty fields to the reject flow.

- To view the validation result in tables on the **Run** console, double-click each **tLogRow** component and select the **Table** option in the **Basic settings** view.

- Save your Job and press **F6** to launch it.

  Two tables are displayed on the console, showing the valid data and rejected data respectively.

```
Starting job feature_14780 at 17:32 06/01/2011.

[statistics] connecting to socket on port 3622
[statistics] connected
.--+------+----------+--------+----------.
|               tLogRow_1                 |
|=-+------+----------+--------+---------=|
|ID|Name  |BirthDate |State   |City      |
|=-+------+----------+--------+---------=|
|1 |Dwight|06-04-2008|Delaware|Concord   |
|4 |Harry |14-04-2008|Kansas  |Annapolis |
|6 |James |19-08-2007|Delaware|Charleston|
|9 |Thomas|09-05-2008|Maryland|Albany    |
|10|Ronald|11-02-2008|Florida |Hartford  |
'--+------+----------+--------+----------'
.--+--------+----------+----------+----------+---------+-----------------------------------------.
|                                        tLogRow_2                                                |
|=-+--------+----------+----------+----------+---------+-----------------------------------------=|
|ID|Name    |BirthDate |State     |City      |errorCode|errorMessage                             |
|=-+--------+----------+----------+----------+---------+-----------------------------------------=|
|2 |Warren  |25-10-2008|Montana   |          |4        |City:empty or null                       |
|3 |Benjamin|17-08-2008|Washington|Austin    |8        |Name:exceed max length                   |
|5 |Ulysses |2007-04-12|Michigan  |Raleigh   |2        |BirthDate:wrong DATE pattern or wrong DATE data|
|.7|Bill    |20-04-2007|Illinois  |Bismarck  |2        |ID:wrong type                            |
|8 |Ulysses |04-12-2008|          |Saint Paul|4        |State:empty or null                      |
'--+--------+----------+----------+----------+---------+-----------------------------------------'

[statistics] disconnected
Job feature_14780 ended at 17:32 06/01/2011. [exit code=0]
```

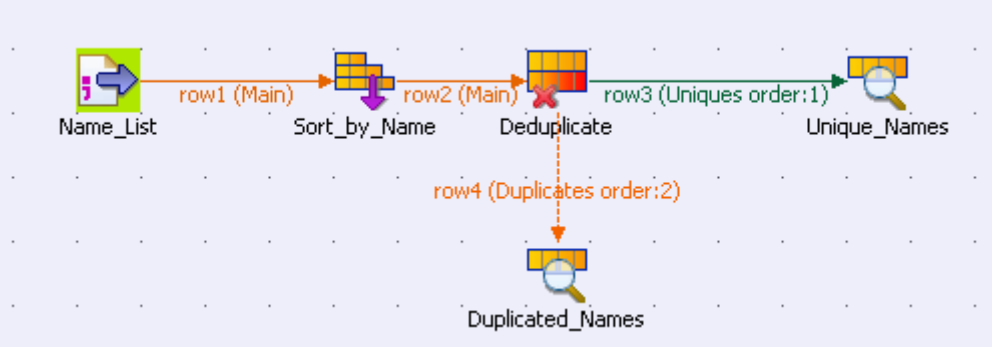
tUniqRow

tUniqRow Properties

Component family	Data Quality	
Function	Compares entries and sorts out duplicate entries from the input flow.	
Purpose	Ensures data quality of input or output flow in a Job.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Unique key	In this area, select one or more columns to carry out deduplication on the particular column(s) - Select the **Key attribute** check box to carry out deduplication on all the columns - Select the **Case sensitive** check box to differentiate upper case and lower case
Advanced settings	*Only once each duplicated key*	Select this check box if you want to have only the first duplicated entry in the column(s) defined as key(s) sent to the output flow for duplicates.
	Use of disk (suitable for processing large row set)	Select this check box to enable generating temporary files on the hard disk when processing a large amount of data. This helps to prevent Job execution failure caused by memory overflow. With this check box selected, you need also to define: - **Buffer size in memory**: Select the number of rows that can be buffered in the memory before a temporary file is to be generated on the hard disk. - **Directory for temp files**: Set the location where the temporary files should be stored. *Make sure that you specify an existing directory for temporary files; otherwise your Job execution will fail.*
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Scenario 1: Deduplicating entries

In this five-component Job, we will sort entries on an input name list, find out duplicated names, and display the unique names and the duplicated names on the **Run** console.

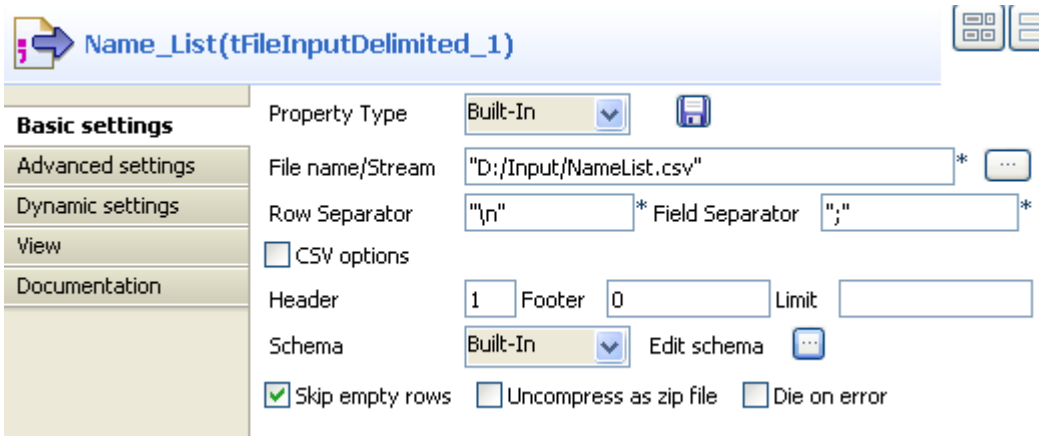


Setting up the Job

1. Drop a **tFileInputDelimited**, a **tSortRow**, a **tUniqRow**, and two **tLogRow** components from the **Palette** to the design workspace, and name the components as shown above.
2. Connect the **tFileInputDelimited** component, the **tSortRow** component, and the **tUniqRow** component using **Row** > **Main** connections.
3. Connect the **tUniqRow** component and the first **tLogRow** component using a **Main** > **Uniques** connection.
4. Connect the **tUniqRow** component and the second **tLogRow** component using a **Main** > **Duplicates** connection.

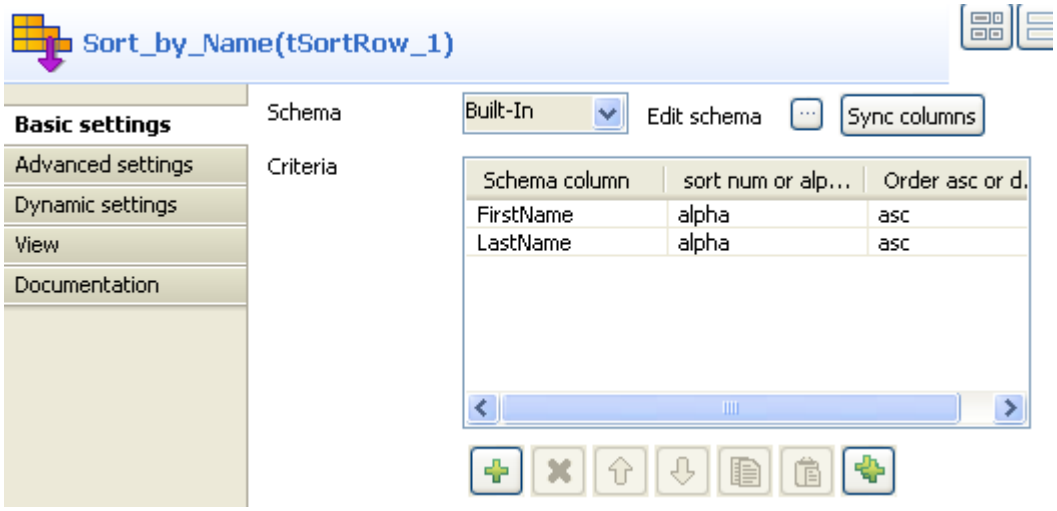
Configuring the components

1. Double-click the **tFileInputDelimited** component to display its **Basic settings** view.



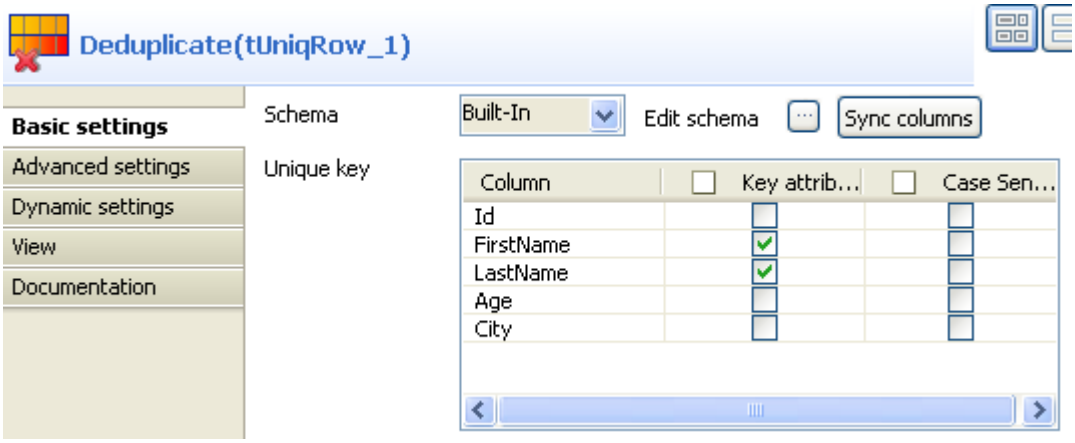
2. Click the **[...]** button next to the **File Name** field to browse to your input file.
3. Define the header and footer rows. In this use case, the first row of the input file is the header row.
4. Click **Edit schema** to define the schema for this component. In this use case, the input file has five columns: *Id*, *FirstName*, *LastName*, *Age*, and *City*. Then click **OK** to propagate the schema and close the schema editor.

5. Double-click the **tSortRow** component to display its **Basic settings** view.



6. To rearrange the entries in the alphabetic order of the names, add two rows in the **Criteria** table by clicking the plus button, select the *FirstName* and *LastName* columns under **Schema column**, select *alpha* as the sorting type, and select the sorting order.

7. Double-click the **tUniqRow** component to display its **Basic settings** view.



8. In the **Unique key** area, select the columns on which you want deduplication to be carried out. In this use case, you will sort out duplicated names.

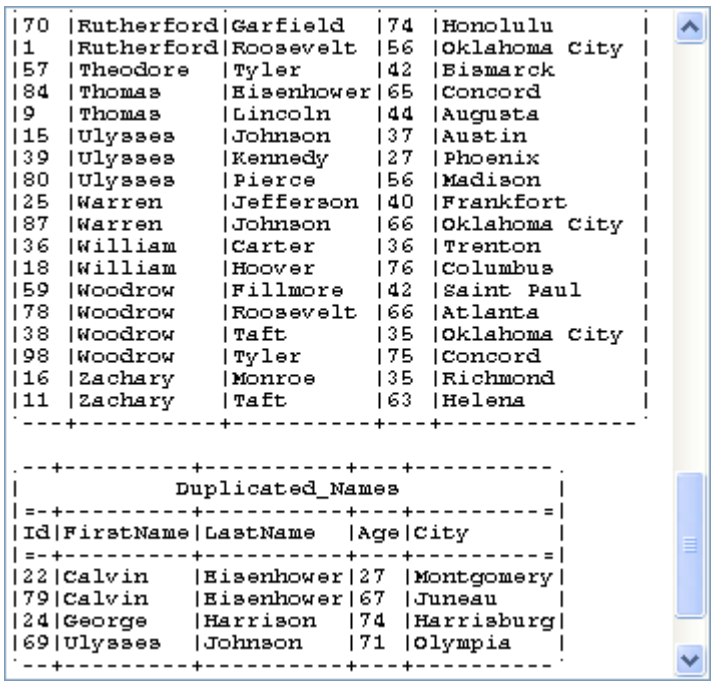
9. In the **Basic settings** view of each of the **tLogRow** components, select the **Table** option to view the Job execution result in table mode.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Run the Job by pressing **F6** or clicking the **Run** button on the **Run** tab.

   The unique names and duplicated names are displayed in different tables on the **Run** console.



```
|70 |Rutherford|Garfield  |74 |Honolulu      |
|1  |Rutherford|Roosevelt |56 |Oklahoma City |
|57 |Theodore  |Tyler     |42 |Bismarck      |
|84 |Thomas    |Eisenhower|65 |Concord       |
|9  |Thomas    |Lincoln   |44 |Augusta       |
|15 |Ulysses   |Johnson   |37 |Austin        |
|39 |Ulysses   |Kennedy   |27 |Phoenix       |
|80 |Ulysses   |Pierce    |56 |Madison       |
|25 |Warren    |Jefferson |40 |Frankfort     |
|87 |Warren    |Johnson   |66 |Oklahoma City |
|36 |William   |Carter    |36 |Trenton       |
|18 |William   |Hoover    |76 |Columbus      |
|59 |Woodrow   |Fillmore  |42 |Saint Paul    |
|78 |Woodrow   |Roosevelt |66 |Atlanta       |
|38 |Woodrow   |Taft      |35 |Oklahoma City |
|98 |Woodrow   |Tyler     |75 |Concord       |
|16 |Zachary   |Monroe    |35 |Richmond      |
|11 |Zachary   |Taft      |63 |Helena        |
'---+----------+----------+---+--------------'

.--+---------+----------+---+----------.
|           Duplicated_Names            |
|=-+---------+----------+---+---------=|
|Id|FirstName|LastName  |Age|City      |
|=-+---------+----------+---+---------=|
|22|Calvin   |Eisenhower|27 |Montgomery|
|79|Calvin   |Eisenhower|67 |Juneau    |
|24|George   |Harrison  |74 |Harrisburg|
|69|Ulysses  |Johnson   |71 |Olympia   |
'--+---------+----------+---+----------'
```

tUniservBTGeneric

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservBTGeneric properties

Component family	Data quality	
Function	**tUniservBTGeneric** enables the execution of a processing created with the Uniserv product *DQ Batch Suite*.	
Purpose	**tUniservBTGeneric** sends the data to the *DQ Batch Suite* and starts the specified *DQ Batch Suite* job. When the job execution is finished, the results are returned to the *Data Quality Service Hub Studio* for further processing.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the **Repository**. Click **Retrieve Schema** to create a schema for the components that matches the input and output fields in the *DQ Batch Suite* job.
	Host name	Host on which the Master Server of *DQ Batch Suite* runs, between double quotation marks.
	Port	Port number on which the *DQ Batch Suite* server runs, between double quotation marks.
	Client Server	Name of the client server of the *DQ Batch Suite*, between double quotation marks.
	User name	User name for the registration on the *DQ Batch Suite* server. The stated user must have the right to execute the *DQ Batch Suite* job.
	Password	Password of the stated user.
	Job directory	Directory in the *DQ Batch Suite*, in which the job is saved.
	Job name	Name of the *DQ Batch Suite* job that is to be executed.
	Job file path	File path under which the *DQ Batch Suite* job to be executed will be saved. The path to the file must be stated absolutely.
Advanced settings	*Temporary directory*	Directory in which the temporary files created during job execution are to be saved.
	Input Parameters	These parameters must correspond to the parameters in the function **Input** (tab "Format") of the *DQ Batch Suite* job. **File location**: State whether the input file is saved in the pool or the local job directory. **Directory**: If the File location = Pool, it means the directory is related to the pool directory. If the File location = Job, *"input"* must be specified here. **File name**: Name of the delimiter file which has been generated by **tUniservBTGeneric** and is to be transferred to the *DQ Batch Suite*. The file name must correspond to the file name which is defined in the function **Input** of the *DQ Batch Suite* job. **No. of header rec.**: 0 = no header record, 1 = header record in the input file. **Field separator**: Field separator defined in the function **Input** of the *DQ Batch Suite* job.

	Output Parameters	These parameters must correspond to the parameters in the function **Output** (tab "Format") of the *DQ Batch Suite* job. **File location**: State whether the output file is to be saved in the pool or the local job directory. **Directory**: If the File location = Pool, it means the directory is related to the pool directory. If the File location = Job, "output" must be specified here. **File name**: Name of the output file in the delimiter format, which is created by the *DQ Batch Suite* job. The file name must correspond to the file name defined in the function **Output** of the *DQ Batch Suite* job. **No. of header rec.**: 0 = no header record, 1 = header record in the output file. **Field separator**: Field separator defined in the function **Output** of the *DQ Batch Suite* job.
Usage	**tUniservBTGeneric** sends data to *DQ Batch Suite* and starts the specified *DQ Batch Suite* job. When the execution is finished, the output data of the job is returned to *Data Quality Service Hub Studio* for further processing.	
Limitation	To use **tUniservBTGeneric**, the Uniserv software *DQ Batch Suite* must be installed. Please note the following: • The job must be configured and executable in the *DQ Batch Suite*. • The user must have the authority to execute the *DQ Batch Suite* job. • The *DQ Batch Suite* job may only have one line. • The files defined in the functions **Input** and **Output** must possess the record format delimiter. • Input and output data must be provided in the UTF-8 character set.	

Scenario: Execution of a Job in the *Data Quality Service Hub Studio*

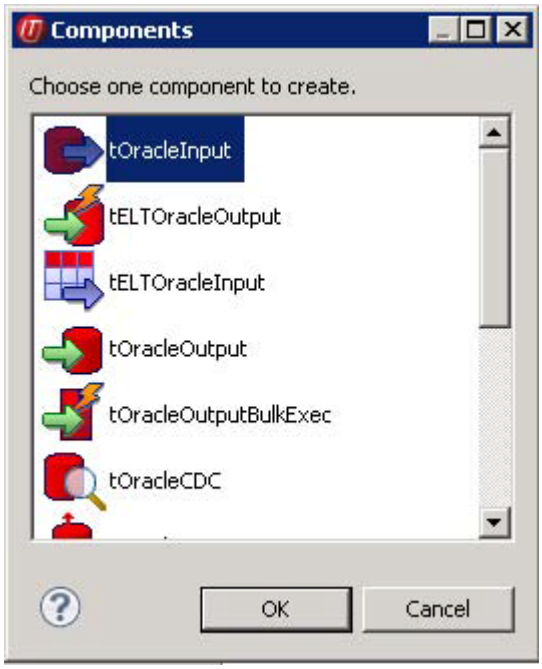
This scenario describes a *DQ Batch Suite* job which execution results are processed in the *Data Quality Service Hub Studio*. The input source for the job is provided by the *Data Quality Service Hub Studio*.

The job was completely defined in the *DQ Batch Suite* and saved under the name *"BTGeneric_Sample"*. In the function **Input**, the file *"btinput.csv"* was specified as the input file saved in the job directory and all fields were assigned. The file is not yet existent physically as it will only be provided by the *Data Quality Service Hub Studio*, so that the job cannot yet run.

In the *Data Quality Service Hub Studio*, the input source (here a table from an Oracle database) for this scenario was already saved in the **Repository**, so that all schema metadata is available.

1. In the **Repository** view, expand the **Metadata** node and the directory in which you saved the source. Then drag this source into the design workspace.

   The dialog box below appears.

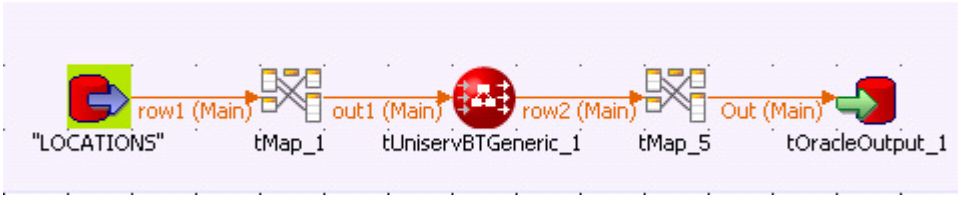


2. Select **tOracleInput** and then click **OK** to close the dialog box.

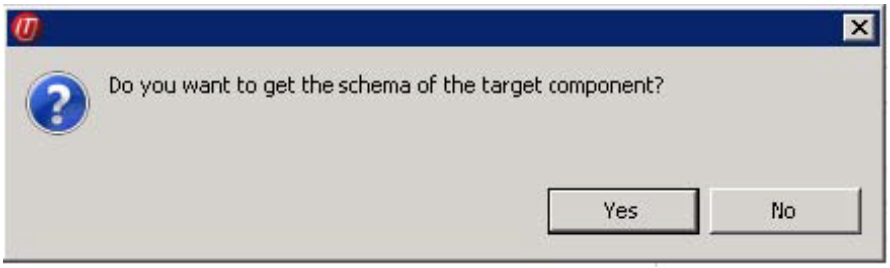
   The component is displayed in the workspace. The table used in this scenario is called *LOCATIONS*.

3. Drag the following components from the **Palette** into the design workspace: two **tMap** components, **tOracleOutput** and **tUniservBTGeneric**.

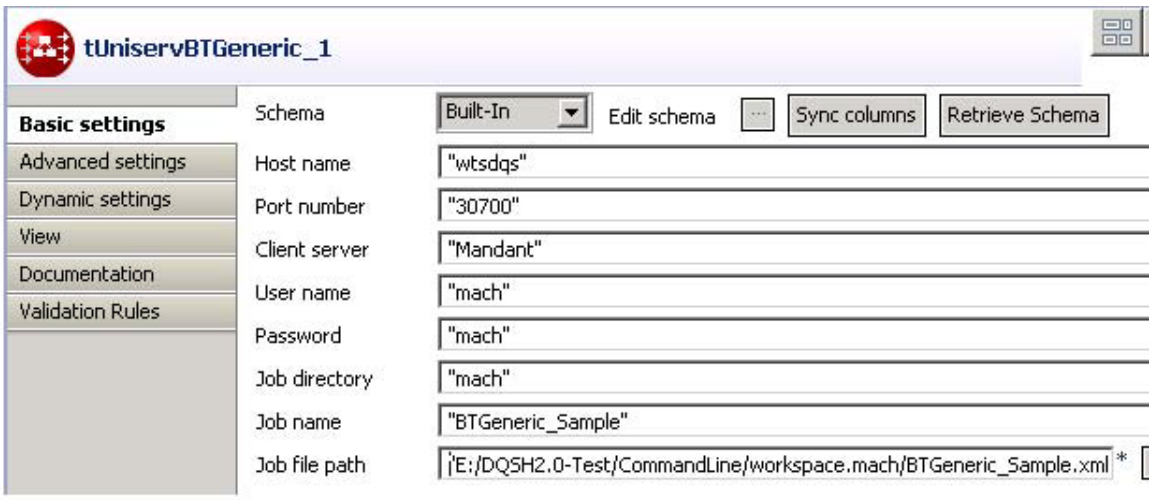
4. Connect the components via **Row** > **Main**.



   During the process, accept the schema from **tUniservBTGeneric** by clicking **Yes** in the validation window.



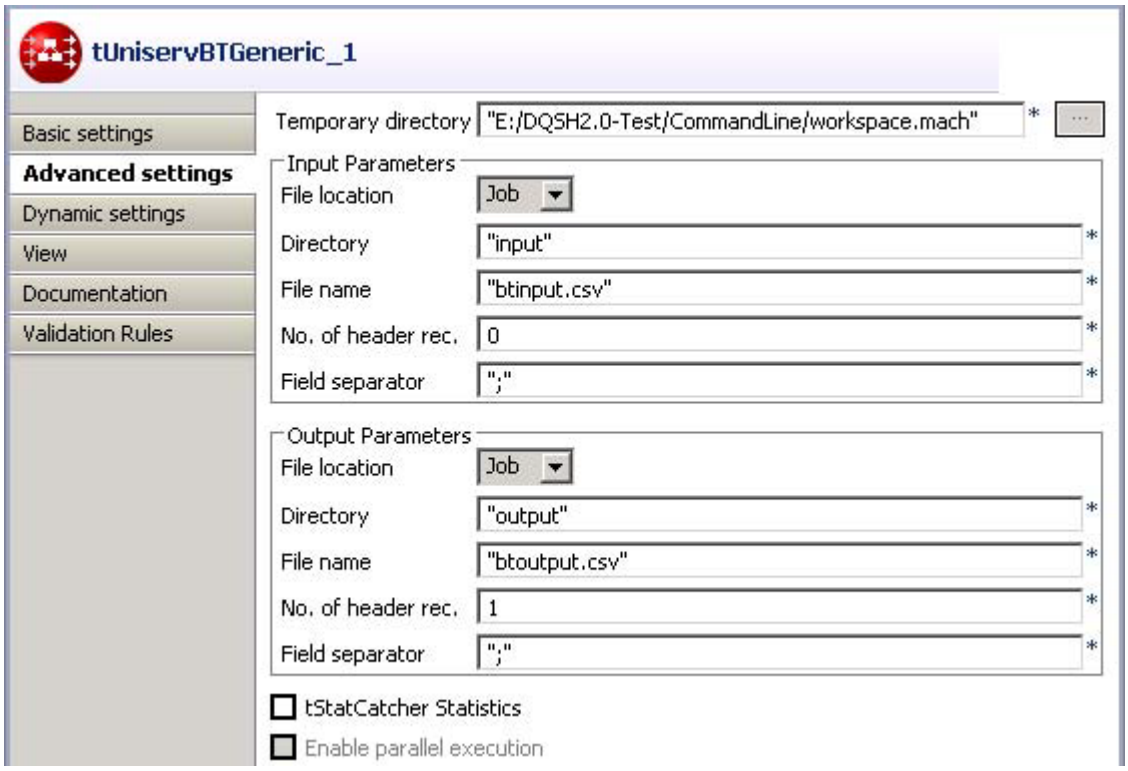
5. Double-click **tUniservBTGeneric** to open its **Basic Settings** view.



6. Enter the connection data for the *DQ Batch Suite* job. Note that the absolute path must be entered in the field **Job File Path**.

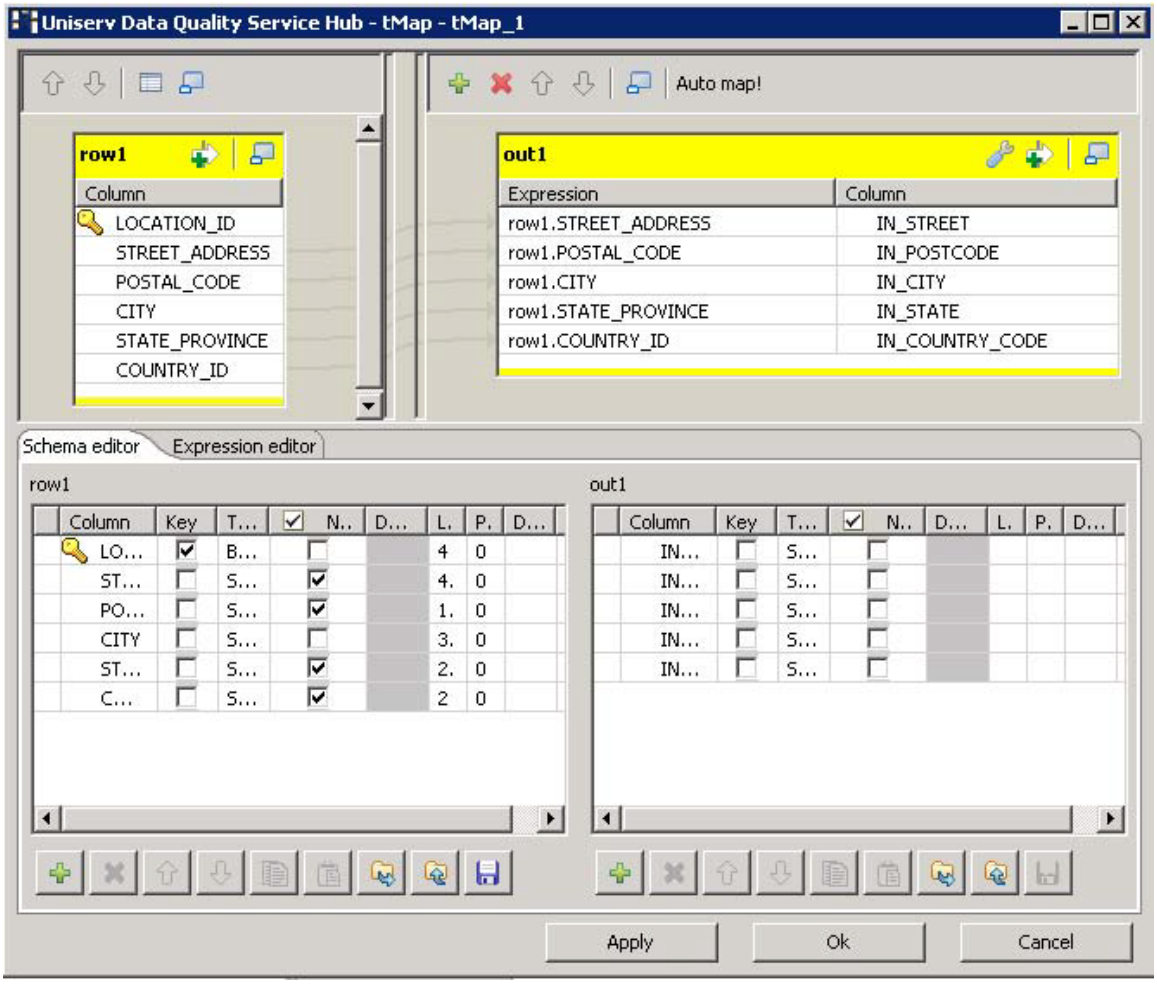
7. Click **Retrieve Schema** to automatically create a schema for **tUniservBTGeneric** from the input and output definitions of the *DQ Batch Suite* job and automatically fill in the fields in the **Advanced Settings**.

8. Check the details in the **Advanced Settings** view. The definitions for input and output must be defined exactly the same as the *DQ Batch Suite* job. If necessary, adapt the path for the temporary files.



9. Double-click **tMap_1** to open the schema mapping window. On the left is the structure of the input source, on the right is the schema of **tUniservBTGeneric** (and thus the input for the *DQ Batch Suite* job). At the bottom is the **Schema Editor**, where you can find the attributes of the individual columns and edit them.

10. Assign the columns of the input source to the respective columns of **tUniservBTGeneric**. For this purpose, select a column of the input source and drag it onto the appropriate column on the right side.



Click **OK** to close the dialog box.

11. Then define how to process the execution results of the job, including which components will be used.

12. Before starting the Job, make sure that all path details are correct, the server of the *DQ Batch Suite* is running and that you are able to access the job.

tUniservRTConvertName

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservRTConvertName properties

Component family	Data quality	
Function	**tUniservRTConvertName** analyzes the name line against the context. For individual persons, it divides the name line into segments (name, first name, title, name prefixes, name suffixes, etc.) and creates the address key. The component recognizes company or institution addresses and is able to provide the form of the organization separately. It also divides lines that contain information on several persons to separate lines and is able to recognize certain patterns that do not belong to the name information in the name line (customer number, handling notes, etc.) and remove them or move them to special memo fields.	
Purpose	**tUniservRTConvertName** provides the basis for a uniform structuring and population of person and company names in the database as well as the personalized salutation.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
	Host name	Server host name between double quotation marks.
	Port	Listening port number of the server between double quotation marks.
	Service	The service type/name is `"cname_d"` by default. Enter a new name if necessary (e.g. due to service suffix), between double quotation marks. Available services: Germany `"cname_d"` Italy `"cname_i"` Austria `"cname_a"` Netherlands `"cname_nl"` Switzerland `"cname_ch"` Belgium `"cname_b"` France `"cname_f"` Spain `"cname_e"`
	Use rejects	Select this option to separately output data sets from a certain result class of the onward name analysis. Enter the respective result class in the field **if result class is greater or equal to**. If this option is not selected, the sets are still output via the **Main** connection even if the analysis failed. If the option is selected, but the **Rejects** connection is not established, the sets are simply sorted out when the analysis failed.
Advanced settings	*Analysis Configuration*	For detailed information, please refer to the Uniserv user manual *convert-name*.
	Output Configuration	For detailed information, please refer to the Uniserv user manual *convert-name*.

	Configuration of not recognized input	For detailed information, please refer to the Uniserv user manual *convert-name*.
	Configuration of free fields	For detailed information, please refer to the Uniserv user manual *convert-name*.
	Cache Configuration	For detailed information, please refer to the Uniserv user manual *convert-name*.
Usage	**tUniservRTConvertName** provides the basis for a uniform structuring and population of person and company names in the database as well as the personalized salutation.	
Limitation	To use **tUniservRTConvertName**, the Uniserv software *convert-name* must be installed.	

Scenario: Analysis of a name line and assignment of the salutation

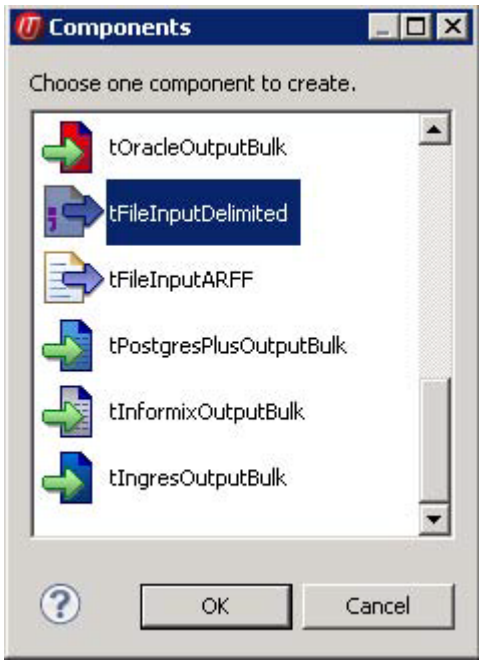
This scenario describes a batch job that analyzes the person names in a file and assigns them a salutation.

The input file for this scenario is already saved in the **Repository**, so that all schema metadata is available.

Please observe that the data from the input source must all be related to the same country.

1. In the **Repository** view, expand the **Metadata** node and the directory in which the file is saved. Then drag this file into the design workspace.

   The dialog box below appears.

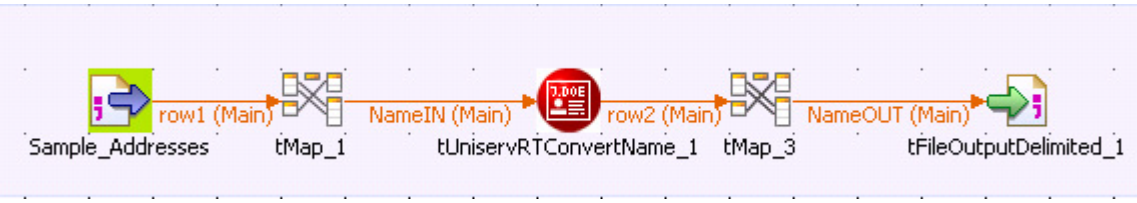


2. Select **tFileInputDelimited** and then click **OK** to close the dialog box.

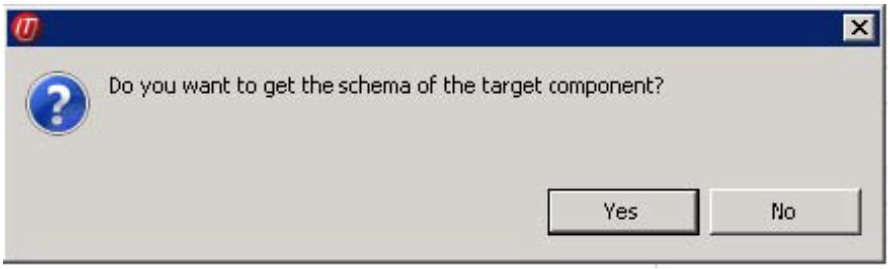
   The component is displayed in the workspace. The file used in this scenario is called *SampleAddresses.*.

3. Drag the following components from the **Palette** into the design workspace: two **tMap** components, **tUniservRTConvertName,** and **tFileOutputDelimited.**.

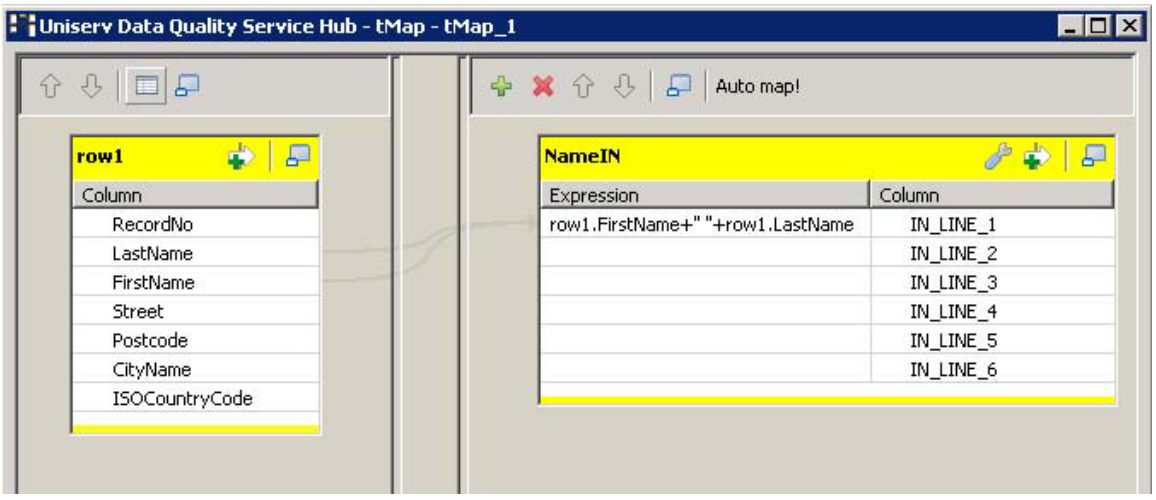
4. Connect the components via **Row** > **Main**.



During the process, accept the schema from **tUniservRTConvertName** by clicking **Yes** in the validation window.



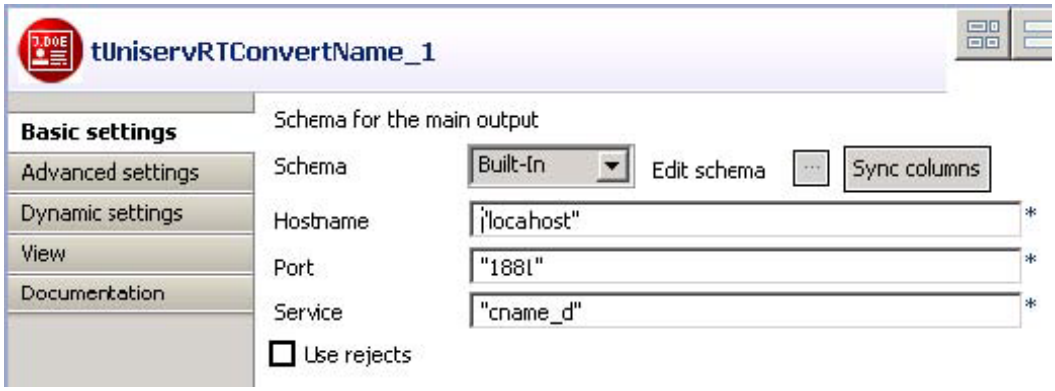
5. Double-click **tMap_1** to open the schema mapping window. On the left is the structure of the input file, on the right is the schema of **tUniservRTConvertName**. At the bottom lies the **Schema Editor**, where you can find the attributes of the individual columns and edit them.



6. Assign the columns of the input source to the respective columns of **tUniservRTConvertName**. For this purpose, select a column of the input source and drag it onto the appropriate column on the right side. If fields from the input file are to be passed on to the output file, like the address fields or IDs, you have to define additional fields.

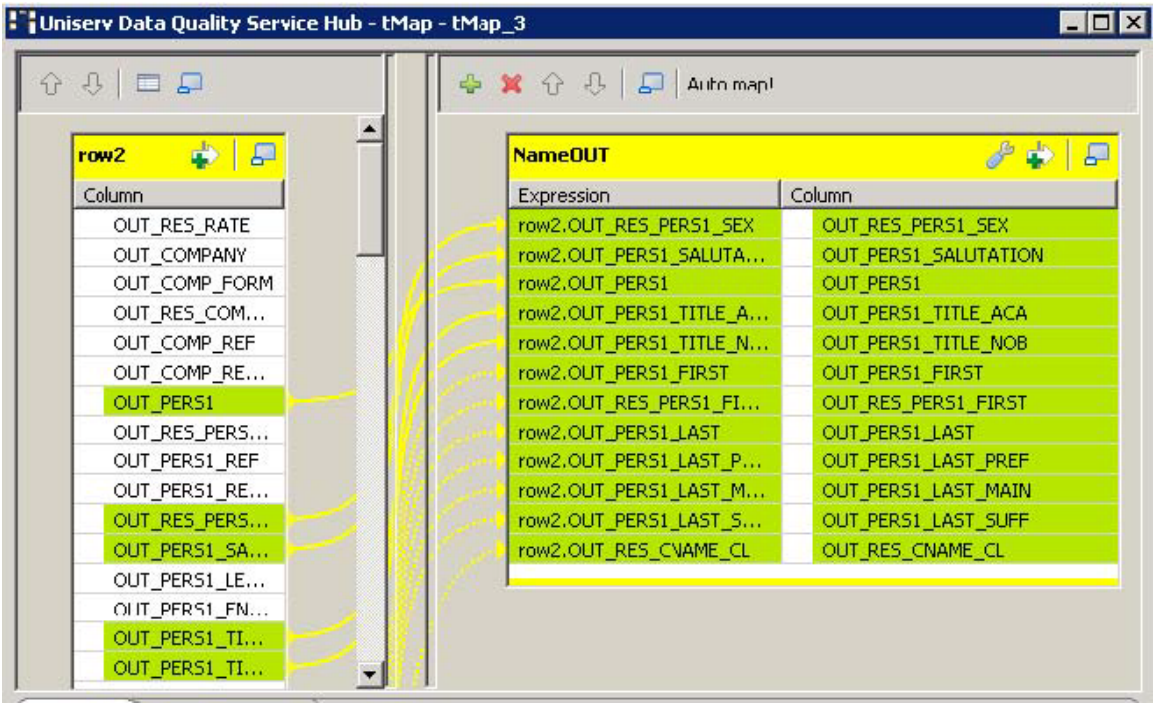
7. Click **OK** to close the dialog box.

8. Double-click **tUniservRTConvertName** to open its **Basic Settings** view.



9. Fill in the server information and specify the country-specific service.

10. Double-click **tMap_3** to open the mapping window. On the left is the schema of **tUniservRTConvertName** and on the right is the schema of the output file.



11. Click **OK** to close the window.

12. Double-click **tFileOutputDelimited** and enter the details for the output file.

tUniservRTMailBulk

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservRTMailBulk properties

Component family	Data quality	
Function	**tUniservRTMailBulk** creates an index pool for *mailRetrieval* with predefined input data.	
Purpose	**tUniservRTMailBulk** prepares the index pool for duplicate search.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
	Host name	Server host name between double quotation marks.
	Port	Listening port number of the server between double quotation marks.
	Service	The service name is `"mail"` by default. Enter a new name if necessary (e.g. due to service suffix), between double quotation marks.
Advanced settings	*Uniserv Parameters*	For detailed information, please refer to the Uniserv user manual *mailRetrieval*.
	tStatCatcher Statistics	Select this check box to collect log data at the Job and the component levels.
Usage	**tUniservRTMailBulk** prepares the index pool for duplicate search.	
Limitation	To use **tUniservRTMailBulk**, the Uniserv software *mailRetrieval* must be installed. An input component and a map are needed to read the address from the database or a file. The component does not have an output connection.	

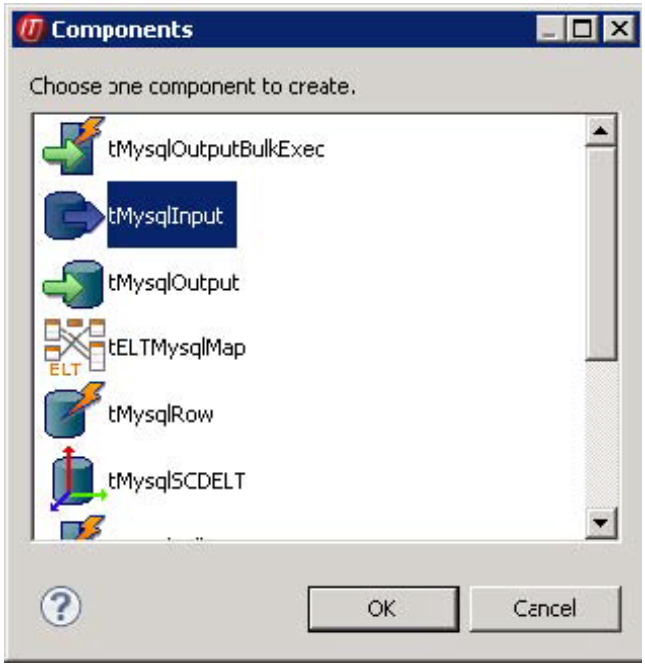
Scenario: Creating an index pool

This scenario describes a batch job that loads the address list of an SQL database into the index pool.

The database for this scenario is already saved in the **Repository**, so that all schema metadata is available.

1. In the **Repository** view, expand the **Metadata** node and the directory in which the database is saved. Then drag this database into the design workspace.

   The dialog box below appears.

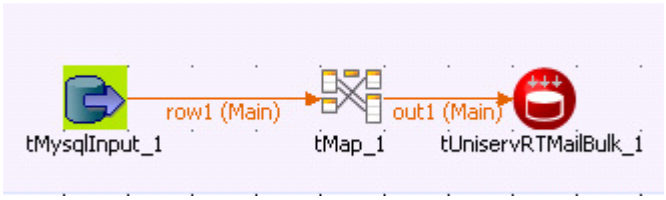


2. Select **tMysqlInput** and then click **OK** to close the dialog box.

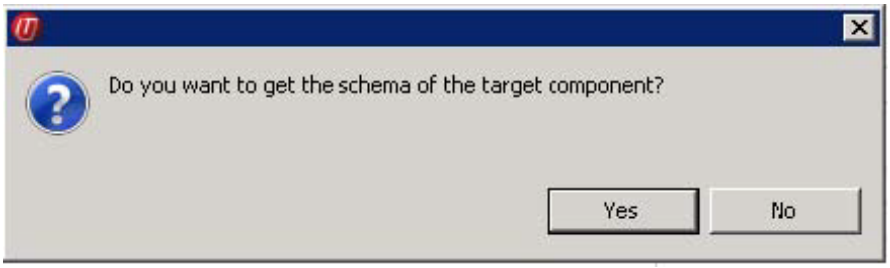
   The component is then displayed in the workspace.

3. Drag the following components from the **Palette** into the design workspace: **tMap** and **tUniservRTMailBulk**.

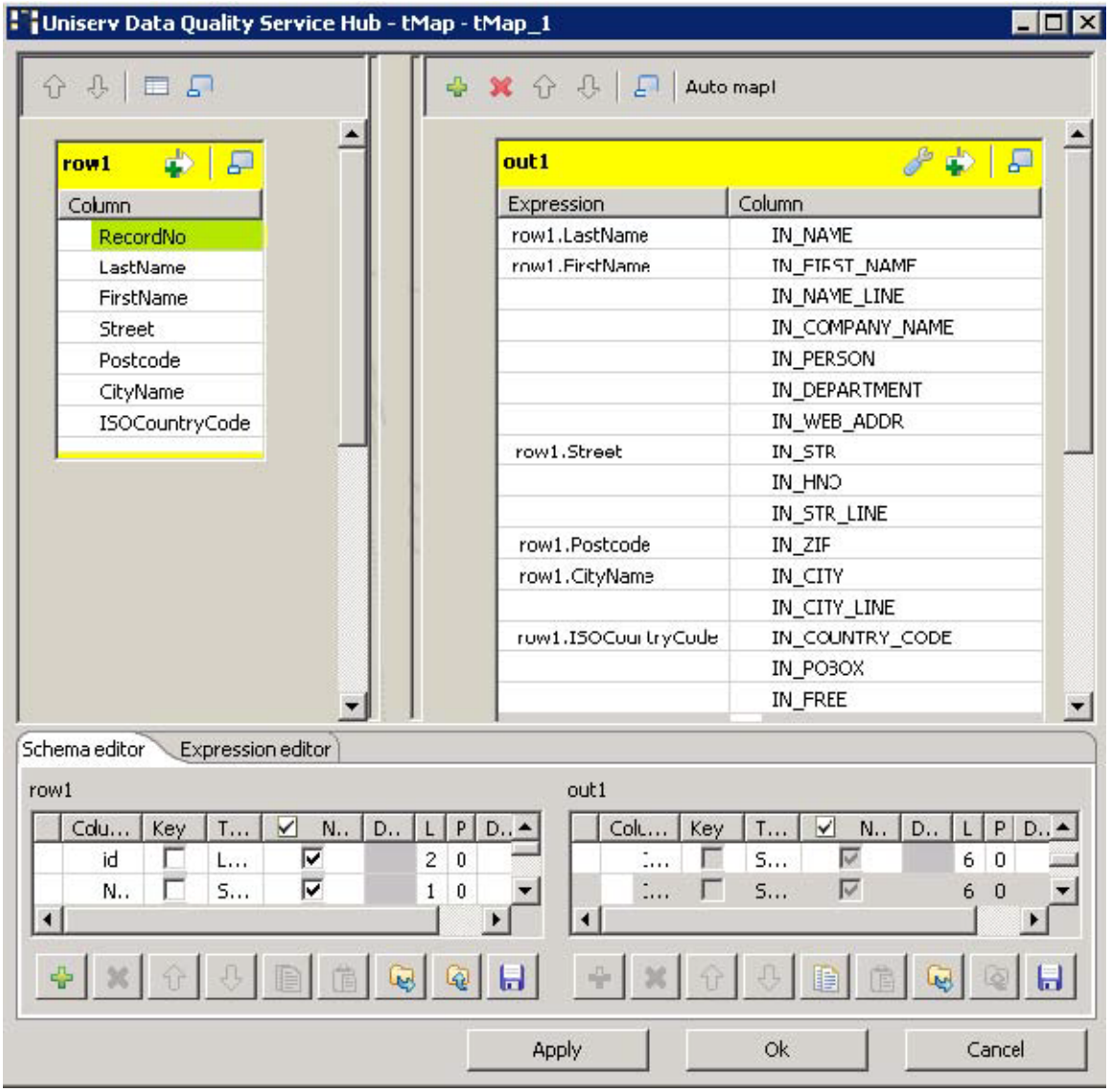
4. Connect the components via **Row** > **Main**.



   During the process, accept the schema from **tUniservRTMailBulk** by clicking **Yes** in the validation window.



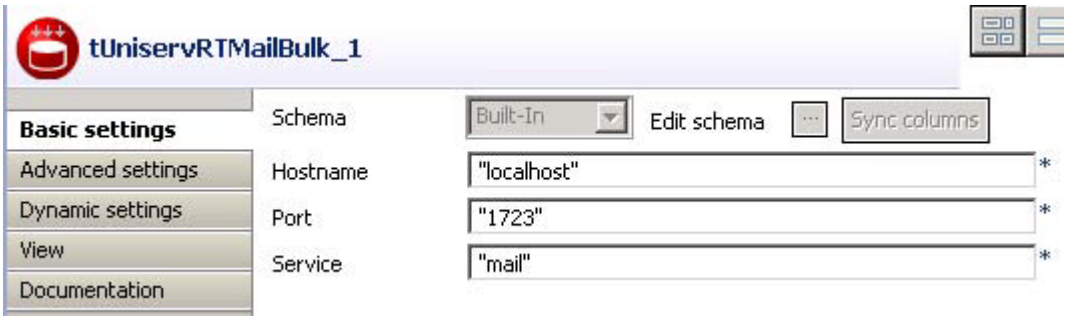
5. Double-click **tMap_1** to open the schema mapping window. On the left is the schema of the database file and on the right is the schema of **tUniservRTMailBulk**. At the bottom is displayed the **Schema Editor**, where you can find the attributes of the individual columns and edit them.



6. Assign the columns of the input source to the respective columns of **tUniservRTMailBulk**. For this purpose, select a column of the input source and drag it onto the appropriate column on the right side. The meaning of the individual arguments is described in the Uniserv user manual *mailRetrieval*.

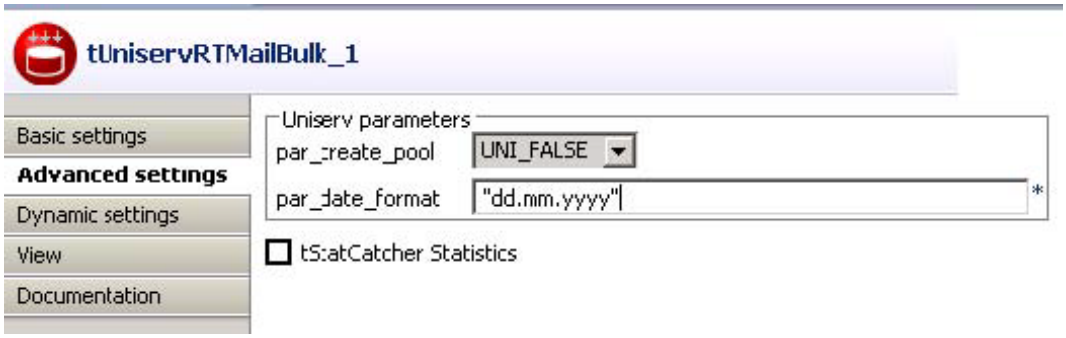
7. Click **OK** to close the window.

8. Double-click **tUniservRTMailBulk** to open its **Basic Settings** view.



9. Fill in the server information and specify the service.

10. Select **Advanced Settings** to adapt the server parameters.


tUniservRTMailBulk_1
Basic settings
Advanced settings
Dynamic settings
View
Documentation
Uniserv parameters
par_create_pool
UNI_FALSE
par_date_format
"dd.mm.yyyy"
tStatCatcher Statistics

tUniservRTMailOutput

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservRTMailOutput properties

Component family	Data Quality	
Function	**tUniservRTMailOutput** updates the index pool that is used for duplicate search..	
Purpose	**tUniservRTMailOutput** keeps the index pool synchronized.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the **Repository**.
	Host name	Server host name between double quotation marks.
	Port	Listening port number of the server between double quotation marks.
	Service	The service name is `"mail"` by default. Enter a new name if necessary (e.g. due to service suffix), between double quotation marks.
	Action on data	Operations that can be made on the index pool. **Insert**: inserts a new record in the index pool. This request will fail if the record with the given reference already exists in the index pool. **Update**: updates an existing record in the index pool. This request will fail if the record with the given reference does not exist in the index pool. **Insert or update**: inserts a new record in the index pool. If the record with the given reference already exists, an update would be made. **Update or insert**: updates the record with the given reference. If the record does not exist in the index pool, a new record would be inserted. **Delete**: deletes the record with the given reference from the index pool.
Advanced settings	*Uniserv Parameters*	For detailed information, please refer to the Uniserv user manual *mailRetrieval*.
	tStatCatcher Statistics	Select this check box to collect log data at the Job and the component levels.
Usage	**tUniservRTMailOutput** updates the index pool and passes the input set on. The output is amended by the status of the operation. If the operation fails, an error message will be displayed.	
Limitation	To use **tUniservRTMailOutput**, the Uniserv software *mailRetrieval* must be installed. Before the first use of **tUniservRTMailOutput**, an index pool must be created. You can create the index pool with **tUniservRTMailBulk**.	

Related scenarios

For a related scenario, see section *Scenario: Adding contacts to the mailRetrieval index pool*.

tUniservRTMailSearch

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservRTMailSearch properties

Component family	Data quality	
Function	**tUniservRTMailSearch** searches for similar data based on the given input record.	
Purpose	**tUniservRTMailSearch** searches for duplicate values and adds additional data to each record.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the **Repository**.
	Host name	Server host name between double quotation marks.
	Port	Listening port number of the server between double quotation marks.
	Service	The service name is `"mail"` by default. Enter a new name if necessary (e.g. due to service suffix), between double quotation marks.
	Maximum of displayed duplicates (0 = All)	Enter the maximum number of duplicates to be displayed in the **Run** view. The default value is 0, which means that all duplicates will be displayed (up to 1000 duplicates can be displayed).
	Use rejects	Select this check box to set parameters based on which duplicate records should be added to the reject flow. Then set the: **Element**: Duplicate count. **Operator**: <, <=, =, >=, >. **Value**: Enter the number manually.
Advanced settings	*Uniserv Parameters*	For detailed information, please refer to the Uniserv user manual *mailRetrieval*.
	tStatCatcher Statistics	Select this check box to collect log data at the Job and the component levels.
Usage	**tUniservRTMailSearch** requires an input component and one or more output components.	
Limitation	To use **tUniservRTMailSearch**, the Uniserv software *mailRetrieval* must be installed. Before the first use of **tUniservRTMailSearch**, an index pool must be created. You can create the index pool with **tUniservRTMailBulk**.	

Scenario: Adding contacts to the mailRetrieval index pool

This scenario describes a batch job that adds contacts to the index pool of *mailRetrieval*. Before the addition, it must be checked whether these contacts already exist.

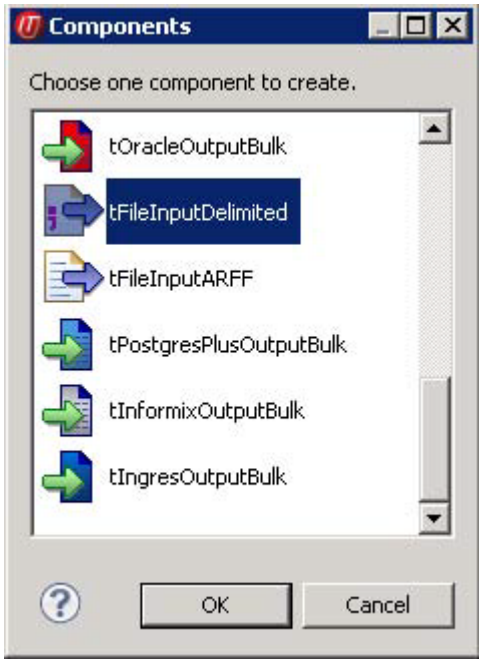
The input file for this scenario is already saved in the **Repository**, so that all schema metadata is available.

Please note that the data from the input source must be related to the same country.

Dropping and connecting the components

1. In the **Repository** view, expand the **Metadata** node and the directory in which the file is saved. Then drag this file into the design workspace.

   The dialog box below appears.



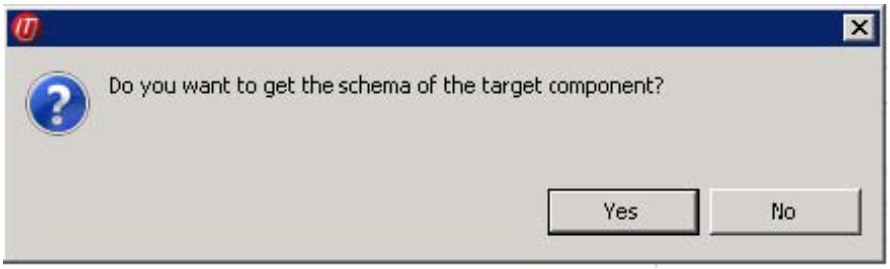
2. Select **tFileInputDelimited** and then click **OK** to close the dialog box.

   The component is displayed in the workspace.

3. Drag the following components from the **Palette** into the design workspace: two **tMap** components, **tUniservRTMailSearch** and **tUniservRTMailOutput** .

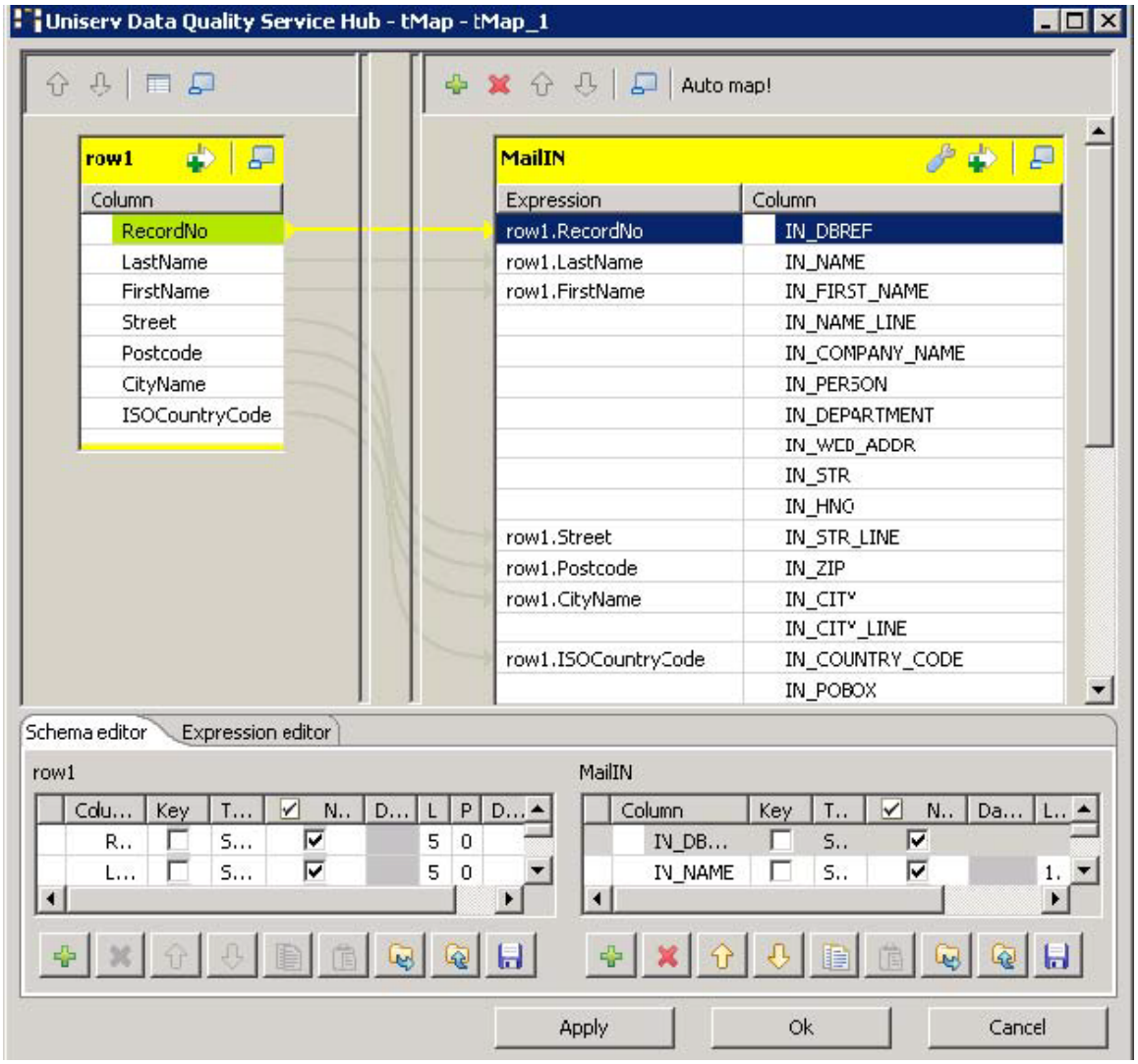
4. Connect the components via **Row** > **Main**.

   During the process, accept the schema from **tUniservRTMailSearch** by clicking **Yes** in the validation window.



Configuring the components

1. Double-click **tMap_1** to open the schema mapping window. On the left is the structure of the input file and on the right is the schema of **tUniservRTMailSearch**. At the bottom lies the **Schema Editor**, where you can find the attributes of the individual columns and edit them.

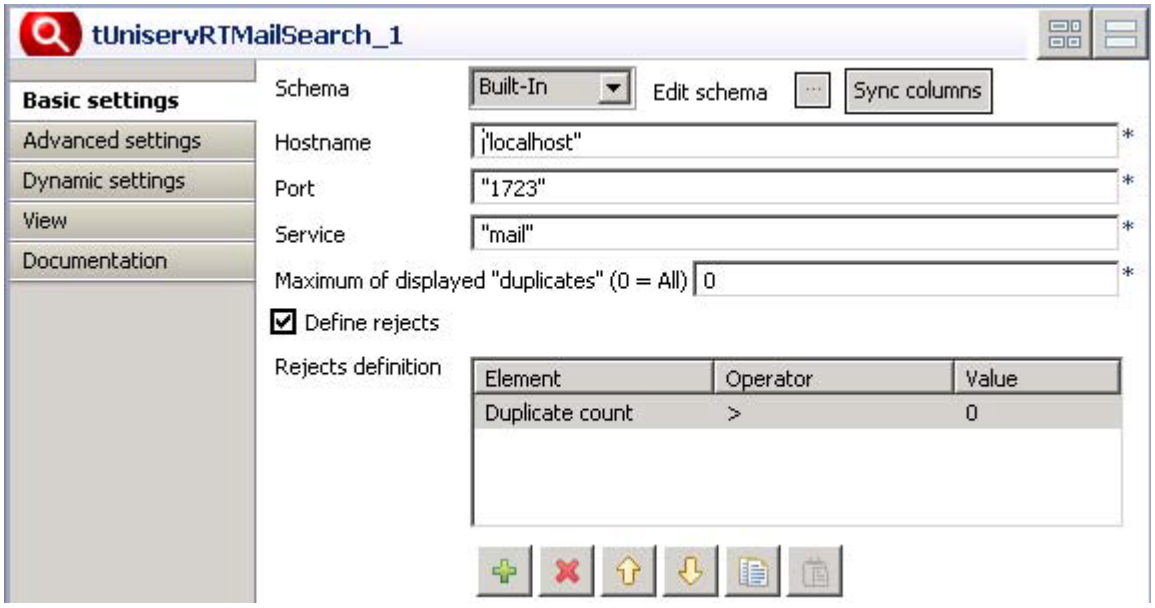


2. Assign the columns of the input file to the respective columns of **tUniservRTMailSearch**. For this purpose, select a column of the input source and drag it onto the appropriate column on the right side.

3. When your input list contains a reference ID, you should adopt it. In order to do so, create a new column *IN_DBREF* in the **Schema Editor** and connect it with your reference ID.

   Click **OK** to close the window.

4. Double-click **tUniservRTMailSearch** to open its **Basic settings** view.



5. Under **Maximum of displayed "duplicates"**, enter *0* to display all the duplicates.

   Select **Define rejects** to open the rejects definition window.

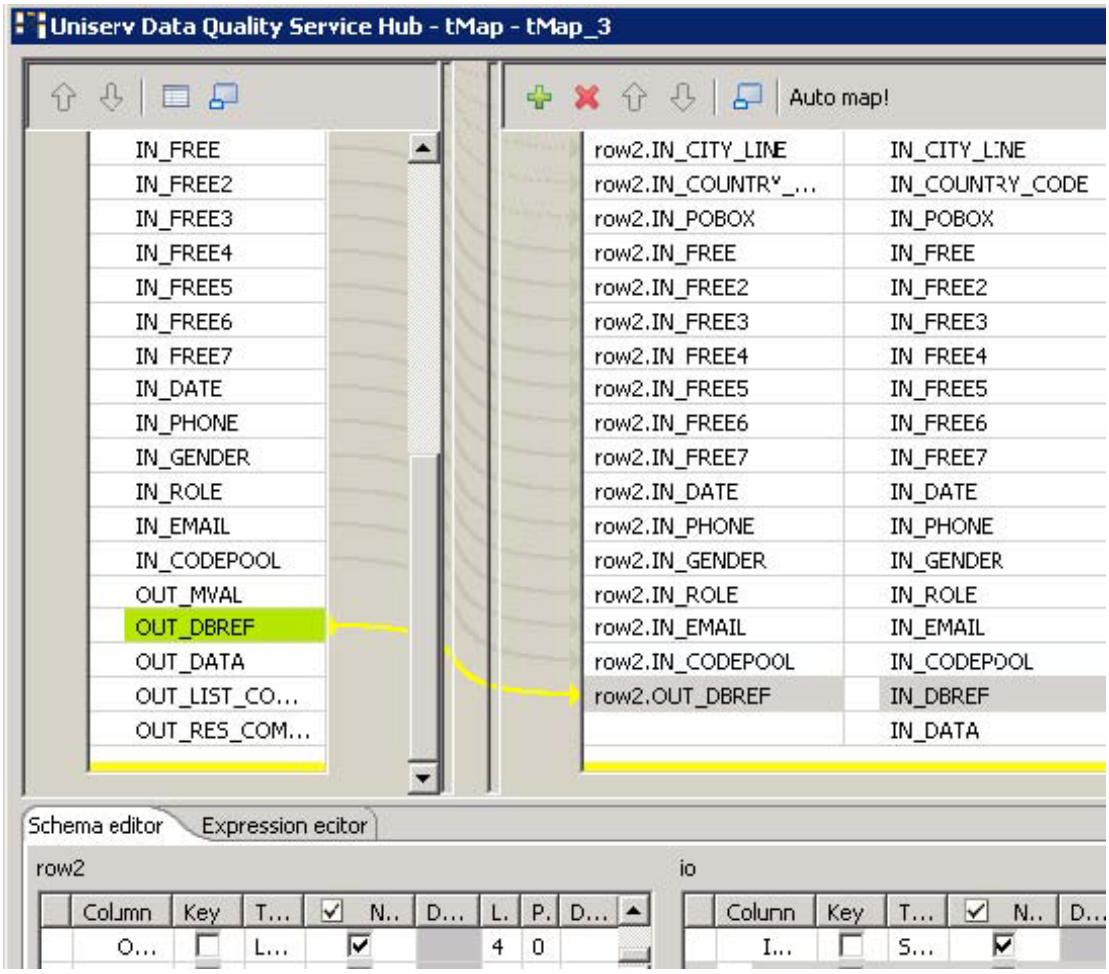
6. Click the **[+]** button to insert a new line in the window. Select *Duplicate count* under the element column, > under the operator column, and *0* under the value column. So all the existing contacts are disqualified and only the new contact will be added to the index pool.

7. Enter the **Advanced settings** view and check the parameters. Reasonable parameters are preset. Detailed information can be found in the manual *mailRetrieval*.

8. Double-click **tMap_3** to open schema mapping window. On the left is the schema of **tUniservRTMailSearch** and on the right is the schema of **tUniservRTMailOutput**.

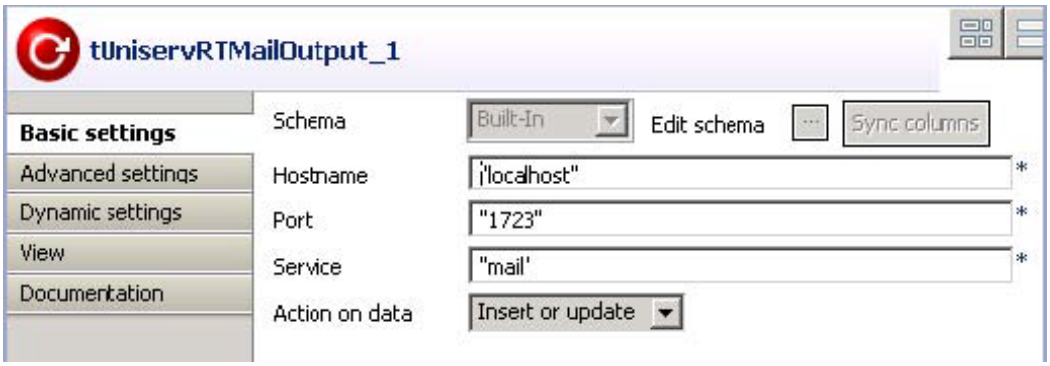
9. Click **Auto map!** to assign the fields automatically.

10. The only field that must be assigned manually is the reference ID. In order to do so, drag *OUT-DBREF* from the left side onto the field *IN_DBREF* on the right side.



Click **OK** to close the dialog box.

11. Double-click **tUniservRTMailOutput** to open the **Basic settings** view.



From the **Action on Data** list, select *Insert or update*. This way, all new contacts are added to the index pool.

tUniservRTPost

*This component will be available in the **Palette** of the studio on the condition that you have subscribed to the relevant edition of Data Quality Service Hub Studio.*

tUniservRTPost properties

Component family	Data quality	
Function	**tUniservRTPost** provides postal validation and correction of addresses, which is critical to improving the quality of addresses. This way, you will be more successful in personalized one-on-one marketing, reducing costs and increasing the efficiency and cost-effectiveness of address management in all the applications.	
Purpose	**tUniservRTPost** helps to improve the addresses quality, which is extremely important for CRM and e-business as it is directly related to postage and advertising costs.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the **Repository**.
	Host name	Server host name between double quotation marks.
	Port	Listening port number of the server between double quotation marks.
	Service	The service name is `"post"` by default. Enter a new name if necessary (e.g. due to service suffix), between double quotation marks.
	Use rejects	Select this check box to collect faulty addresses via the **rejects** connection. Usually they are the addresses with the post result class 5. Valid values for the result class are 1-5. The value must be between double quotation marks. If this check box is not selected, the faulty addresses are output via the **Main** connection. If the check box is selected but the **rejects** connection is not created, the faulty addresses are simply rejected.
	Use File for ambiguous results	Select the check box to define a file for writing the selection list to it. When an address cannot be corrected unambiguously, a selection list is created. This list can be further processed via the **AMBIGUITY** connection. All potential candidate results then run via this connection. The schema of this connection is preinitialized with the arguments of the dissolved selection list of the service *'post'*.
Advanced settings	*Uniserv Parameters*	Select this check box to define the corresponding parameters. For detailed information, please refer to the Uniserv user manual *International Postal Framework*.
	tStatCatcher Statistics	Select this check box to collect log data at the Job and the component levels.
	“Full address” selection list	Select the check box **Display** to show all the columns. Or, select the check box next to a particular column to show it alone. This option controls the content of the file for ambiguous addresses. Only selected columns would be written into the file.
Usage	**tUniservRTPost** requires an input set. Its postal validation will then be checked. In case of an unambiguous result, the corrected set will be output via the **Main** connection. If the address is	

	ambiguous, the potential candidates will be output via the **Ambiguity** connection. If an address was not found, it will be passed on via the **Reject** connection.
Limitation	To use **tUniservRTPost**, the Uniserv software *International Postal Framework* and the required post servers must be installed.

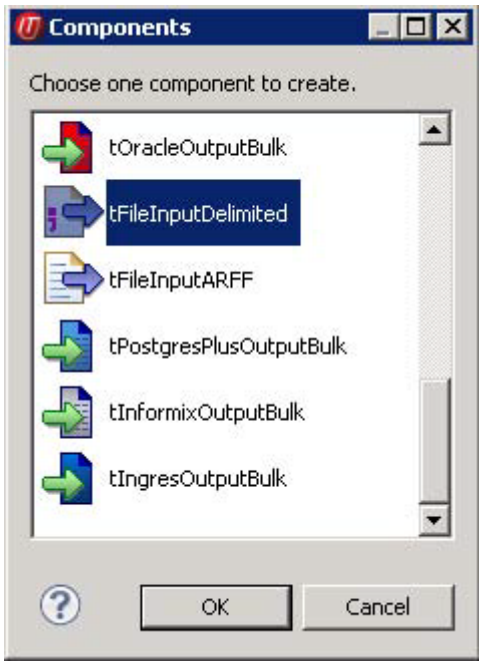
Scenario 1: Checking and correcting the postal code, city and street

This scenario describes a batch job that checks and corrects the addresses and postal codes from a file.

The input file for this scenario is already saved in the **Repository**, so that all schema metadata is available.

1. In the **Repository** view, expand the **Metadata** node and the directory in which the file is saved. Then drag this file into the design workspace.

   The dialog box below appears.



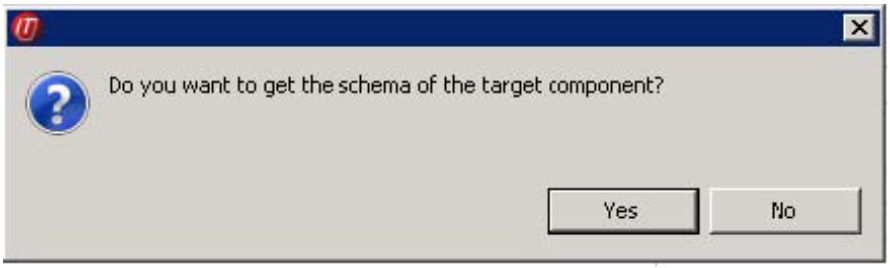
2. Select **tFileInputDelimited** and click **OK** to close the dialog box.

   The component is displayed in the workspace. The file used in this scenario is called *SampleAddresses*. It contains address data that comes with a country code. The street and house number are saved together in the street field, while postal code and city are respectively saved in separate fields.

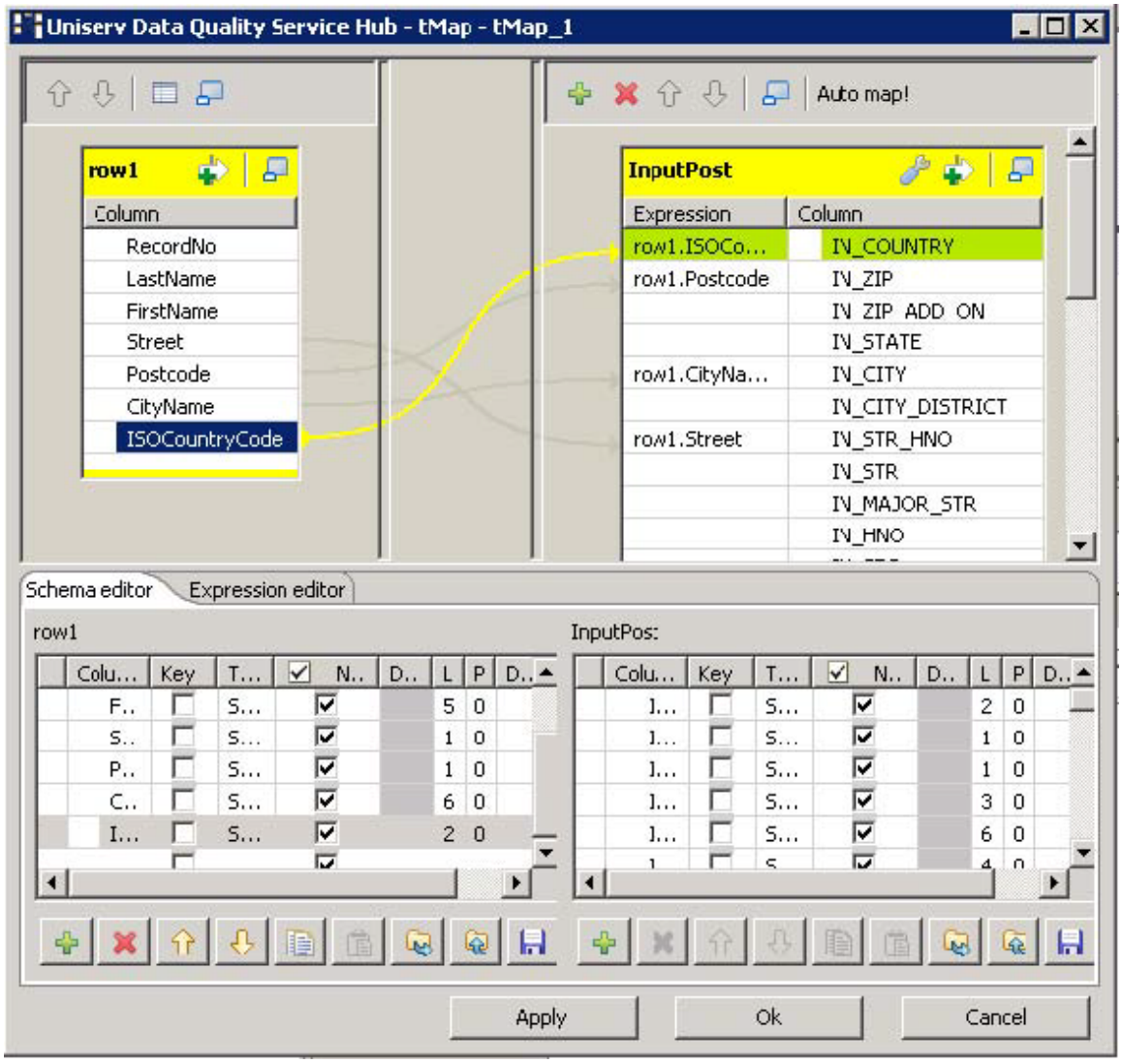
3. Drag the following components from the **Palette** into the design workspace: two **tMap** components, **tUniservRTPost** and **tFileOutputDelimited** .

4. Connect the components via **Row** > **Main**.

5. During the process, accept the schema from **tUniservRTPost** by clicking **Yes** in the validation window.



6. Double-click **tMap_1** to open the schema mapping window. On the left is the structure of the input file and on the right is the schema of **tUniservRTPost**. At the bottom is displayed the **Schema Editor**, where you can find the attributes of the individual columns and edit them.

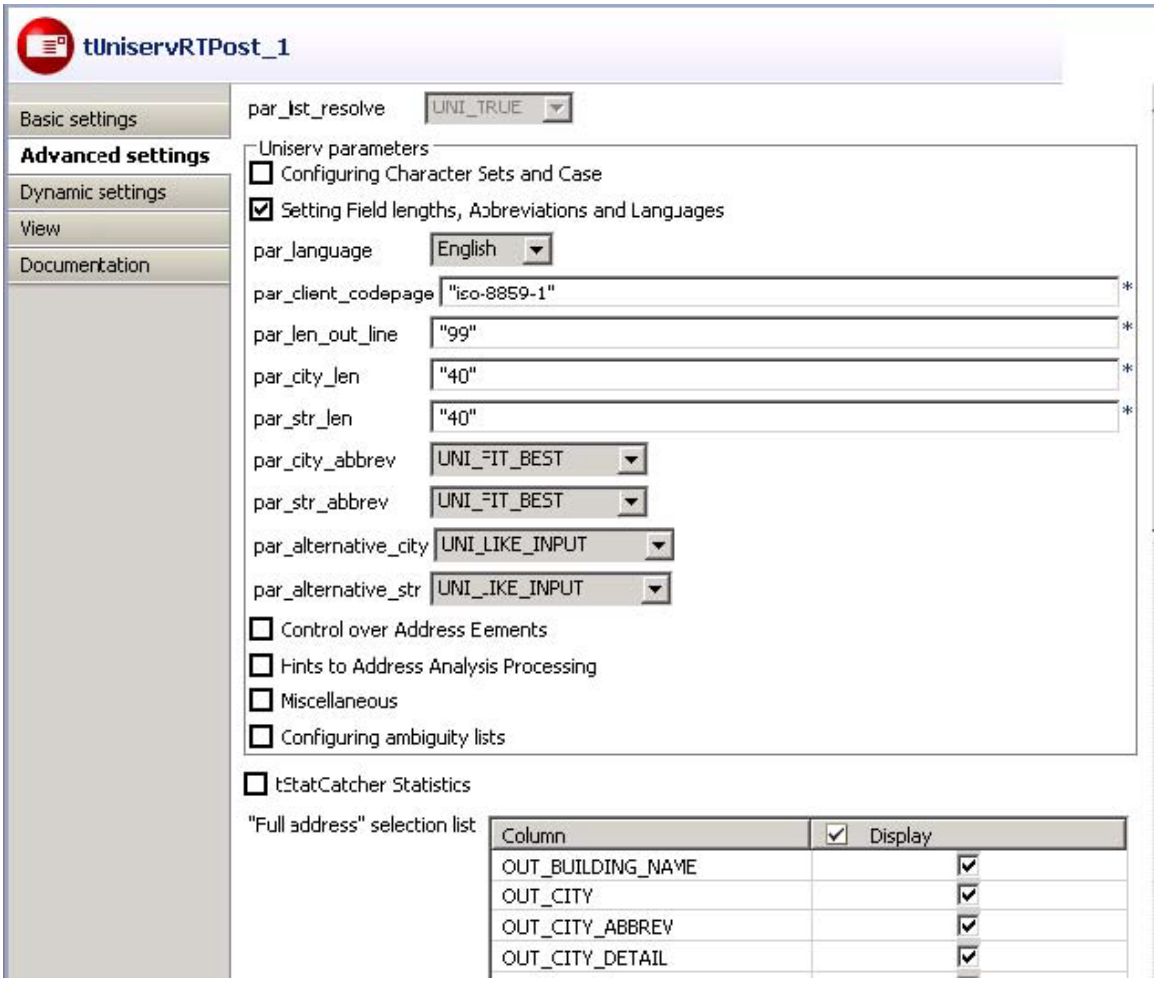


7. Assign the columns of the input file to the respective columns of **tUniservRTPost**. For this purpose, select a column of the input source and drag it onto the appropriate column on the right side. If fields from the input file are to be passed on to the output file, e.g. the names or the IDs, additional fields must be defined.

When assigning the fields, note that street and house number can either be saved together in the street column or respectively in separate fields. If your data list does not have a country code but the addresses are from the same country, the relevant ISO-country code should be manually entered between double quotation marks in the column *IN_COUNTRY*. If you have an international data list without country code, just leave the column *IN_COUNTRY* empty. For detailed information, please refer to the Uniserv user manual *International Postal Framework*.

8. Click **OK** to close the window.

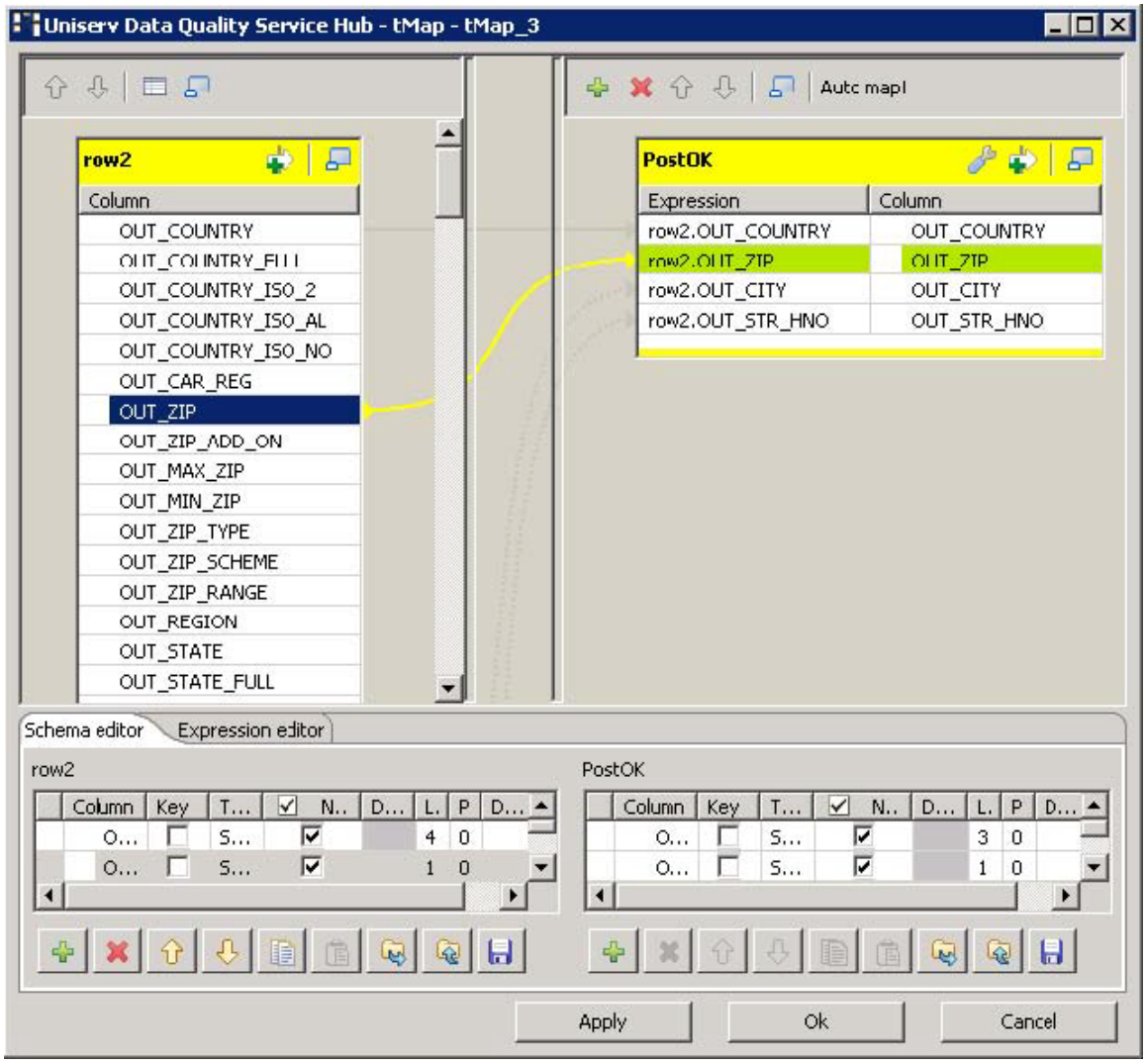
9. Double-click **tUniservRTPost** and enter its **Advanced settings** view.



10. Change the parameters and field lengths if necessary and select the output fields.

Make sure sufficient field length is defined. For detailed information, please refer to the Uniserv user manual *International Postal Framework*.

11. Double-click **tMap_3** to open schema mapping window. On the left is the schema of **tUniservRTPost** and on the right is the schema of the output file.



12. Click **OK** to close the dialog box.

13. Double-click **tFileOutputDelimited** to enter the details for the output file.

Scenario 2: Checking and correcting the postal code, city and street, as well as rejecting the unfeasible

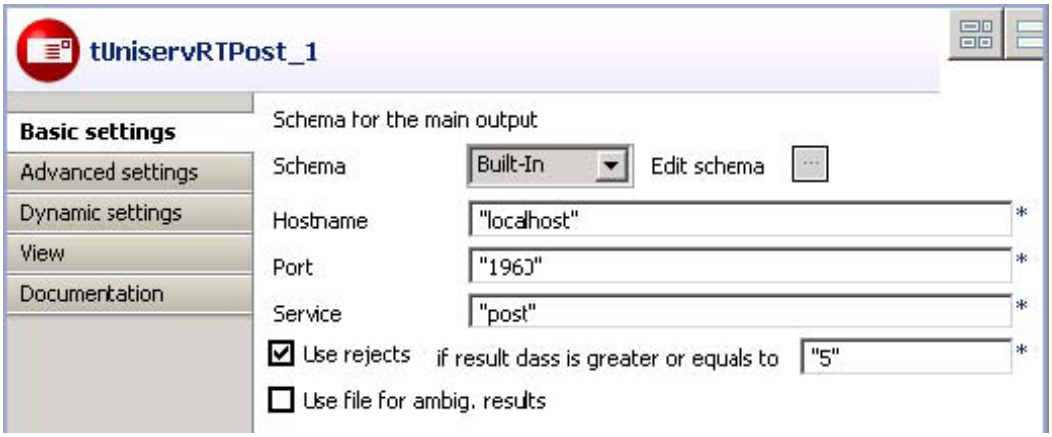
This scenario is closely related to the one above. But the difference is that, the addresses that cannot be assigned are written into a separate file for manual checking. Additionally, to write ambiguous addresses in a separate file, the procedure is the same as described here.

1. Create a job as described in the previous scenario.

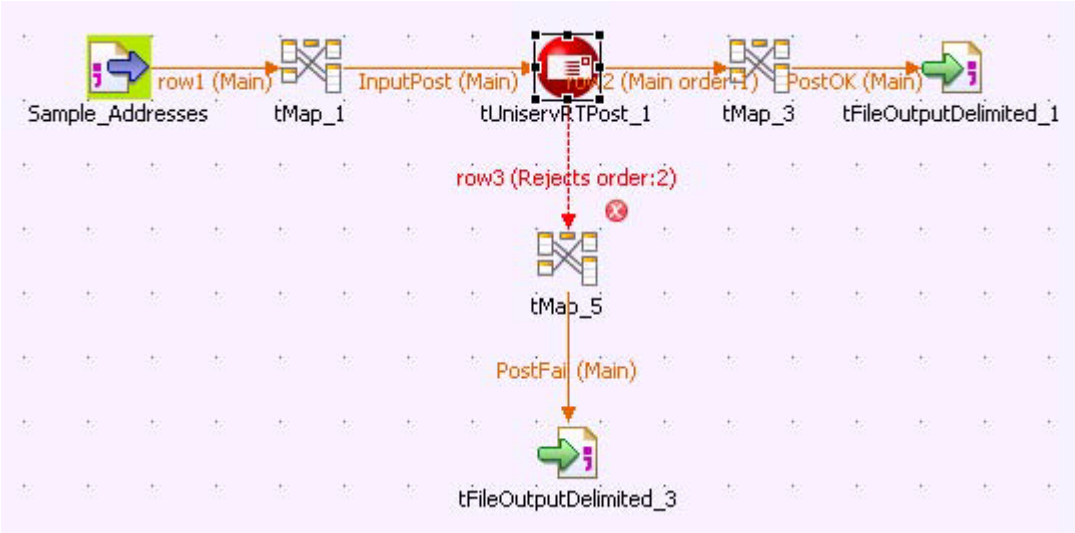
2. Drag the following additional components from the **Palette** into the design workspace: **tMap** and **tFileOutputDelimited**.

3. Double-click **tUniservRTPost** to open its **Basic settings** view.

4. Select the **Use rejects** check box and enter *"5"* in the field **if result class greater or equals to**. This is the result class from the check of postal codes in addresses, which contain too few or unfeasible data.



5. Connect **tUniservRTPost** with **tMap_5** via **Row** > **Rejects**.
6. Connect **tMap** with **tFileOutputDelimited** via **Row** > **Main**.



7. Define the fields for the output file in the mapping window.

Databases - traditional components

This chapter describes connectors for the most popular and traditional databases. These connectors cover various needs, including: opening connections, reading and writing tables, committing transactions as a whole, as well as performing rollback for error handling. Over 40 RDBMS are supported. These components can be found in the **Databases** family in the **Palette** in the **Integration** perspective of the **Talend** Studio.

Other types of database connectors, such as connectors for Appliance/DW databases and database management, are documented in *Databases - appliance/datawarehouse components* and *Databases - other components*.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAccessBulkExec

tAccessBulkExec properties

The **tAccessOutputBulk** and **tAccessBulkExec** components are generally used together to output data to a delimited file and then to perform various actions on the file in an Access database, in a two step process. These two steps are fused together in the **tAccessOutputBulkExec** component, detailed in a separate section. The advantage of using a two step process is that it makes it possible to carry out transformations on the data before loading it in the database.

Component family	Databases/Access	
Function	This component executes an Insert action on the data provided.	
Purpose	As a dedicated component, **tAccessBulkExec** offers gains in performance when carrying out Insert operations in an Access database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data is stored centrally.
	Use an existing connection	Select this check box and select the appropriate **tAccessConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	DB version	Select the version of your database.
	Database	Type in the directory where your database is stored.
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted.

	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist already for the insert operation to succeed.
	Local filename	Browse to the delimited file to be loaded into your database.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic** settings.
	Include header	Select this check box to include the column header.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tAccessOutputBulk** component. Used together, they can offer gains in performance while feeding an Access database.	

Related scenarios

For use cases in relation with **tAccessBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*
- section *Scenario: Inserting data in MySQL database*

tAccessCommit

tAccessCommit Properties

This component is closely related to **tAccessConnection** and **tAccessRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Access	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tAccessConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tAccessCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Access components, especially with **tAccessConnection** and **tAccessRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAccessConnection** and **tAccessRollback**. It usually does not make much sense to use one of these without using a **tAccessConnection** component to open a connection for the current transaction.

For **tAccessCommit** related scenario, see section *tMysqlConnection*

tAccessConnection

tAccessConnection Properties

This component is closely related to **tAccessCommit**, **tAccessInput** and **tAccessOutput**. It usually does not make much sense to use one of these without using a **tAccessConnection** component to open a connection for the current transaction.

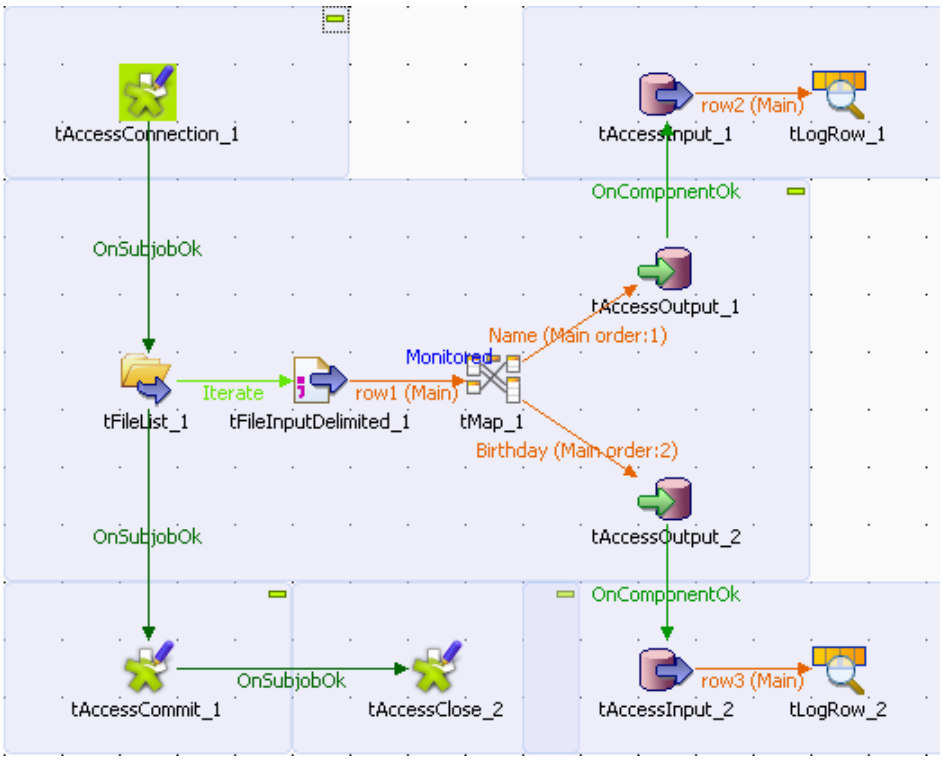
Component family	Databases/Access	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Access 2003 or later versions.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating.
Usage	This component is to be used along with Access components, especially with **tAccessCommit** and **tAccessOutput** components.	
Limitation	n/a	

Scenario: Inserting data in parent/child tables

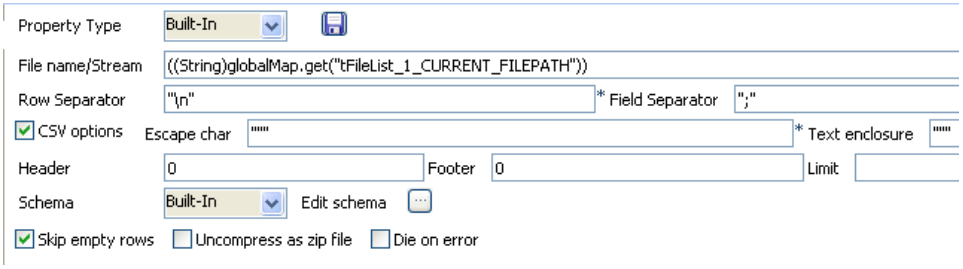
The following Job is dedicated to advanced database users, who want to carry out multiple table insertions using a parent table *Table1* to generate two child tables: *Name* and *Birthday*.

- In Access 2007, create an Access database named *Database1*.
- Once the Access database is created, create a table named *Table1* with two column headings: *Name* and *Birthday*.

Back into *Talend Open Studio for Big Data*, the Job requires twelve components including **tAccessConnection**, **tAccessCommit**, **tAccessInput**, **tAccessOutput** and **tAccessClose**.

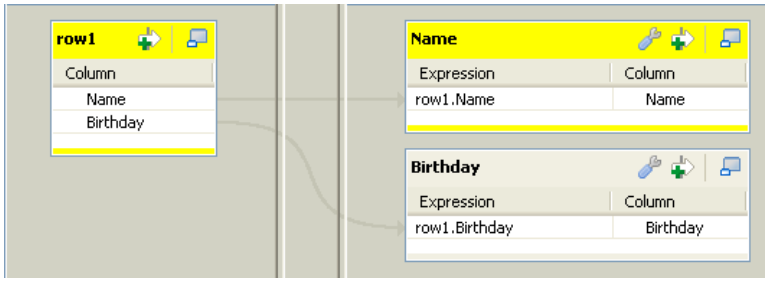


- Drop the following components from the **Palette** to the design workspace: **tFileList**, **tFileInputDelimited**, **tMap**, **tAccessOutput** (x2), **tAccessInput** (x2), **tAccessCommit**, **tAccessClose** and **tLogRow** (x2).
- Connect the **tFileList** component to the input file component using an **Iterate** link. Thus, the name of the file to be processed will be dynamically filled in from the **tFileList** directory using a global variable.
- Connect the **tFileInputDelimited** component to the **tMap** component and dispatch the flow between the two output Access components. Use a **Row** link for each of these connections representing the main data flow.
- Set the **tFileList** component properties, such as the directory where files will be fetched from.
- Add a **tAccessConnection** component and connect it to the starter component of this Job. In this example, the **tFileList** component uses an **OnComponentOk** link to define the execution order.
- In the **tAccessConnection** Component view, set the connection details.
- In the **tFileInputDelimited** component's **Basic settings** view, press **Ctrl+Space** bar to access the variable list. Set the **File Name** field to the global variable: *tFileList_1.CURRENT_FILEPATH*. For more information about using variables, see *Talend Open Studio for Big Data User Guide*.

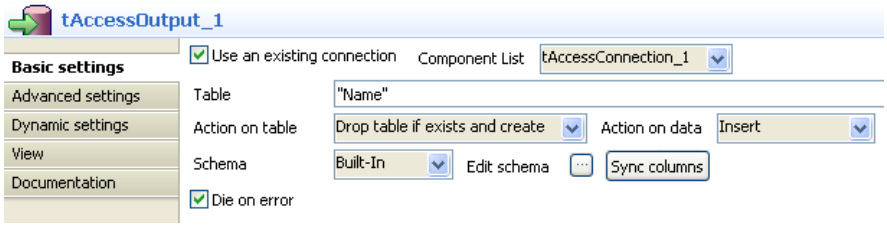


- Set the rest of the fields as usual, defining the row and field separators according to your file structure.
- Then set the schema manually through the **Edit schema** dialog box. Make sure the data type is correctly set, in accordance with the nature of the data processed.

- In the **tMap** Output area, add two output tables, one called *Name* for the *Name* table, the second called *Birthday*, for the *Birthday* table. For more information about the **tMap** component, see *Talend Open Studio for Big Data User Guide*.
- Drag the *Name* column from the **Input** area, and drop it to the *Name* table.
- Drag the *Birthday* column from the **Input** area, and drop it to the *Birthday* table.



- Then connect the output row links to distribute the flow correctly to the relevant DB output components.
- In each of the **tAccessOutput** components' **Basic settings** view, select the **Use an existing connection** check box to retrieve the **tAccessConnection** details.



- Set the **Table** name making sure it corresponds to the correct table, in this example either *Name* or *Birthday*.
- There is no action on the table as they are already created.
- Select **Insert** as **Action on data** for both output components.
- Click on **Sync columns** to retrieve the schema set in the **tMap**.
- Then connect the first **tAccessOutput** component to the first **tAccessInput** component using an **OnComponentOk** link.
- In each of the **tAccessInput** components' **Basic settings** view, select the **Use an existing connection** check box to retrieve the distributed data flow. Then set the schema manually through **Edit schema** dialog box.
- Then set the **Table Name** accordingly. In **tAccessInput_1**, this will be *Name*.
- Click on the **Guess Query**.
- Connect each **tAccessInput** component to **tLogRow** component with a **Row** > **Main** link. In each of the **tLogRow** components' **basic settings** view, select **Table** in the **Mode** field.
- Add the **tAccessCommit** component below the **tFileList** component in the design workspace and connect them together using an **OnComponentOk** link in order to terminate the Job with the transaction commit.
- In the **basic settings** view of **tAccessCommit** component and from the **Component list**, select the connection to be used, **tAccessConnection_1** in this scenario.
- Save your Job and press **F6** to execute it.

```
Starting job Access at 18:02 28/12/2010.

[statistics] connecting to socket on port 3938
[statistics] connected
.____.
|tLogRow_1|
|=---=|
|Name|
|=---=|
|Amy  |
|Bob  |
'____'

.----------.
|tLogRow_2 |
|=---------=|
|Birthday  |
|=---------=|
|02-02-2011|
|03-03-2023|
'----------'

[statistics] disconnected
Job Access ended at 18:02 28/12/2010. [exit code=0]
```

The parent table *Table1* is reused to generate the *Name* table and *Birthday* table.

tAccessInput

tAccessInput properties

Component family	Databases/Access	
Function	**tAccessInput** reads a database and extracts fields based on a query.	
Purpose	**tAccessInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the next component via a **Row** > **Main** connection.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and select the appropriate **tAccessConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	DB Version	Select the version of Access that you are using.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic** settings.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

Related topic in description of section *tContextLoad*.

tAccessOutput

tAccessOutput properties

Component family	Databases/Access	
Function	**tAccessOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAccessOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and select the appropriate **tAccessConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	DB Version	Select the version of Access that you are using.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries. **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing. **Delete**: Remove entries corresponding to the input flow. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic** settings. You can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table.

		Make sure the Nullable check box is selected for the corresponding columns in the schema.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Access database. It also allows you to create a reject flow using a **Row > Rejects** link to filSchemaSchemater data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tAccessOutputBulk

tAccessOutputBulk properties

The **tAccessOutputBulk** and **tAccessBulkExec** components are generally used together to output data to a delimited file and then to perform various actions on the file in an Access database, in a two step process. These two steps are fused together in the **tAccessOutputBulkExec** component, detailed in a separate section. The advantage of using a two step process is that it makes it possible to carry out transformations on the data before loading it in the database.

Component family	Databases/Access	
Function	**tAccessOutputBulk** writes a delimited file.	
Purpose	**tAccessOutputBulk** prepares the file which contains the data used to feed the Access database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Create directory if not exists	Select this check box to create the as yet non-existant file directory that specified in the **File name** field.
	Append	Select this check box to add any new rows to the end of the file
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Include header*	Select this check box to include the column header in the file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tAccessBulkExec** component. Used together they offer gains in performance while feeding an Access database.	

Related scenarios

For use cases in relation with **tAccessOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*
- section *Scenario: Inserting data in MySQL database*

tAccessOutputBulkExec

tAccessOutputBulkExec properties

The **tAccessOutputBulk** and **tAccessBulkExec** components are generally used together to output data to a delimited file and then to perform various actions on the file in an Access database, in a two step process. These two steps are fused together in **tAccessOutputBulkExec**.

Component family	Databases/Access	
Function	The **tAccessOutputBulkExec** component executes an Insert action on the data provided.	
Purpose	As a dedicated component, it improves performance during Insert operations in an Access database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and select the appropriate **tAccessConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	DB Version	Select the version of Access that you are using.
	DB name	Name of the database
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if doesn't exist**: The table is created if it does not already exist. **Clear a table**: The table content is deleted.
	Table	Name of the table to be written.

		Note that only one table can be written at a time and that the table must already exist for the insert operation to succeed
	FileName	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Create directory if not exists	Select this check box to create the as yet non existant file directory specified in the **File name** field.
	Append	Select this check box to append new rows to the end of the file.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic** settings. You can press **Ctrl+Space** to access a list of predefined global variables.
	Include header	Select this check box to include the column header to the file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded in the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tAccessOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting data in MySQL database*
- section *Scenario: Inserting transformed data in MySQL database*

tAccessRollback

tAccessRollback properties

This component is closely related to **tAccessConnection** and **tAccessCommit** components. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Access	
Function	**tAccessRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids involuntary commitment of part of a transaction.	
Basic settings	*Component list*	Select the **tAccessConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Access components, especially with **tAccessConnection** and **tAccessCommit**.	
Limitation	n/a	

Related scenarios

For **tAccessRollback** related scenario, see **tMysqlRollback**.

tAccessRow

tAccessRow properties

Component family	Databases/Access	
Function	**tAccessRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAccessRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and select the appropriate **tAccessConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	DB Version	Select the Access database version that you are using.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the source table where changes made to data should be captured.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Use PreparedStatement	Select this check box if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario: Resetting a DB auto-increment*
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tAS400Close

tAS400Close properties

Component family	Databases/AS400	
Function	**tAS400Close** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tAS400Connection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AS400 components, especially with **tAS400Connection** and **tAS400Commit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tAS400Commit

tAS400Commit Properties

This component is closely related to **tAS400Connection** and **tAS400Rollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/AS400	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tAS400Connection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tAS400Commit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AS400 components, especially with **tAS400Connection** and **tAS400Rollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAS400Connection** and **tAS400Rollback**. It usually does not make much sense to use one of these without using a **tAS400Connection** component to open a connection for the current transaction.

For **tAS400Commit** related scenario, see section *tMysqlConnection*

tAS400Connection

tAS400Connection Properties

This component is closely related to **tAS400Commit** and **tAS400Rollback**. It usually does not make much sense to use one of the components without using a **tAS400Connection** component to open a connection for the current transaction.

Component family	Databases/AS400	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the AS400 version in use
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Auto commit	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with AS400components, especially with **tAS400Commit** and **tAS400Rollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAS400Commit** and **tAS400Rollback**. It usually does not make much sense to use one of these without using a **tAS400Connection** component to open a connection for the current transaction.

For **tAS400Connection** related scenario, see section *tMysqlConnection*

tAS400Input

tAS400Input properties

Component family	Databases/AS400	
Function	**tAS400Input** reads a database and extracts fields based on a query.	
Purpose	**tAS400SInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tAS400Connection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the AS 400 version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topic, see **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*
- section *Scenario 2: Using StoreSQLQuery variable*.
-

Related topic in **tContextLoad**, see section *Scenario: Dynamic context use in MySQL DB insert*.

tAS400LastInsertId

tAS400LastInsertId properties

Component family	Databases	
Function	**tAS400LastInsertId** fetches the last inserted ID from a selected AS400 Connection.	
Purpose	**tAS400LastInsertId** obtains the primary key value of the record that was last inserted in an AS400 table by a user.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Component list	Select the relevant **tAS400Connection** component in the list if more than one connection is planned for the current job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used as an intermediary component.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario: Get the ID for the last inserted record*.

tAS400Output

tAS400Output properties

Component family	Databases/DB2	
Function	**tAS400Output** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAS400Output** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the AS400 version in use
	Use an existing connection	Select this check box and click the relevant **tAS400Connection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again.

		Clear a table: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Use commit control*	Select this check box to have access to the **Commit every** field where you can define the commit operation. **Commit every**: Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.

	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a AS400 database. It also allows you to create a reject flow using a **Row > Reject**s link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related scenarios

For related topics, see

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tAS400Rollback

tAS400Rollback properties

This component is closely related to **tAS400Commit** and **tAS400Connection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/AS400	
Function	**tAS400Rollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids involuntary commitment of part of a transaction.	
Basic settings	*Component list*	Select the **tAS400Connection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with AS400 components, especially with **tAS400Connection** and **tAS400Commit**.	
Limitation	n/a	

Related scenarios

For **tAS400Rollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tAS400Row

tAS400Row properties

Component family	Databases/AS400	
Function	**tAS400Row** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAS400Row** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tAS400Connection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the AS400 version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC Parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this check box if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tDB2BulkExec

tDB2BulkExec properties

Component family	Databases/DB2	
Function	**tDB2BulkExec** executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tDB2BulkExec** allows gains in performance during Insert operations to a DB2 database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Table Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear table**: The table content is deleted.

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Data file	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow.
Advanced settings	*Field terminated by*	Character, string or regular expression to separate fields.
	Date Format	Use this field to define the way months and days are ordered.
	Time Format	Use this field to define the way hours, minutes and seconds are ordered.
	Timestamp Format	Use this field to define the way date and time are ordered.
	Remove load pending	When the box is ticked, tables blocked in "pending" status following a bulk load are de-blocked.
	Load options	Click + to add data loading options: **Parameter**: select a loading parameter from the list. **Value**: enter a value for the parameter selected.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This dedicated component offers performance and flexibility of DB2 query handling.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tDB2BulkExec** related topics, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tDB2Close

tDB2Close properties

Component family	Databases/DB2	
Function	**tDB2Close** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tDB2Connection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with DB2 components, especially with **tDB2Connection** and **tDB2Commit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tDB2Commit

tDB2Commit Properties

This component is closely related to **tDB2Connection** and **tDB2Rollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/DB2	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tDB2Connection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tDB2Commit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with DB2 components, especially with **tDB2Connection** and **tDB2Rollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tDB2Connection** and **tDB2Rollback**. It usually doesn't make much sense to use one of these without using a **tDB2Connection** component to open a connection for the current transaction.

For **tDB2Commit** related scenario, see section *tMysqlConnection*

tDB2Connection

tDB2Connection properties

This component is closely related to **tDB2Commit** and **tDB2Rollback**. It usually does not make much sense to use one of these without using a **tDB2Connection** to open a connection for the current transaction.

Component family	Databases/DB2	
Function	**tDB2Connection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host name	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Table Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with DB2 components, especially with **tDB2Commit** and **tDB2Rollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

This component is closely related to **tDB2Commit** and **tDB2Rollback**. It usually does not make much sense to use one of these without using a **tDB2Connection** component to open a connection for the current transaction.

For **tDB2Connection** related scenario, see section *tMysqlConnection*

tDB2Input

tDB2Input properties

Component family	Databases/DB2	
Function	**tDB2Input** reads a database and extracts fields based on a query.	
Purpose	**tDB2Input** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link. If double quotes exist in the column names of a table, the double quotation marks cannot be retrieved when retrieving the column. Therefore, it is recommended not to use double quotes in column names in a DB2 database table.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Select the source table where to capture any changes made on data.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for DB2 databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also the related topic in section *Scenario: Dynamic context use in MySQL DB insert*.

tDB2Output

tDB2Output properties

Component family	Databases/DB2	
Function	**tDB2Output** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tDB2Output** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tDB2Connection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connention across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Table schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Convert columns and table names to uppercase	Select this check box to uppercase the names of the columns and the name of the table.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Make sure the Nullable check box is selected for the corresponding columns in the schema.
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected,

		you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected the **Insert**, the **Update** or the **Delete** option in the **Action on data** field.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a DB2 database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tDB2Output** related topics, see

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tDB2Rollback

tDB2Rollback properties

This component is closely related to **tDB2Commit** and **tDB2Connection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/DB2	
Function	**tDB2Rollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tDB2Connection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with DB2 components, especially with **tDB2Connection** and **tDB2Commit**.	
Limitation	n/a	

Related scenarios

For **tDB2Rollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables* of the **tMysqlRollback**.

tDB2Row

tDB2Row properties

Component family	Databases/DB2	
Function	**tDB2Row** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tDB2Row** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tDB2Connection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tDB2Row** related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tDB2SCD

tDB2SCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tDB2SCD*.

tDB2SCDELT

tDB2SCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tDB2SCDELT*.

tDB2SP

tDB2SP properties

Component family	Databases/DB2	
Function	**tDB2SP** calls the database stored procedure.	
Purpose	**tDB2SP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tDB2Connection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Check this box, if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure. Select the **Type** of parameter:

		IN: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the section *tPostgresqlCommit* component if you want to analyze a set of records from a database table or DB query and return single records.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topic, see section *Scenario: Executing a stored procedure in the MDM Hub*.

Check section *tPostgresqlCommit* as well if you want to analyze a set of records from a database table or DB query and return single records.

tInformixBulkExec

tInformixBulkExec Properties

tInformixOutputBulk and **tInformixBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tInformixOutputBulkExec** component, detailed in another section. The advantage of using two components is that data can be transformed before it is loaded in the database.

Component Family	Databases/Informix	
Function	**tInformixBulkExec** executes Insert operations on the data supplied.	
Purpose	**tInformixBulkExec** is a dedicated component which improves performance during Insert operations in Informix databases.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Execution Platform	Select the operating system you are using.
	Use an existing connection	Select this check box and click the relevant **tInformixBulkExec** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	DB server listening port.
	Database	Name of the database.
	Schema	Name of the schema.
	Username et *Password*	DB user authentication data.
	Instance	Name of the Informix instance to be used. This information can generally be found in the *SQL hosts* file.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out.

		Drop and create a table: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Informix Directory	Indicate the access path to your Informix directory.
	Data file	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Action on data	On the data of the table defined, you can perform the following operations: **Insert**: Add new data to the table. If duplicates are found, the job stops. **Update**: Update the existing table data. **Insert or update**: Add data or update the existing data. **Update or insert** : Update the existing entries or create them if they do not already exist. **Delete**: Delete the entry data which corresponds to the input flow. *You must specify at least one key upon which the Update and Delete operations are to be based. It is possible to define the columns which should be used as the key from the schema, from both the **Basic Settings** and the **Advanced Settings**, to optimise these operations.*
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Field terminated by	Character, string or regular expression which separates the fields.
	Set DBMONEY	Select this check box to define the decimal separator in the **Decimal separator** field.
	Set DBDATE	Select the date format that you want to apply.
	Rows Before Commit	Enter the numbere of rows to be processed before the commit.
	Bad Rows Before Abort	Enter the number of rows in error at which point the Job should stop.
	tStat Catcher Statistics	Select this check box to colelct the log data at component level.
	Output	Where the output should go.
Usage	This component offers database query flexibility and covers all possible DB2 queries which may be required.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which **tInformixBulkExec** might be used, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tInformixClose

tInformixClose properties

Component Family	Databases/Informix	
Function	**tInformixClose** closes an active connection to a database.	
Purpose	This component closes connection to Informix databases.	
Basic settings	*Component list*	If there is more than one connection used in the Job, select **tInformixConnection** from the list.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is generally used as an input component. It requires an output component.	
Limitation	n/a	

Related scenario

This component is for use with **tInformixConnection** and **tInformixRollback**. They are generally used along with **tInformixConnection** as the latter allows you to open a connection for the transaction which is underway.

To see a scenario in which **tInformixClose** might be used, see section *tMysqlConnection*.

tInformixCommit

tInformixCommit properties

This component is closely related to **tInformixConnection** and **tInformixRollback**. They are generally used to execute transactions together.

Component Family	Databases/Informix	
Function	**tInformixCommit** validates data processed in a job from a connected database.	
Purpose	Using a single connection, make a global commit just once instead of commiting every row or batch of rows separately. This improves performance.	
Basic settings	*Component list*	If there is more than one connection in the Job, select **tInformixConnection** from the list.
	Close connection	This check box is selected by default. It means that the database conenction will be closed once the commit has been made. Clear the check box to continue using the connection once the component has completed its task. *If you are using a **Row** > **Main** type connection to link **tInformixCommit** to your Job, your data will be committed row by row. If this is the case, do not select this check bx otherwise the conenction will be closed before the commit of your first row is finalized.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is generally used along with Informix components, particularly **tInformixConnection** and **tInformixRollback**.	
Limitation	n/a	

Related Scenario

This component is for use with **tInformixConnection** and **tInformixRollback**. They are generally used along with **tInformixConnection** as the latter allows you to open a connection for the transaction which is underway

To see a scenario in which **tInformixCommit** might be used, see section *tMysqlConnection*.

tInformixConnection

tInformixConnection properties

This component is closely related to **tInformixCommit** and **tInformixRollback**. They are generally used along with **tInformixConnection**, with **tInformixConnection** opening the connection for the transaction.

Database Family	Databases/Informix	
Function	**tInformixConnection** opens a connection to a database in order that a transaction may be made.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	DB server listening port.
	Database	Name of the database.
	Schema	Name of the schema
	Username et *Password*	DB user authentication data.
	Instance	Name of the Informix instance to be used. This information can generally be found in the SQL hosts file.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Use Transaction*	Clear this check box when the database is configured in NO_LOG. mode. If the check box is selected, you can choose whether to activate the Auto Commit option.
	*tStat*Catcher Statistics	Select this check box to collect the log data at a component level.
Usage	This component is generally used with other Informix components, particularly **tInformixCommit** and **tInformixRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which the **tInformixConnection**, might be used, see section *Scenario: Inserting data in mother/daughter tables*.

tInformixInput

tInformixInput properties

Component family	Databases/Informix	
Function	**tInformixInput** reads a database and extracts fields based on a query.	
Purpose	**tInformixInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	DB server	Name of the database server
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Usage	This component covers all possible SQL queries for DB2 databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also scenario for **tContextLoad**: section *Scenario: Dynamic context use in MySQL DB insert*.

tInformixOutput

tInformixOutput properties

Component family	Databases/Informix	
Function	**tInformixOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tInformixOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	DB server	Name of the database server
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update*

		operation. Do the same in the ***Key in delete column*** *for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. sYou can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
	Optimize the batch insertion	Ensure the check box is selected, to optimize the insertion of batches of data.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Informix database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tInformixOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tInformixOutputBulk

tInformixOutputBulk properties

tInformixOutputBulk and **tInformixBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tInformixOutputBulkExec** component, detailed in another section. The advantage of using two components is that data can be transformed before it is loaded in the database.

Component family	Databases/Informix	
Function	Writes a file composed of columns, based on a defined delimiter and on Informix standards.	
Purpose	Prepares the file to be used as a parmameter in the INSERT query used to feed Informix databases.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to append new rows to the end of the file.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression used to separate fields
	Set DBMONEY	Select this box if you want to define the decimal separator in the corresponding field.
	Set DBDATE	Select the date format that you want to apply.
	Create directory if not exists	This check box is selected automatically. The option allows you to create a folder for the output file if it doesn’t already exist.
	Custom the flush buffer size	Select this box in order to customize the memory size used to store the data temporarily. In the **Row number** field enter the number of rows at which point the memory should be freed.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	*tStat*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is generally used along with **tInformixBulkExec**. Together, they improve performance levels when adding data to an Informix database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which **tInformixOutputBulk** might be used, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tInformixOutputBulkExec

tInformixOutputBulkExec properties

tInformixOutputBulk and **tInformixBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tInformixOutputBulkExec** component.

Component Family	Databases/Informix	
Function	**tInformixOutputBulkExec** carries out Insert operations using the data provided.	
Purpose	**tInformixOutputBulkExec** is a dedicated componant which improves performance during Insert operations in Informix databases.	
Basic settings	*Property Type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		No properties stored centrally
	Execution platform	Select the operating system you are using.
	Use an existing connection	Select the check box and choose the appropriate **tInformixConnection** component from the list to use pre-defined connection parameters. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	Database server IP address.
	Port	DB server listening port.
	Database	Name of the database.
	Schema	Name of the schema.
	Username et *Password*	DB user authentication data.
	Instance	Name of the Informix instance to be used. This information can generally be found in the *SQL hosts* file.
	Table	Name of the table to be written. Note that only one table can be written at a time and the table must already exist for the insert operation to be authorised.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out.

		Drop and create a table: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Informix Directory	Indicate the access path to your Informix directory.
	Data file	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add rows to the end of the file.
	Action on data	Select the operation you want to perform: **Bulk insert Bulk update** The details asked will be different according to the action chosen.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Fields terminated by	Character, string or regular expression used to separate the fields
	Set DBMONEY	Select this check box to define the decimal separator used in the corresponding field.
	Set DBDATE	Select the date format you want to apply.
	Rows Before Commit	Enter the number of rows to be processed before the commit.
	Bad Rows Before Abort	Enter the number of rows in error at which point the Job should stop.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if required.
	Custom the flush buffer size	Select this box in order to customize the memory size used to store the data temporarily. In the **Row number** field enter the number of rows at which point the memory should be freed.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect the log data at a component level.
	Output	Where the output should go.
Usage	This component is generally used when no particular transformation is required on the data to be inserted in the database.	
Limitation	n/a	

Related scenario

For a scenario in which **tInformixOutputBulkExec** might be used, see:

- section *Scenario: Inserting transformed data in MySQL database*.

- section *Scenario: Inserting data in MySQL database*.

tInformixRollback

tInformixRollback properties

This component is closely related to **tInformixCommit** and **tInformixConnection**. They are generally used together to execute transactions.

Famille de composant	Databases/Informix	
Function	**tInformixRollback** cancels transactions in connected databases.	
Purpose	This component prevents involuntary transaction commits.	
Basic settings	*Component list*	Select the **tInformixConnection** component from the list if you plan to add more than one connection to the Job.
	Close Connection	Clear this checkbox if you want to continue to use the connection once the component has completed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component must be used with other Informix components, particularly **tInformixConnection** and **tInformixCommit**.	
Limitation	n/a	

Related Scenario

For a scenario in which **tInformixRollback** might be used, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tInformixRow

tInformixRow properties

Component family	Databases/Informix	
Function	**tInformixRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tInformixRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tInformixConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this check box if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tInformixSCD

The **tInformixSCD** component belongs to two different families: **Business Intelligence** and **Databases**. For further information, see section *tInformixSCD*.

tInformixSP

tInformixSP properties

Component Family	Databases/Informix	
Function	**tInformixSP** calls procedures stored in a database.	
Purpose	**tInformixSP** allows you to centralise multiple and complex queries in a database and enables you to call them more easily.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No properties stored centrally.
	Use an existing connection	Select the check box and choose the appropriate **tInformixConnection** component from the list to use pre-defined connection parameters. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username et *Password*	User authentication information.
	Instance	Name of the Informix instance to be used. This information can generally be found in the *SQL hosts* file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Enter the exact name of the stored procedure (SP).
	Is Function / *Return result in*	Select this check box if only one value must be returned. From the list, select the the schema column upon which the value to be obtained is based.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check section *tPostgresqlCommit*, if you want to analyze a set of records from a database table or DB query and return single records.
	Use Transaction	Clear this check box if the database is configured in the NO_LOG mode.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to collect log data at a component level.
Usage	This is an intermediary component. It can also be used as an entry component. In this case, only the entry parameters are authorized.	
Limitation	The stored procedure syntax must correspond to that of the database. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which **tInformixSP** may be used, see:

- section *Scenario: Executing a stored procedure in the MDM Hub*.
- section *Scenario: Checking number format using a stored procedure*.

Also, see section *tPostgresqlCommit* if you want to analyse a set of records in a table or SQL query.

tMSSqlBulkExec

tMSSqlBulkExec properties

The **tMSSqlOutputBulk** and **tMSSqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tMSSqlOutputBulkExec** component, detailed in a separate section. The advantage of using a two step process is that the data can be transformed before it is loaded in the database.

Component family	Databases/MSSql	
Function	Executes the Insert action on the provided data.	
Purpose	As a dedicated component, **tMSSqlBulkExec** offers gains in performance while carrying out the Insert operations to a MSSql database	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data is stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created.

		Create table if not exists: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Remote File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Action*	Select the action to be carried out **Bulk insert Bulk update Bcp query out** Depending on the action selected, the requied information varies.
Bulk insert & Bulk update	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Fields terminated	Character, string or regular expression to separate fields.
	Rows terminated	Character, string or regular expression to separate rows.
	First row	Type in the number of the row where the action should start
	Code page	This value can be any of the followings: **OEM** (by default value) **ACP RAW** User-defined
	Data file type	Select the type of data being handled.
	Output	Select the type of output for the standard output of the MSSql database: **to console**, **to global variable**.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Bcp query out	*Fields terminated*	Character, string or regular expression to separate fields.
	Rows terminated	Character, string or regular expression to separate rows.
	Data file type	Select the type of data being handled.
	Output	Select the type of output to pass the processed data onto: **to console**: data is viewed in the Log view. **to global variable**: data is put in output variable linked to a tsystem component
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMSSqlOutputBulk** component. Used together, they can offer gains in performance while feeding a MSSql database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tMSSqlBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tMSSqlColumnList

tMSSqlColumnList Properties

Component family	Databases/MS SQL	
Function	Iterates on all columns of a given table through a defined MS SQL connection.	
Purpose	Lists all column names of a given MSSql table.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current job.
	Table name	Enter the name of the tabe.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlConnection**.	
Limitation	n/a	

Related scenario

For **tMSSqlColumnList** related scenario, see section *Scenario: Iterating on a DB table and listing its column names*.

tMSSqlClose

tMSSqlClose properties

Component family	Databases/MSSql	
Function	**tMssqlClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tMssqlConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with tMssql components, especially with **tMssqlConnection** and **tMssqlCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tMSSqlCommit

tMSSqlCommit properties

This component is closely related to **tMSSqlConnection** and **tMSSqlRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/MSSql	
Function	**tMSSqlCommit** validates the data processed through the job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current Job.
	Close connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tMSSqlCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is to be used along with Mssql components, especially with **tMSSqlConnection** and **tMSSqlRollback** components.	
Limitation	n/a	

Related scenarios

This component is closely related to **tMSSqlConnection** and **tMSSqlRollback**. It usually does not make much sense to use one of these without using a **tMSSqlConnection** component to open a connection for the current transaction.

For a **tMSSqlCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tMSSqlConnection

tMSSqlConnection properties

This component is closely related to **tMSSqlCommit** and **tMSSqlRollback**. Both components are usually used with a **tMSSqlConnection** component to open a connection for the current transaction.

Component family	Databases/MSSQL	
Function	**tMSSqlConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Schema	Schema name.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlCommit** and **tMSSqlRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

This component is closely related to **tMSSqlCommit** and **tMSSqlRollback**. It usually does not make much sense to use one if these without using a **tMSSqlConnection** component to open a connection for the current transaction.

For **tMSSqlConnection** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tMSSqlInput

tMSSqlInput properties

Component family	Databases/MS SQL Server	
Function	**tMSSqlInput** reads a database and extracts fields based on a query.	
Purpose	**tMSSqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for MS SQL server databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

Related topics in **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*
- section *Scenario 2: Using StoreSQLQuery variable*.

For related topic in **tContextLoad**, see section *Scenario: Dynamic context use in MySQL DB insert*.

tMSSqlLastInsertId

tMSSqlLastInsertId properties

Component Family	Databases/MS SQL server	
Function	**tMSSqlLastInsertId** displays the last IDs added to a table from a MSSql specified connection.	
Purpose	**tMSSqlLastInsertId** enables you to retrieve the last primary keys added by a user to a MSSql table.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Component list	Select the **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined, if there are more than one component in this list.
Advanced settings	*tStat*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Get the ID for the last inserted record*

tMSSqlOutput

tMSSqlOutput properties

Component family	Databases/MS SQL server	
Function	**tMSSqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tMSSqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Schema	Name of the schema.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

		Truncate table: The table content is deleted. You do not have the possibility to rollback the operation.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Single Insert Query**: Add entries to the table in a batch **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. **Insert if not exist** : Add new entries to the table if they do not exist. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Specify identity field	Select this check box to specify the identity field, which is made up of an automatically incrementing identification number. When this check box is selected, three other fields display: **Identity field**: select the column you want to define as the identity field from the list. **Start value**: type in a start value, used for the very first row loaded into the table. **Step**: type in an incremental value, added to the value of the previous row that was loaded. You can also specify the identity field from the schema of the component. To do so, set the **DB Type** of the relevant column to **INT IDENTITY**. When the **Specify identity field** check box is selected, the **INT IDENTITY DB Type** in the schema is ignored.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.

		You can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Ignore date validation	Select this check box to ignore the date validation and insert the data directly into the database for the data types of DATE, DATETIME, DATETIME2 and DATETIMEOFFSET.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Make sure that the Nullable check box is selected for the corresponding columns in the schema.
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected the **Insert**, the **Update**, the **Single Insert Query** or the **Delete** option in the **Action on data** field. If you are using the MS Sql Server 2008 version, make sure that the Batch Size is less than or equal to 2000 parameter markers divided by the number of columns in the schema.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a MSSql database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tMSSqlOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.

- section *Scenario 1: Adding a new column and altering data in a DB table*.

tMSSqlOutputBulk

tMSSqlOutputBulk properties

The **tMSSqlOutputBulk** and **tMSSqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tMSSqlOutputBulkExec** component, detailed in a separate section. The advantage of using a two step process is that the data can be transformed before it is loaded in the database.

Component family	Databases/MSSql	
Function	Writes a file with columns based on the defined delimiter and the MSSql standards.	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the MSSql database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the records.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check to include the column header.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStaCatcher statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMSSqlBulkExec** component. Used together they offer gains in performance while feeding a MSSql database.	

Related scenarios

For use cases in relation with **tMSSqlOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tMSSqlOutputBulkExec

tMSSqlOutputBulkExec properties

The **tMSSqlOutputBulk** and **tMSSqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tMSSqlOutputBulkExec** component.

Component family	Databases/MSSql	
Function	Executes actions on the provided data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a MSSql database.	
Basic settings	*Action*	Select the action to be carried out **Bulk insert Bulk update**
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	DB name	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out.

		Drop and create a table: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation. **Clear a table**: The table content is deleted. You have the possibility to rollback the operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the records
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	First row	Type in the number of the row where the action should start.
	Include header	Select this check box to include the column header.
	Code page	**OEM** code pages used to map a specific set of characters to numerical code point values.
	Data file type	Select the type of data being handled.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStaCatcher statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tMSSqlOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*
- section *Scenario: Inserting data in MySQL database*

tMSSqlRollback

tMSSqlRollback properties

This component is closely related to **tMSSqlCommit** and **tMSSqlConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlConnection** and **tMSSqlCommit** components.	
Limitation	n/a	

Related scenario

For **tMSSqlRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tMSSqlRow

tMSSqlRow properties

Component family	Databases/DB2	
Function	**tMSSqlRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tMSSqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be used.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.

		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the **Query** field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tMSSqlSCD

tMSSqlSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tMSSqlSCD*.

tMSSqlSP

tMSSqlSP Properties

Component family	Databases/MSSql	
Function	**tMSSqlSP** calls the database stored procedure.	
Purpose	**tMSSqlSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Select this check box, if only a value is to be returned. Select on the list the schema column, the value to be returned is based on.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check section *tPostgresqlCommit*, if you want to analyze a set of records from a database table or DB query and return single records.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related scenarios, see:

- section *Scenario: Executing a stored procedure in the MDM Hub*.
- section *Scenario: Checking number format using a stored procedure*.

Check as well section *tPostgresqlCommit* to analyze a set of records from a database table or DB query and return single records.

tMSSqlTableList

tMSSqlTableList Properties

Component family	Databases/MS SQL	
Function	Iterates on a set of table names through a defined MS SQL connection.	
Purpose	Lists the names of a given set of MSSql tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlConnection**.	
Limitation	n/a	

Related scenario

For **tMSSqlTableList** related scenario, see section *Scenario: Iterating on a DB table and listing its column names*.

tMysqlBulkExec

tMysqlBulkExec properties

The **tMysqlOutputBulk** and **tMysqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT statement used to feed a database. These two steps are fused together in the **tMysqlOutputBulkExec** component, detailed in a separate section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

Component family	Databases/Mysql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tMysqlBulkExec** offers gains in performance while carrying out the Insert operations to a Mysql database	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of My SQL that you are using.
	Use an existing connection	Select this check box when using a configured **tMysqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.

		Clear table: The table content is deleted. You have the possibility to rollback the operation.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Local file Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*s.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Lines terminated by	Character or sequence of characters used to separate lines.
	Fields terminated by	Character, string or regular expression to separate fields.
	Enclosed by	Character used to enclose text.
	Action on data	On the data of the table defined, you can perform: **Insert records in table**: Add new records to the table. **Update records in table**: Make changes to existing records. **Replace records in table**: Replace existing records with new ones. **Ignore records in table:** Ignore the existing records, or insert the new ones.
	Records contain NULL value	Check this box if you want to retrieve the null values from the input data flow. If you do not check this box, the null values from the input data flow will be considered as empty fields in the output data flow.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMysqlOutputBulk** component. Used together, they can offer gains in performance while feeding a Mysql database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tMysqlBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tMysqlClose

tMysqlClose properties

Function	**tMysqlClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection** and **tMysqlCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tMysqlColumnList

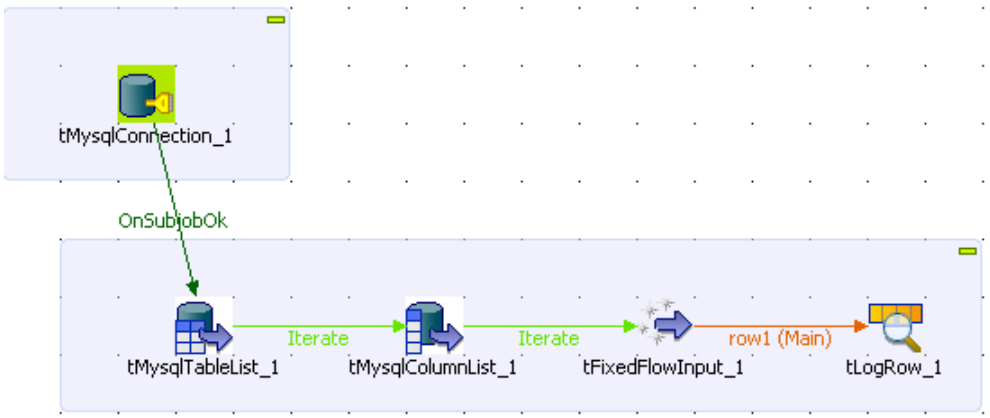
tMysqlColumnList Properties

Component family	Databases/MySQL	
Function	Iterates on all columns of a given table through a defined Mysql connection.	
Purpose	Lists all column names of a given Mysql table.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Table name	Enter the name of the table.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection**.	
Limitation	n/a	

Scenario: Iterating on a DB table and listing its column names

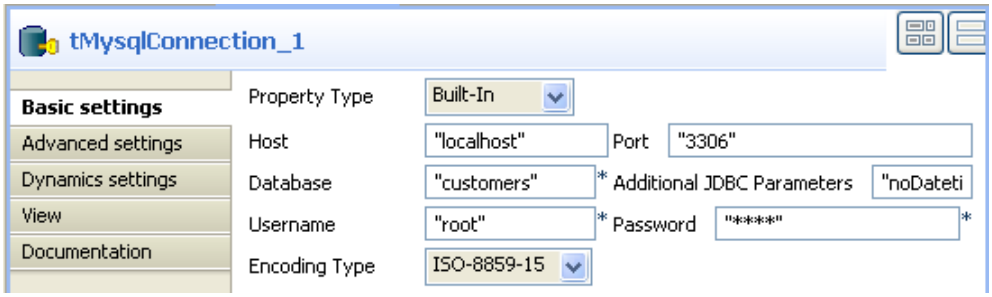
The following Java scenario creates a five-component job that iterates on a given table name from a Mysql database using a Where clause and lists all column names present in the table.

- Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tMysqlTableList**, **tMysqlColumnList**, **tFixedFlowInput**, and **tLogRow**.
- Connect **tMysqlConnection** to **tMysqlTableList** using an **OnSubjobOk** link.
- Connect **tMysqlTableList**, **tMysqlColumnList**, and **tFixedFlowInput** using **Iterate** links.
- Connect **tFixedFlowInput** to **tLogRow** using a **Row Main** link.



- In the design workspace, select **tMysqlConnection** and click the **Component** tab to define its basic settings.
- In the **Basic settings** view, set the database connection details manually or select them from the context variable list, through a **Ctrl+Space** click in the corresponding field if you have stored them locally as Metadata DB connection entries.

For more information about Metadata, see *Talend Open Studio for Big Data* **User Guide**.



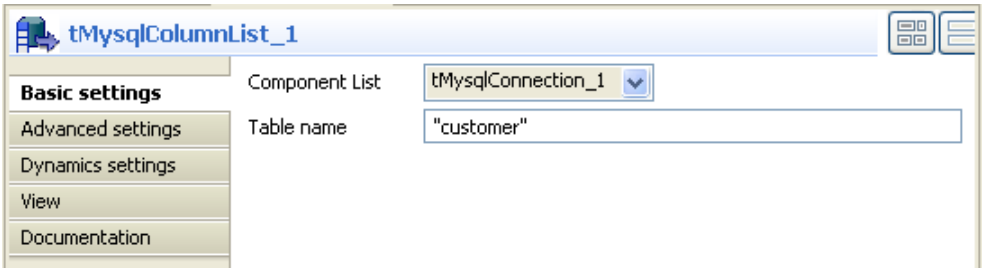
In this example, we want to connect to a Mysql database called *customers*.

- In the design workspace, select **tMysqlTableList** and click the **Component** tab to define its basic settings.

- On the **Component list**, select the relevant Mysql connection component if more than one connection is used.
- Enter a Where clause using the right syntax in the corresponding field to iterate on the table name(s) you want to list on the console.

  In this scenario, the table we want to iterate on is called *customer*.

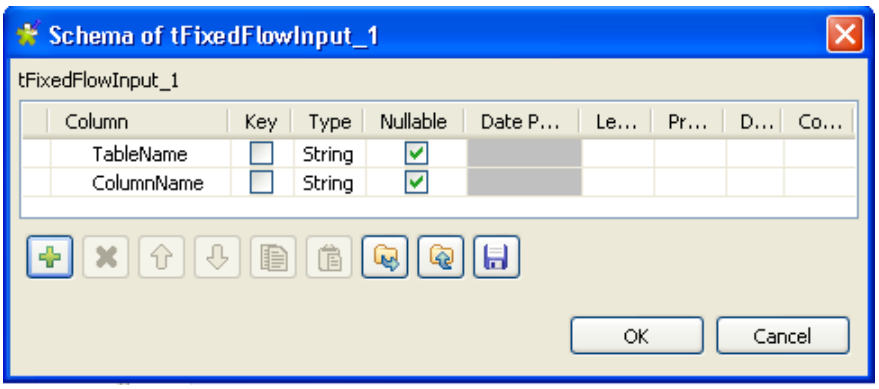
- In the design workspace, select **tMysqlColumnList** and click the **Component** tab to define its basic settings.



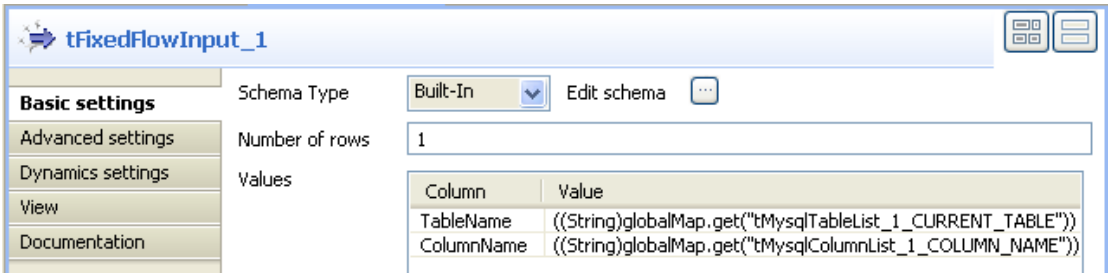
- On the **Component list**, select the relevant Mysql connection component if more than one connection is used.
- In the **Table name** field, enter the name of the DB table you want to list its column names.

  In this scenario, we want to list the columns present in the DB table called *customer*.

- In the design workspace, select **tFixedFlowInput** and click the **Component** tab to define its basic settings.
- Set the **Schema** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to define the data you want to use as input. In this scenario, the schema is made of two columns, the first for the table name and the second for the column name.



- Click **OK** to close the dialog box, and accept propagating the changes when prompted by the system. The defined columns display in the **Values** panel of the **Basic settings** view.
- Click in the **Value** cell for each of the two defined columns and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *((String)globalMap.get("tMysqlTableList_1_CURRENT_TABLE"))* and *((String)globalMap.get("tMysqlColumnList_1_COLUMN_NAME"))* for the *TableName* and *ColumnName* respectively.



- In the design workspace, select **tLogRow**.
- Click the **Component** tab and define the basic settings for **tLogRow** as needed.
- Save your job and press **F6** to execute it.

```
Starting job Column_Table_List at 00:55 17/11/2008.
customer|id
customer|First_Name
customer|Last_Name
customer|Address
customer|id_State
Job Column_Table_List ended at 00:55 17/11/2008. [exit code=0]
```

The name of the DB table is displayed on the console along with all its column names.

tMysqlCommit

tMysqlCommit Properties

This component is closely related to **tMysqlConnection** and **tMysqlRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/MySQL	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tMysqlCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection** and **tMysqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tMysqlConnection** and **tMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tMysqlConnection** component to open a connection for the current transaction.

For **tMysqlCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tMysqlConnection

tMysqlConnection Properties

This component is closely related to **tMysqlCommit** and **tMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tMysqlConnection** component to open a connection for the current transaction.

Component family	Databases/MySQL	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime.
Usage	This component is to be used along with Mysql components, especially with **tMysqlCommit** and **tMysqlRollback** components.	
Limitation	n/a	

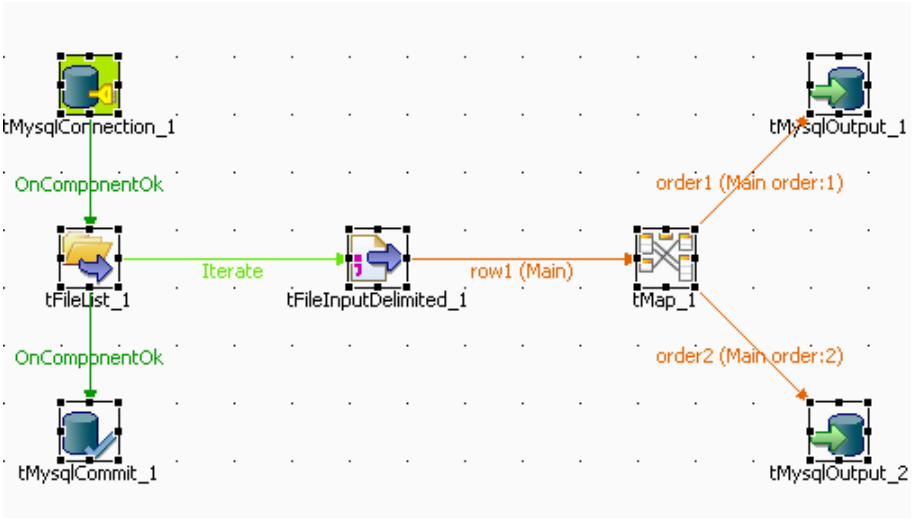
Scenario: Inserting data in mother/daughter tables

The following Job is dedicated to advanced database users, who want to carry out multiple table insertions using a parent table id to feed a child table. As a prerequisite to this Job, follow the steps described below to create the relevant tables using an engine such as *innodb*.

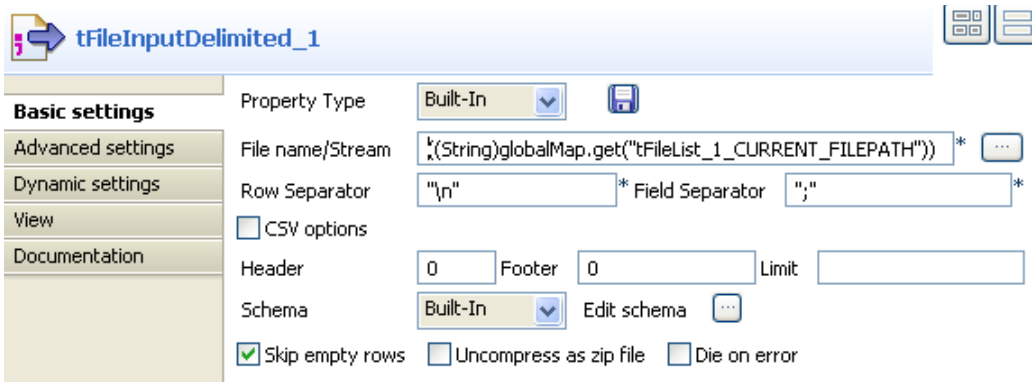
- In a command line editor, connect to your Mysql server.

- Once connected to the relevant database, type in the following command to create the parent table: create table f1090_mum(id int not null auto_increment, name varchar(10), primary key(id)) engine=innodb;
- Then create the second table: create table baby (id_baby int not null, years int) engine=innodb;

Back in *Talend Open Studio for Big Data*, the Job requires seven components including **tMysqlConnection** and **tMysqlCommit**.

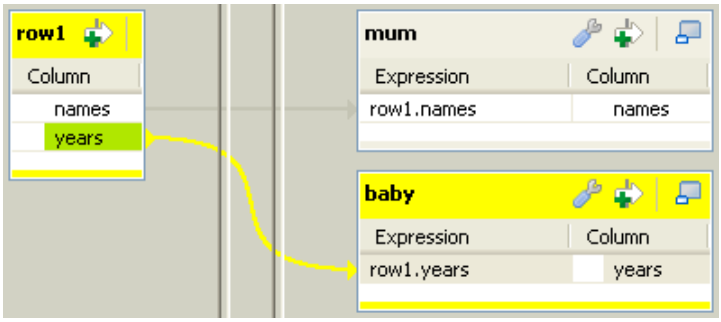


- Drag and drop the following components from the **Palette**: **tFileList**, **tFileInputDelimited**, **tMap**, **tMysqlOutput** (x2).
- Connect the **tFileList** component to the input file component using an **Iterate** link as the name of the file to be processed will be dynamically filled in from the **tFileList** directory using a global variable.
- Connect the **tFileInputDelimited** component to the **tMap** and dispatch the flow between the two output Mysql DB components. Use a **Row** link for each for these connections representing the main data flow.
- Set the **tFileList** component properties, such as the directory. name where files will be fetched from.
- Add a **tMysqlConnection** component and connect it to the starter component of this job, in this example, the **tFileList** component using an **OnComponentOk** link to define the execution order.
- In the **tMysqlConnection** Component view, set the connection details.
- On the **tFileInputDelimited** component's **Basic settings** panel, press **Ctrl+Space** bar to access the variable list. Set the **File Name** field to the global variable: *tFileList_1.CURRENT_FILEPATH*

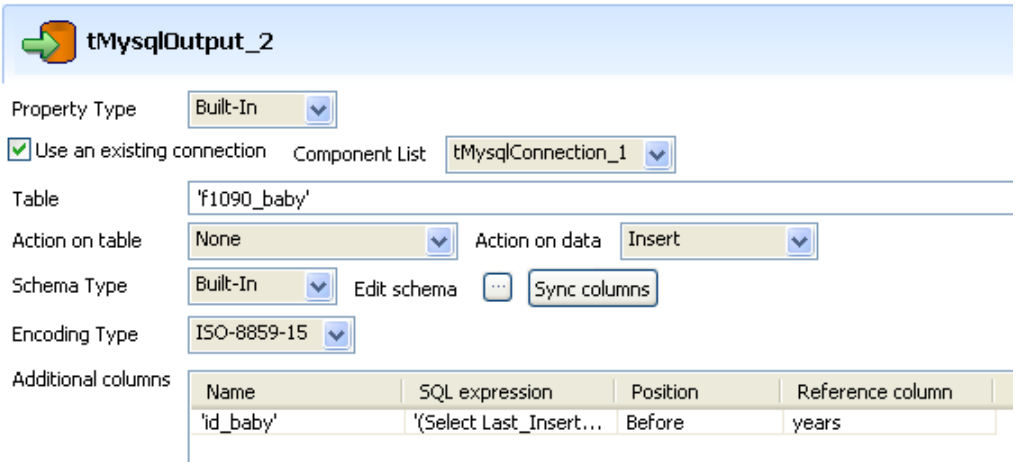


- Set the rest of the fields as usual, defining the row and field separators according to your file structure.
- Then set the schema manually through the **Edit schema** feature or select the schema from the Repository. In Java version, make sure the data type is correctly set, in accordance with the nature of the data processed.

- In the **tMap** Output area, add two output tables, one called mum for the parent table, the second called baby, for the child table.
- Drag the *Name* column from the **Input** area, and drop it to the mum table.
- Drag the *Years* column from the **Input** area and drop it to the baby table.



- Make sure the mum table is on the top of the baby table as the order is determining for the flow sequence hence the DB insert to perform correctly.
- Then connect the output row link to distribute correctly the flow to the relevant DB output component.
- In each of the **tMysqlOutput** components' **Basic settings** panel, select the **Use an existing connection** check box to retrieve the **tMysqlConnection** details.



- Set the **Table** name making sure it corresponds to the correct table, in this example either *f1090_mum* or *f1090_baby*.
- There is no action on the table as they are already created.
- Select **Insert** as **Action on data** for both output components.
- Click on Sync columns to retrieve the schema set in the **tMap**.
- In the **Additional columns** area of the DB output component corresponding to the child table (*f1090_baby*), set the id_baby column so that it reuses the id from the parent table.
- In the **SQL expression** field type in: '(Select Last_Insert_id())'
- The position is *Before* and the **Reference column** is *years*.
- Add the **tMysqlCommit** component to the design workspace and connect it from the **tFileList** component using a **OnComponentOk** connection in order for the Job to terminate with the transaction commit.

- On the **tMysqlCommit Component** view, select in the list the connection to be used.
- Save your Job and press **F6** to execute it.

```
mysql> select * from f1090_mum
    -> ;
+----+--------+
| id | names  |
+----+--------+
|  6 | john   |
|  7 | bruce  |
|  8 | beth   |
|  9 | andrew |
| 10 | donald |
| 11 | betty  |
| 12 | john   |
| 13 | bruce  |
| 14 | beth   |
| 15 | andrew |
| 16 | donald |
| 17 | betty  |
+----+--------+
12 rows in set (0.00 sec)
```

```
mysql> select * from f1090_baby
    -> ;
+---------+-------+
| id_baby | years |
+---------+-------+
|       6 |    10 |
|       7 |    23 |
|       8 |    34 |
|       9 |    10 |
|      10 |    23 |
|      11 |    34 |
|      12 |    10 |
|      13 |    23 |
|      14 |    34 |
|      15 |    10 |
|      16 |    23 |
|      17 |    34 |
+---------+-------+
12 rows in set (0.00 sec)
```

The parent table *id* has been reused to feed the *id_baby* column.

tMysqlInput

tMysqlInput properties

Component family	Databases/MySQL	
Function	**tMysqlInput** reads a database and extracts fields based on a query.	
Purpose	**tMysqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tMysqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.

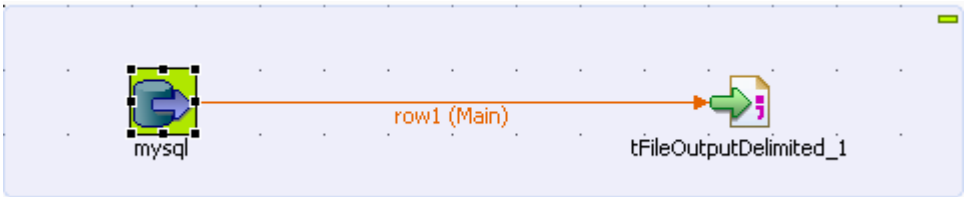
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. When you need to handle data of the time-stamp type *0000-00-00 00:00:00* using this component, set the parameter as: `noDatetimeStringSync=true&zeroDa-teTimeBehavior=convertToNull`.
	Enable stream	Select this check box to enables streaming over buffering which allows the code to read from a large table without consuming a large amount of memory in order to optimize the performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns. Clear **Trim all the String/Char columns** to enable Trim columns in this field.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Mysql databases.	

Scenario 1: Writing columns from a MySQL database to an output file

In this scenario we will read certain columns from a MySQL database, and then write them to a table in a local output file.

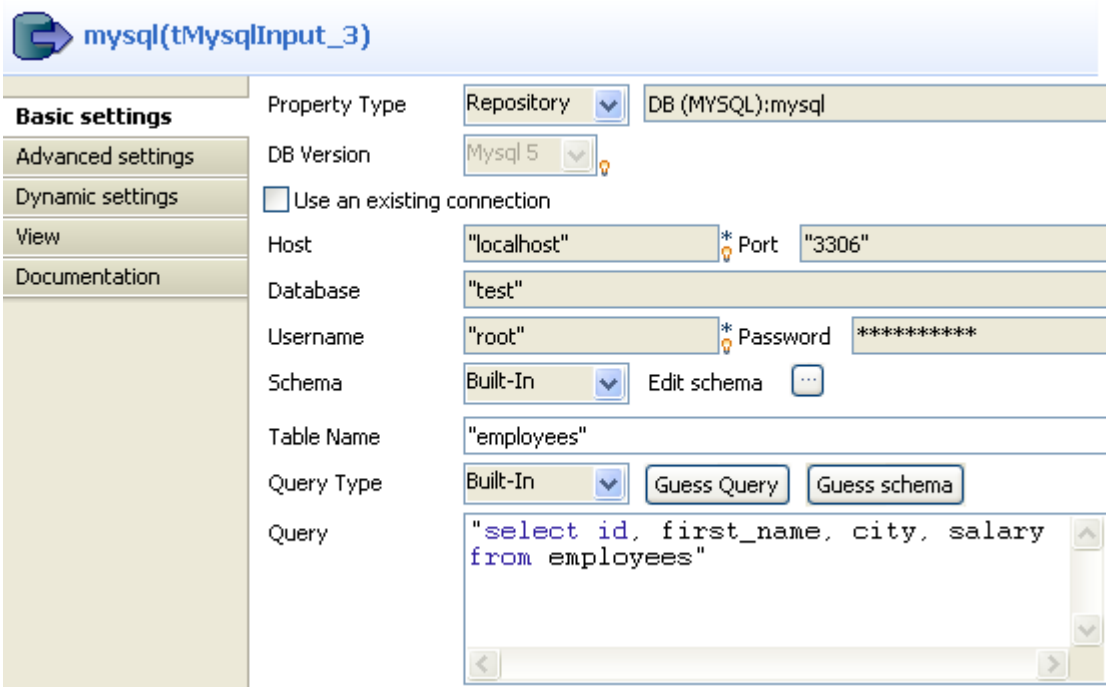
Dragging and dropping components and linking them together

1. Drop **tMysqlInput** and **tFileOutputDelimited** from the **Palette** onto the workspace.
2. Link **tMysqlInput** to **tFileOutputDelimited** using a **Row > Main** connection.



Configuring the components

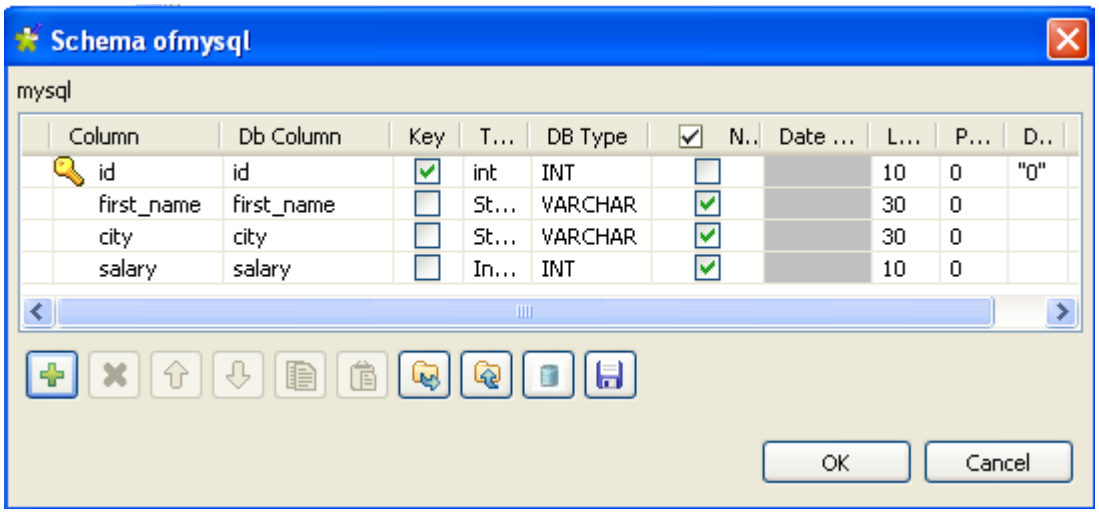
1. Double-click **tMysqlInput** to open its **Basic Settings** view in the **Component** tab.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Define the DB connection details in the corresponding fields.
3. Click **Edit schema** to define the desired schema.

   The schema editor opens:



4. Click the **[+]** button to add the rows that you will use to define the schema, four columns in this example *id*, *first_name*, *city* and *salary*.

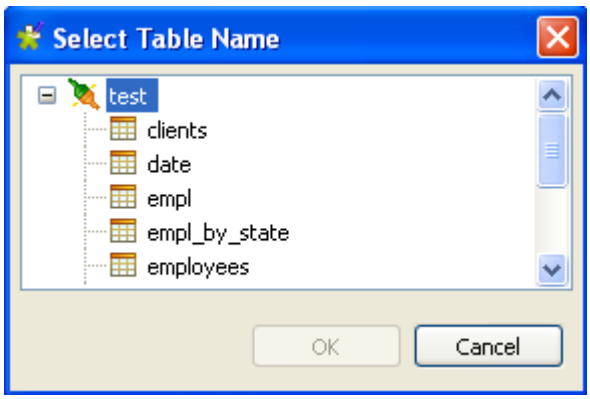
   Under **Column**, click in the fields to enter the corresponding column names.

   Click the field under **Type** to define the type of data.

   Click **OK** to close the schema editor.

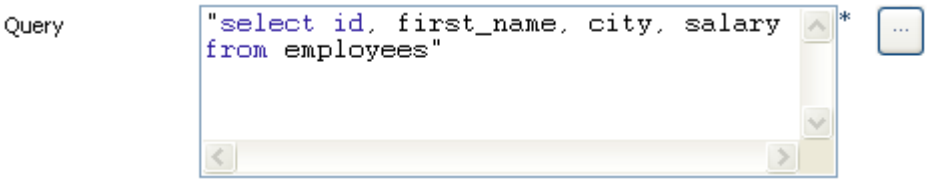
5. Next to the **Table Name** field, click the **[...]** button to select the database table of interest.

   A dialog box displays a tree diagram of all the tables in the selected database:

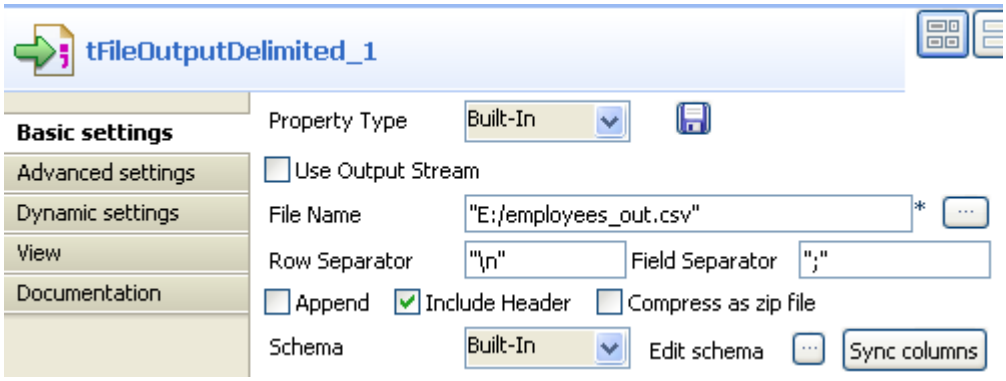


6. Click the table of interest and then click **OK** to close the dialog box.

7. In the **Query** box, enter the query required to retrieve the desired columns from the table.



8. Double-click **tFileOutputDelimited** to set its **Basic settings** in the **Component** tab.



9. Next to the **File Name** field, click the **[...]** button to browse your directory to where you want to save the output file, then enter a name for the file.

   Select the **Include Header** check box to retrieve the column names as well as the data.

10. Save the Job.

Executing the Job

The results below can be found after **F6** is pressed to run the Job.

```
1  id;first_name;city;salary
2  1;Martin;Sacramento;9011
3  2;Zachary;Atlanta;8118
4  3;James;Hartford;5087
5  4;Herbert;Charleston;9233
6  5;Herbert;Hartford;6289
7  6;Ulysses;Nashville;6269
```

As shown above, the output file is written with the desired column names and corresponding data, retrieved from the database:

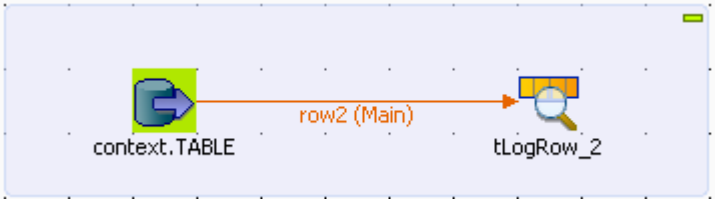
The Job can also be run in the **Traces Debug** mode, which allows you to view the rows as they are being written to the output file, in the workspace.

Scenario 2: Using context parameters when reading a table from a MySQL database

In this scenario, we will read a table from a MySQL database, using a context parameter to refer to the table name.

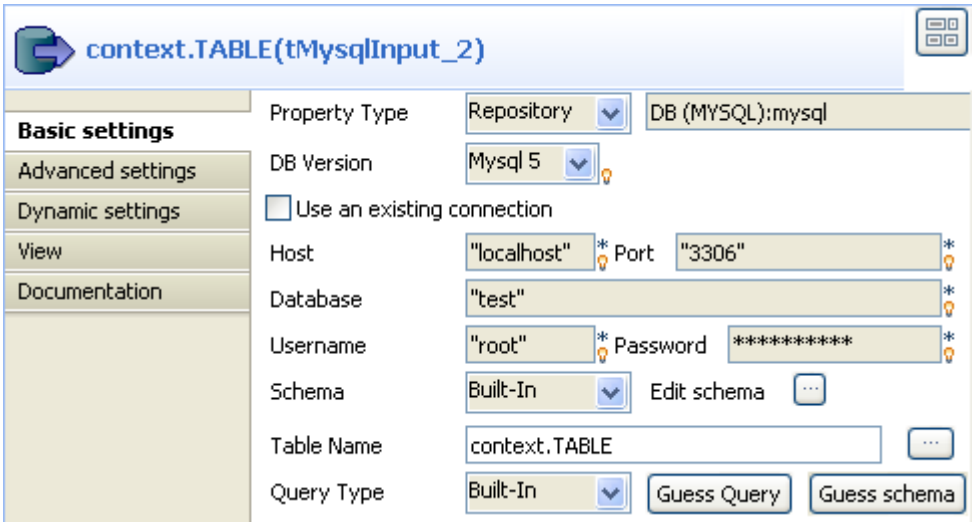
Dragging and dropping components and linking them together

1. Drop **tMysqlInput** and **tLogRow** from the **Palette** onto the workspace.
2. Link **tMysqlInput** to **tLogRow** using a **Row > Main** connection.



Configuring the components

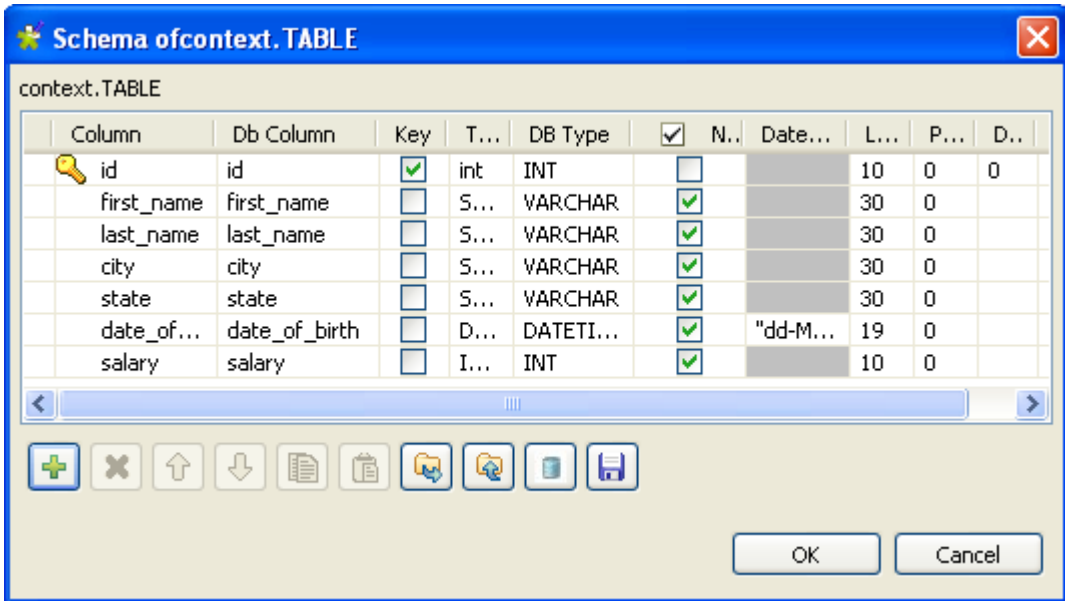
1. Double-click **tMysqlInput** to open its **Basic Settings** view in the **Component** tab.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Define the DB connection details in the corresponding fields.
3. Click **Edit schema** to define the desired schema.

The schema editor opens:



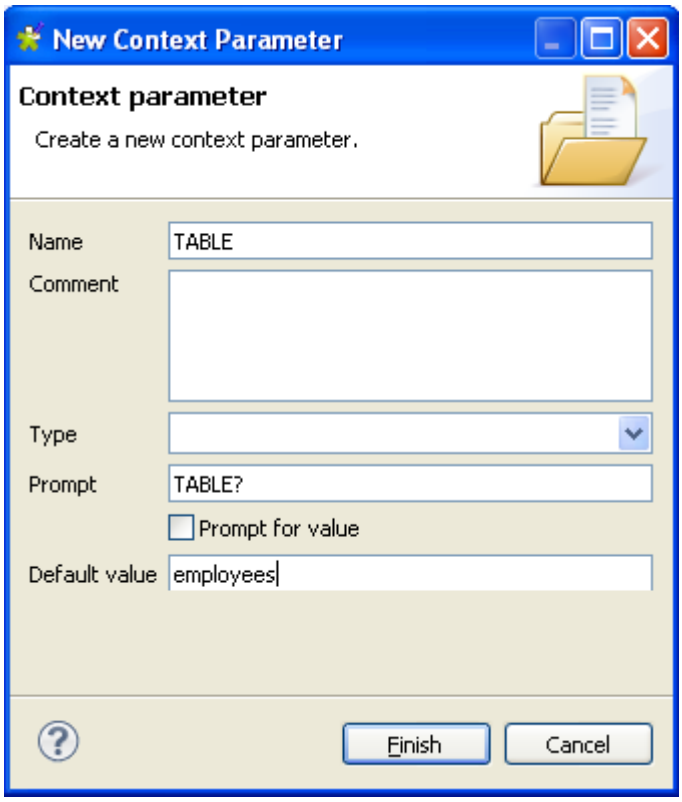
4. Click the **[+]** button to add the rows that you will use to define the schema, seven columns in this example: *id*, *first_name*, *last_name*, *city*, *state*, *date_of_birth* and *salary*.

   Under **Column**, click the fields to enter the corresponding column names.

   Click the fields under **Type** to define the type of data.

   Click **OK** to close the schema editor.

5. Put the cursor in the **Table Name** field and press **F5** for context parameter setting.



For more information about context settings, see *Talend Open Studio for Big Data User Guide*.

6. Keep the default setting in the **Name** field and type in the name of the database table in the **Default value** field, *employees* in this case.

7. Click **Finish** to validate the setting.

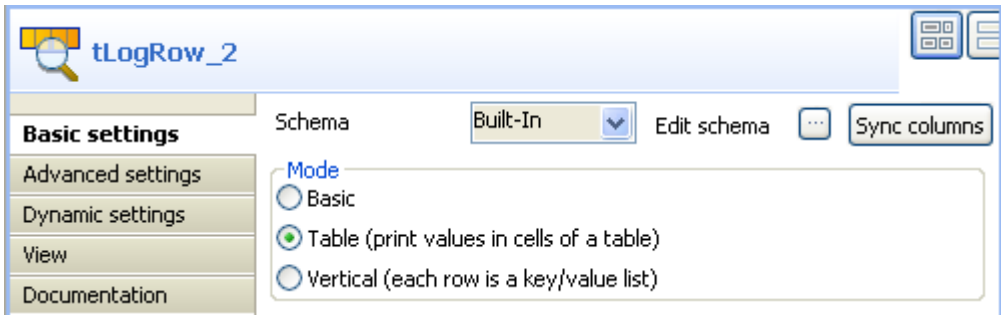
   The context parameter *context.TABLE* automatically appears in the **Table Name** field.

8. Click **Guess Query** to get the query statement.

   In this use case, we want to read the records with the salary above 8000. Therefore, we add a `Where` clause and the final query statement is as follows:

```
"SELECT
  "+context.TABLE+".`id`,
  "+context.TABLE+".`first_name`,
  "+context.TABLE+".`last_name`,
  "+context.TABLE+".`city`,
  "+context.TABLE+".`state`,
  "+context.TABLE+".`date_of_birth`,
  "+context.TABLE+".`salary`
FROM "+context.TABLE+"
WHERE
  "+context.TABLE+".`salary` > 8000"
```

9. Double-click **tLogRow** to set its **Basic Settings** in the **Component** tab.



10. In the **Mode** area, select **Table (print values in cells of a table)** for a better display of the results.

11. Save the Job.

Executing the Job

The results below can be found after **F6** is pressed to run the Job.

```
.--+----------+---------+----------+-----+-------------+------.
|                          tLogRow_2                           |
|=-+----------+---------+----------+-----+-------------+-----=|
|id|first_name|last_name|city      |state|date_of_birth|salary|
|=-+----------+---------+----------+-----+-------------+-----=|
|1 |Martin    |Roosevelt|Sacramento|ND   |19-10-1950   |9011  |
|2 |Zachary   |Johnson  |Atlanta   |ND   |14-11-1978   |8118  |
|4 |Herbert   |Harrison |Charleston|AB   |15-08-1963   |9233  |
'--+----------+---------+----------+-----+-------------+------'
```

As shown above, the records with the salary greater than 8000 are retrieved.

tMysqlLastInsertId

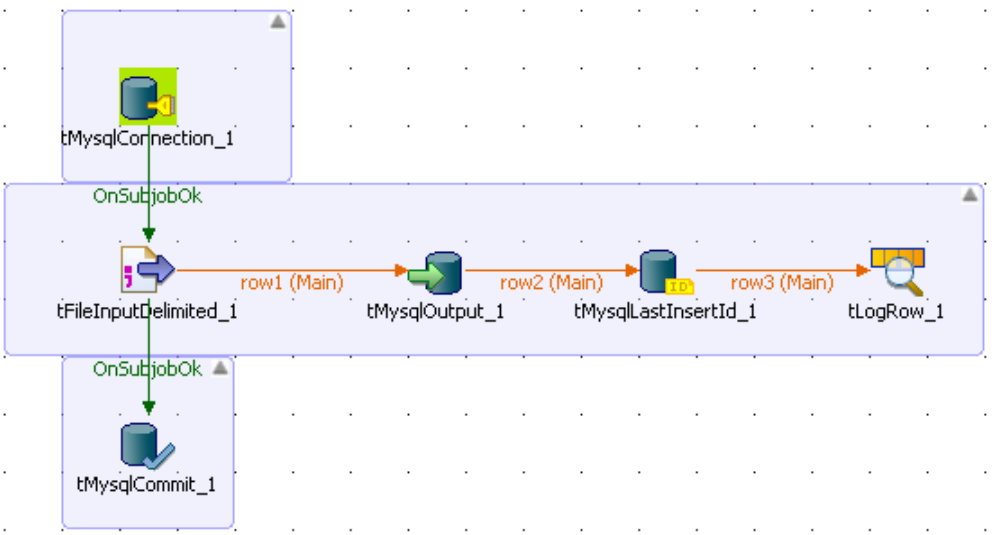
tMysqlLastInsertId properties

Component family	Databases	
Function	**tMysqlLastInsertId** fetches the last inserted ID from a selected MySQL Connection.	
Purpose	**tMysqlLastInsertId** obtains the primary key value of the record that was last inserted in a Mysql table by a user.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Component list	Select the relevant **tMysqlConnection** component in the list if more than one connection is planned for the current job.
Usage	This component is to be used as an intermediary component. *If you use this component with **tMySqlOutput**, verify that the **Extend Insert** check box in the **Advanced Settings** tab is not selected. **Extend Insert** allows you to make a batch insertion, however, if the check box is selected, only the ID of the last line in the last batch will be returned.*	
Limitation	n/a	

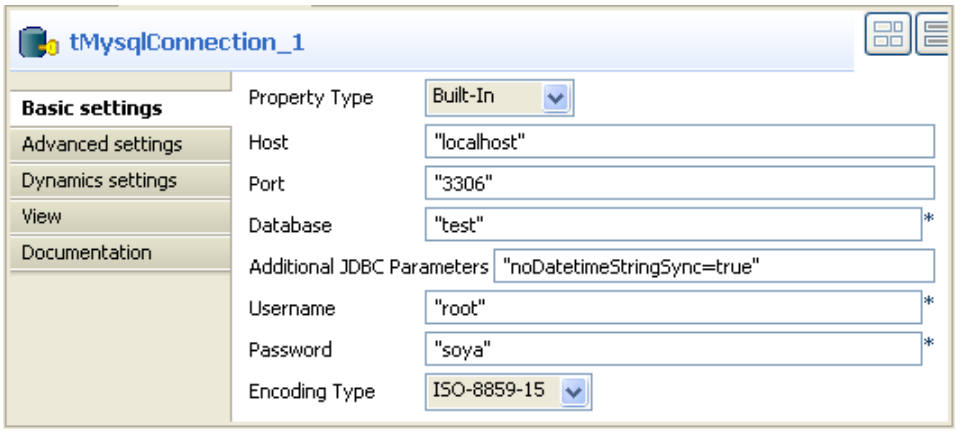
Scenario: Get the ID for the last inserted record

The following Java scenario creates a job that opens a connection to Mysql database, writes the defined data into the database, and finally fetches the last inserted ID on the existing connection.

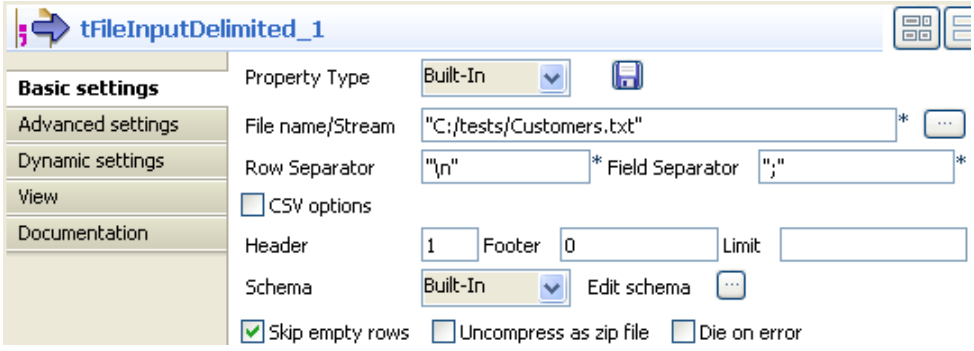
- Drop the following components from the **Palette** onto the design workspace: **tMySqlConnection**, **tMySqlCommit**, **tFileInputDelimited**, **tMySqlOutput**, **tMysqlLastInsertId**, and **tLogRow**.
- Connect **tMySqlConnection** to **tFileInputDelimited** using an **OnSubjobOk** link.
- Connect **tFileInputDelimited** to **tMySqlCommit** using an **OnSubjobOk** link.
- Connect **tFileInputdelimited** to the three other components using **Row Main** links.



- In the design workspace, select **tMysqlConnection**.
- Click the **Component** tab to define the basic settings for **tMysqlConnection**.
- In the **Basic settings** view, set the connection details.

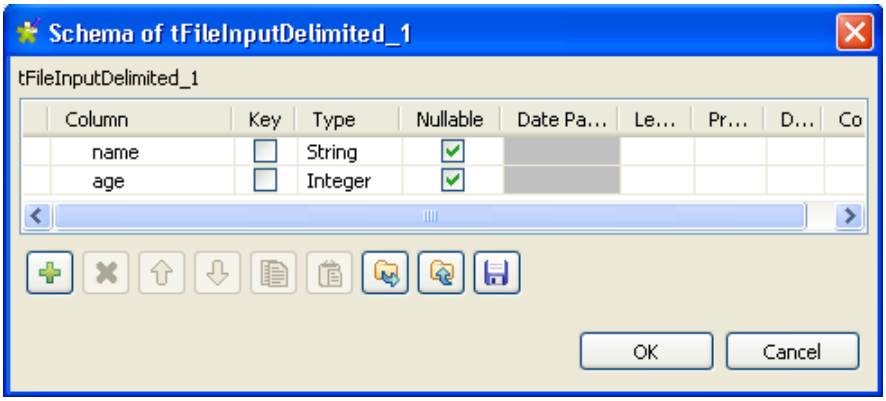


- In the design workspace, select **tMysqlCommit** and click the **Component** tab to define its basic settings.
- On the **Component List**, select the relevant **tMysqlConnection** if more than one connection is used.
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings of **tFileInputDelimited**.



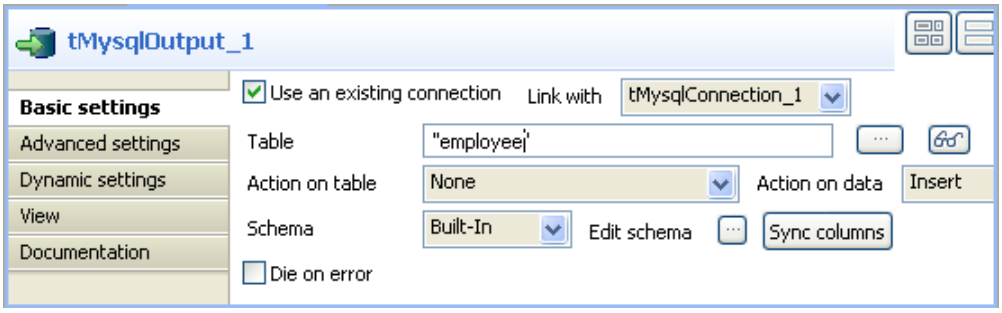
- Fill in a path to the processed file in the **File Name** field. The file used in this example is *Customers*.
- Define the **Row separator** that allow to identify the end of a row. Then define the **Field separator** used to delimit fields in a row.
- Set the header, the footer and the number of processed rows as necessary. In this scenario, we have one header.
- Click the three-dot button next to **Edit Schema** to define the data to pass on to the next component.

Related topics: see *Talend Open Studio for Big Data User Guide*.

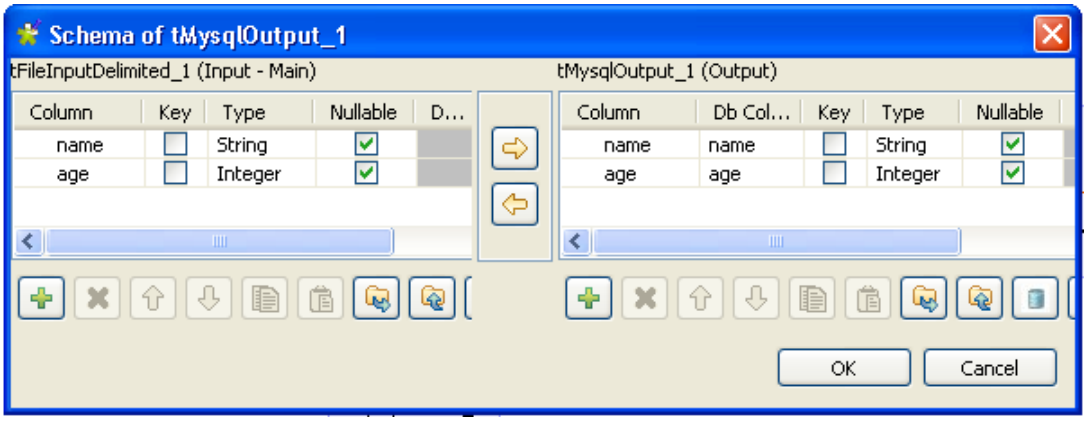


In this scenario, the schema consists of two columns, *name* and *age*. The first holds three employees' names and the second holds the corresponding age for each.

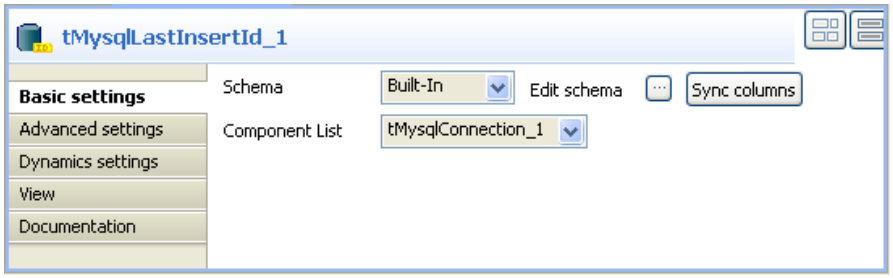
- In the design workspace, select **tMySqlOutput**.
- Click the **Component** tab to define the basic settings of **tMySqlOuptput**.



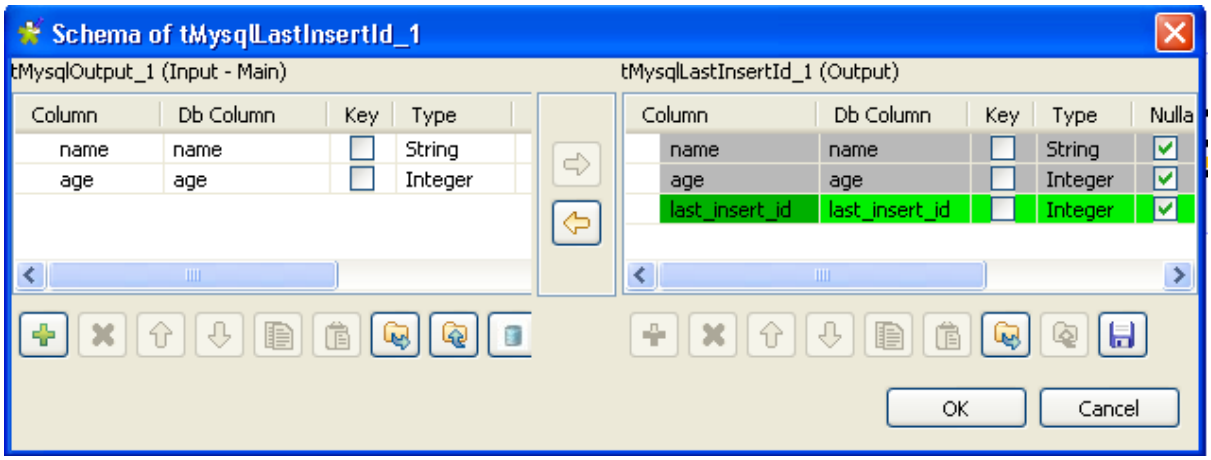
- Select the **Use an existing connection** check box.
- In the **Table** field, enter the name of the table where to write the employees' list, in this example: *employee*.
- Select relevant actions on the **Action on table** and **Action on data** lists. In this example, no action is carried out on table, and the action carried out on data is *Insert*.
- Click **Sync columns** to synchronize columns with the previous component. In this example, the schema to be inserted into the MySql database table consists of the two columns *name* and *age*.



- In the design workspace, select **tMySqlLastInsertId**.
- Click the **Component** tab to define the basic settings of **tMySqlLastInserId**.



- On the **Component List**, select the relevant **tMysqlConnection**, if more than one connection is used.
- Click **Sync columns** to synchronize columns with the previous component. In the output schema of **tMySqlLastInsertId**, you can see the read-only column *last_insert_id* that will fetch the last inserted ID on the existing connection.



You can select the data type *Long* from the **Type** drop-down list in case of a huge number of entries.

- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*.
- Save your job and press **F6** to execute it.

```
Starting job lastinserted at 11:12 01/09/2008.
.--------+---+--------------.
|          tLogRow_1          |
|=--------+---+--------------=|
|name     |age|last_insert_id|
|=--------+---+--------------=|
|Marie    |24 |40            |
|Patrick  |22 |41            |
|Pierrick|27 |42            |
'--------+---+--------------'

Job lastinserted ended at 11:12 01/09/2008. [exit code=0]
```

tMysqlLastInsertId fetched the last inserted ID for each line on the existing connection.

tMysqlOutput

tMysqlOutput properties

Component family	Databases/MySQL	
Function	**tMysqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tMysqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the MySQL version you are using.
	Use an existing connection	Select this check box when using a configured **tMysqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

		Truncate table: The table content is quickly deleted. However, you will not be able to rollback the operation.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, the job stops. **Update**: Make changes to existing entries. **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or creates them if they do not exist. **Delete**: Remove entries corresponding to the input flow. **Replace**: Add new entries to the table. If an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted. **Insert or update on duplicate key or unique index**: Add entries if the inserted value does not exist or update entries if the inserted value already exists and there is a risk of violating a unique index or primary key. **Insert Ignore**: Add only new rows to prevent duplicate key errors. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row in error and complete the process for error-free rows. If needed, you can retrieve the rows in error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.

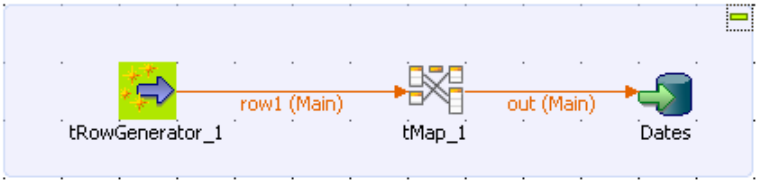
	Extend Insert	Select this check box to carry out a bulk insert of a defined set of lines instead of inserting lines one by one. The gain in system performance is considerable. **Number of rows per insert**: enter the number of rows to be inserted per operation. Note that the higher the value specified, the lower performance levels shall be due to the increase in memory demands. This option is not compatible with the **Reject** link. You should therefore clear the check box if you are using a **Row > Rejects** link with this component. *If you are using this component with **tMysqlLastInsertID**, ensure that the **Extend Insert** check box in **Advanced Settings** is not selected. **Extend Insert** allows for batch loading, however, if the check box is selected, only the ID of the last line of the last batch will be returned.*
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected, the **Update** or the **Delete** option in the **Action on data** field.
	Commit every	Number of rows to be included in the batch before it is committed to the DB. This option ensures transaction quality (but not rollback) and, above all, a higher performance level.
	Additional Columns	This option is not available if you have just created the DB table (even if you delete it beforehand). This option allows you to call SQL functions to perform actions on columns, provided that these are not insert, update or delete actions, or actions that require pre-processing.
		Name: Type in the name of the schema column to be altered or inserted.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the data in the corrsponding column.
		Position: Select **Before**, **Replace** or **After**, depending on the action to be performed on the reference column.
		Reference column: Type in a reference column that **tMySqlOutput** can use to locate or replace the new column, or the column to be modified.
	Use field options	Select this check box to customize a request, particularly if multiple actions are being carried out on the data.
	Use Hint Options	Select this check box to activate the hint configuration area which helps you optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */`. - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	Enable debug mode	Select this check box to display each step involved in the process of writing data in the database.
	Use duplicate key update mode insert	Updates the values of the columns specified, in the event of duplicate primary keys.: **Column**: Between double quotation marks, enter the name of the column to be updated. **Value**: Enter the action you want to carry out on the column.

		To use this option you must first of all select the **Insert** mode in the **Action on data** list found in the **Basic Settings** view.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a MySQL database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

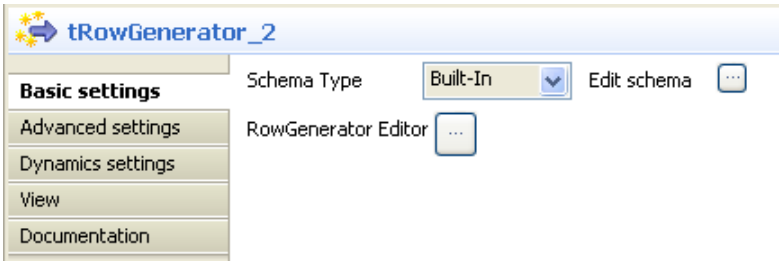
Scenario 1: Adding a new column and altering data in a DB table

This Java scenario is a three-component job that aims at creating random data using a **tRowGenerator**, duplicating a column to be altered using the **tMap** component, and eventually altering the data to be inserted based on an SQL expression using the **tMysqlOutput** component.

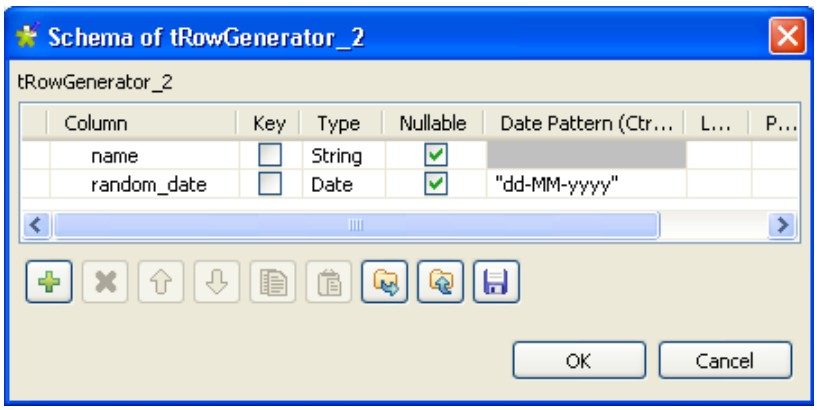
- Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tMap** and **tMySQLOutput**.
- Connect **tRowGenerator**, **tMap**, and **tMysqlOutput** using the **Row Main** link.



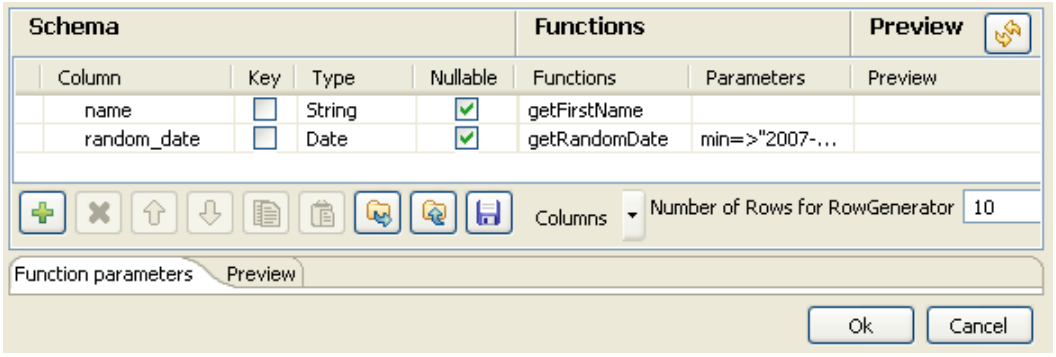
- In the design workspace, select **tRowGenerator** to display its **Basic settings** view.



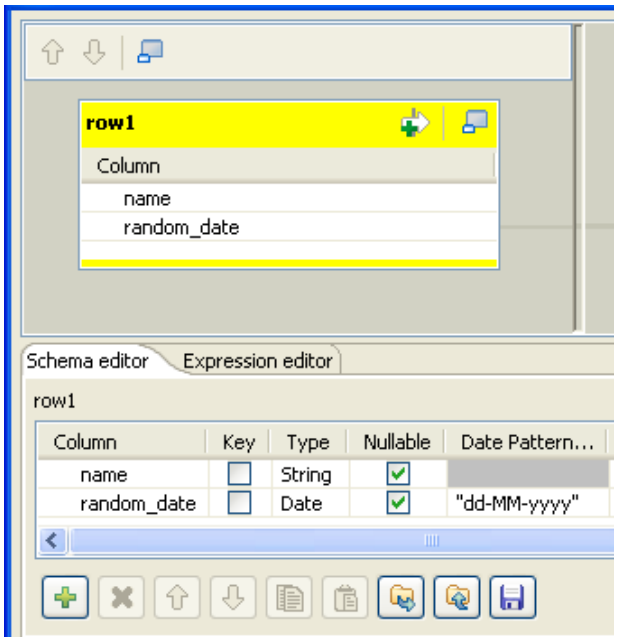
- Click the **Edit schema** three-dot button to define the data to pass on to the **tMap** component, two columns in this scenario, *name* and *random_date*.



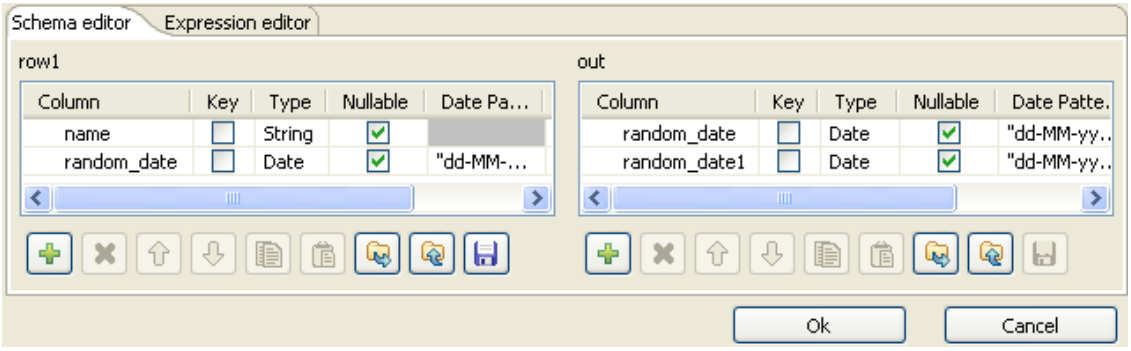
- Click **OK** to close the dialog box.
- Click the **RowGenerator Editor** three-dot button to open the editor and define the data to be generated.



- Click in the corresponding **Functions** fields and select a function for each of the two columns, `getFirstName` for the first column and `getrandomDate` for the second column.
- In the **Number of Rows for Rowgenerator** field, enter 10 to generate ten first name rows and click **Ok** to close the editor.
- Double-click the **tMap** component to open the Map editor. The Map editor opens displaying the input metadata of the **tRowGenerator** component.

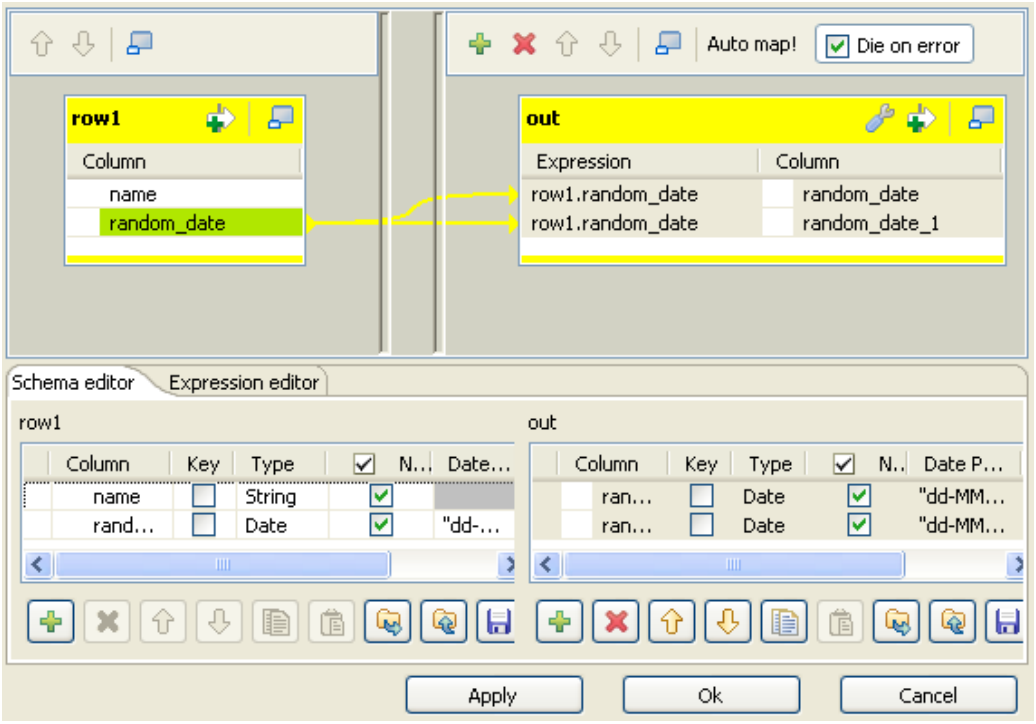


- In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and define the first as *random_date* and the second as *random_date1*.



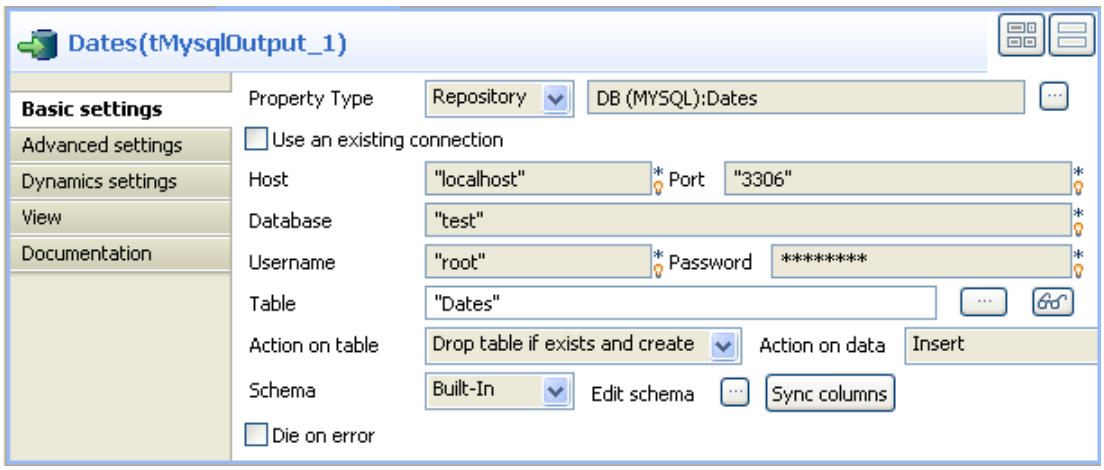
In this scenario, we want to duplicate the *random_date* column and adapt the schema in order to alter the data in the output component.

- In the Map editor, drag the *random_date* row from the input table to the *random_date* and *random_date1* rows in the output table.

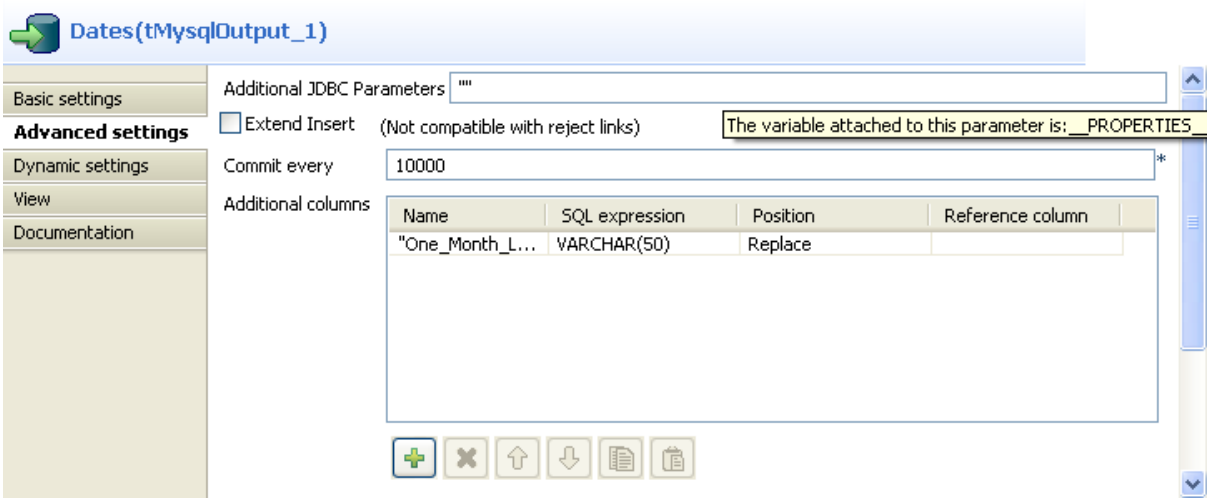


- Click **OK** to close the editor.

- In the design workspace, double-click the **tMysqlOutput** component to display its **Basic settings** view and set its parameters.



- Define the DB connection details in the corresponding fields.
- Click the three-dot button next to the **Table** field and select the table to be altered, *Dates* in this scenario.
- On the **Action on table** list, select **Drop table if exists and create**, select *Insert* on the **Action on data** list.
- If needed, click **Sync columns** to synchronize with the columns coming from the **tMap** component.
- Click the **Advanced settings** tab to display the corresponding view and set the advanced parameters.



- In the **Additional Columns** area, set the alteration to be performed on columns.

  In this scenario, the *One_month_later* column replaces *random_date_1*. Also, the data itself gets altered using an SQL expression that adds one month to the randomly picked-up date of the *random_date_1* column. ex: 2007-08-12 becomes 2007-09-12.

  -Enter *One_Month_Later* in the **Name** cell.

  -In the **SQL expression** cell, enter the relevant addition script to be performed, `"adddate(Random_date, interval 1 month)"` in this scenario.

  -Select **Replace** on the **Position** list.

  -Enter *Random_date1* on the **Reference column** list.

For this job we duplicated the *random_date_1* column in the DB table before replacing one instance of it with the *One_Month_Later* column. The aim of this workaround was to be able to view upfront the modification performed.

- Save your job and press **F6** to execute it.

The new *One_month_later* column replaces the *random_date1* column in the DB table and adds one month to each of the randomly generated dates.

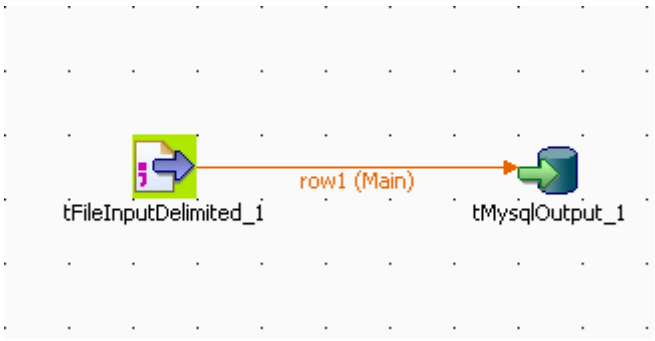
Random_date	One_Month_Later
2007-03-13 21:52:09	2007-04-13 21:52:09
2008-11-02 15:33:29	2008-12-02 15:33:29
2008-05-17 18:47:15	2008-06-17 18:47:15
2007-09-06 17:44:36	2007-10-06 17:44:36
2008-11-17 11:46:36	2008-12-17 11:46:36
2007-05-19 00:46:58	2007-06-19 00:46:58
2008-04-20 15:36:00	2008-05-20 15:36:00
2007-07-24 21:06:08	2007-08-24 21:06:08
2008-04-03 13:24:17	2008-05-03 13:24:17
2007-06-29 13:22:23	2007-07-29 13:22:23

Related topic: see section *tDBOutput properties*.

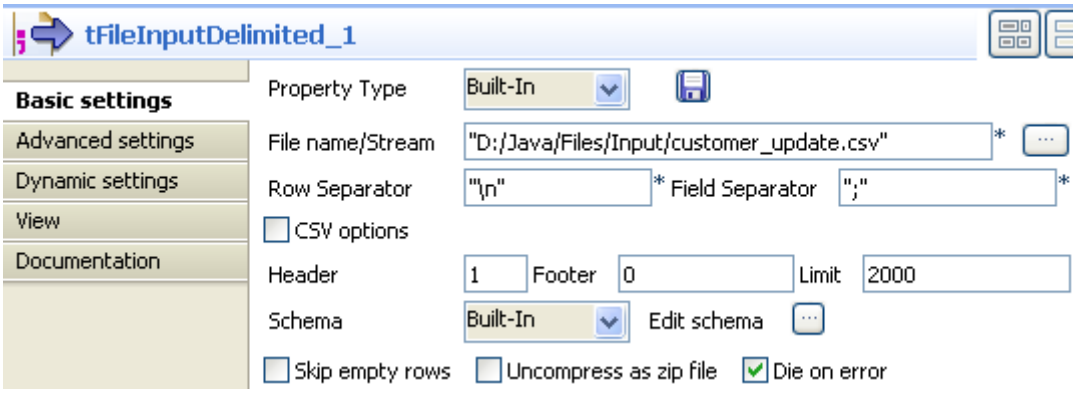
Scenario 2: Updating data in a database table

This Java scenario describes a two-component Job that updates data in a MySQL table according to that in a delimited file.

- Drop **tFileInputDelimited** and **tMysqlOutput** from the **Palette** onto the design workspace.
- Connect the two components together using a **Row Main** link.



- Double-click **tFileInputDelimited** to display its **Basic settings** view and define the component properties.



- In the **File Name** field, click the three-dot button and browse to the source delimited file that contains the modifications to propagate in the MySQL table.

  In this example, we use the *customer_update* file that holds four columns: *id*, *CustomerName*, *CustomerAddress* and *idState*. Some of the data in these four columns is different from that in the MySQL table.

id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Marks, and Kaplan Ltd.	1949 Cloverdale Rd.	9

- Define the row and field separators used in the source file in the corresponding fields.
- If needed, set **Header**, **Footer** and **Limit**.

  In this example, **Header** is set to 1 since the first row holds the names of columns, therefore it should be ignored. Also, the number of processed lines is limited to 2000.

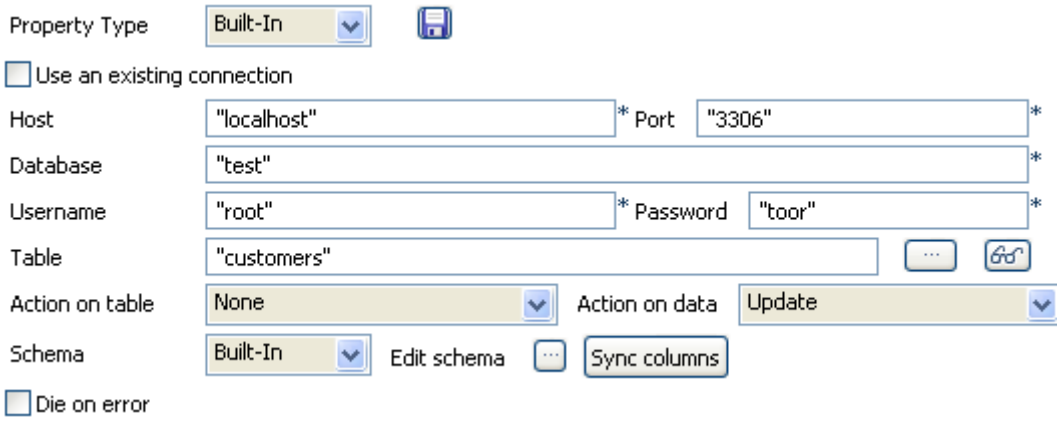
- Click the three-dot button next to **Edit Schema** to open a dialog box where you can describe the data structure of the source delimited file that you want to pass to the component that follows.

Column	Key	Type	Nullable
id	☑	Integer	☑
CustomerName	☐	String	☑
CustomerAddress	☐	String	☑
idState	☐	Integer	☑

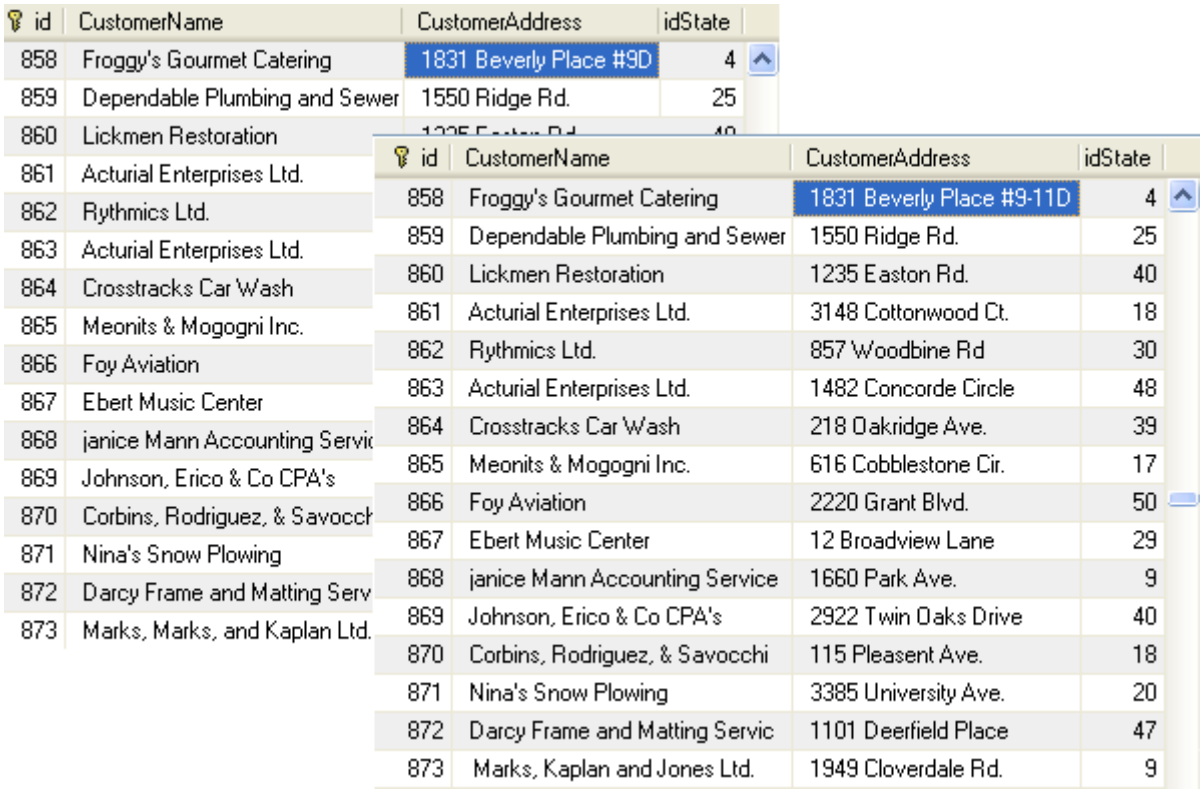
- Select the **Key** check box(es) next to the column name(s) you want to define as key column(s).

It is necessary to define at least one column as a key column for the Job to be executed correctly. Otherwise, the Job is automatically interrupted and an error message displays on the console.

- In the design workspace, double-click **tMysqlOutput** to open its **Basic settings** view where you can define its properties.



- Click **Sync columns** to retrieve the schema of the preceding component. If needed, click the three-dot button next to **Edit schema** to open a dialog box where you can check the retrieved schema.
- Fill in the database connection information in the corresponding fields.
- In the **Table** field, enter the name of the table to update.
- From the **Action on table** list, select the operation you want to perform, **None** in this example since the table already exists.
- From the **Action on data** list, select the operation you want to perform on the data, **Update** in this example.
- Save your Job and press **F6** to execute it.



id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration		
861	Acturial Enterprises Ltd.		
862	Rythmics Ltd.		
863	Acturial Enterprises Ltd.		
864	Crosstracks Car Wash		
865	Meonits & Mogogni Inc.		
866	Foy Aviation		
867	Ebert Music Center		
868	janice Mann Accounting Servic		
869	Johnson, Erico & Co CPA's		
870	Corbins, Rodriguez, & Savocch		
871	Nina's Snow Plowing		
872	Darcy Frame and Matting Serv		
873	Marks, Marks, and Kaplan Ltd.		

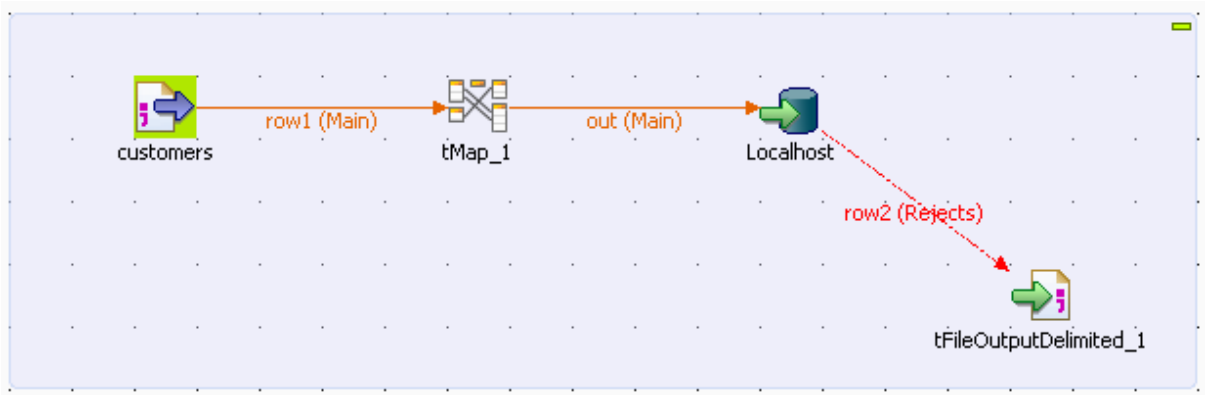
id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9-11D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 Cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Kaplan and Jones Ltd.	1949 Cloverdale Rd.	9

Using you DB browser, you can verify if the MySQL table, *customers*, has been modified according to the delimited file.

In the above example, the database table has always the four columns *id*, *CustomerName*, *CustomerAddress* and *idState*, but certain fields have been modified according to the data in the delimited file used.

Scenario 3: Retrieve data in error with a Reject link

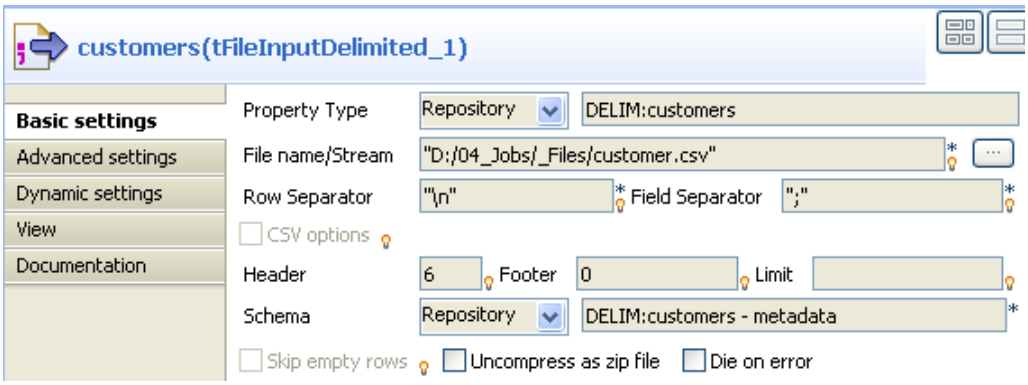
This scenario describes a four-component Job that carries out migration from a customer file to a MySQL database table and redirects data in error towards a CSV file using a **Reject** link.



- Drop a **tFileInputDelimited** component from the family **File** > **Input**, in the **Palette**, and fill in its properties manually in the **Component** tab.
- From the **Palette**, drop a **tMap** from the **Processing** family onto the workspace.
- Drop a **tMysqlOutput** from the **Databases** family in the **Palette** and fill in its properties manually in the **Component** tab.

For more information, see *Talend Open Studio for Big Data User Guide*.

- From the **Palette**, select a **tFileOutputDelimited** from the **File** > **Output** family, and drop it onto the workspace.
- Link the **customers** component to the **tMap** component, and the **tMap** and **Localhost** with a **Row Main** link. Name this second link *out*.
- Link the **Localhost** to the **tFileOutputDelimited** using a **Row** > **Reject** link.
- Double-click the **customers** component to display the **Component** view.



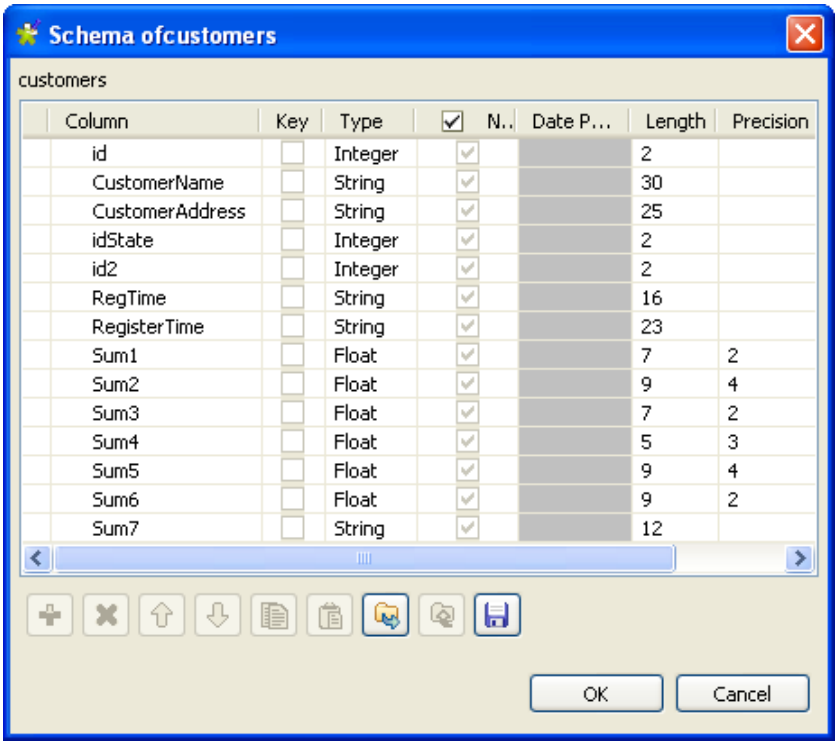
*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

- Define the DB connection details in the corresponding fields.
- Click the **[...]** button next to the **File Name** field, and fill in the path and the name of the file you want to use.
- In the **Row** and **Field Separator** fields, type in between inverted commas the row and field separator used in the file.

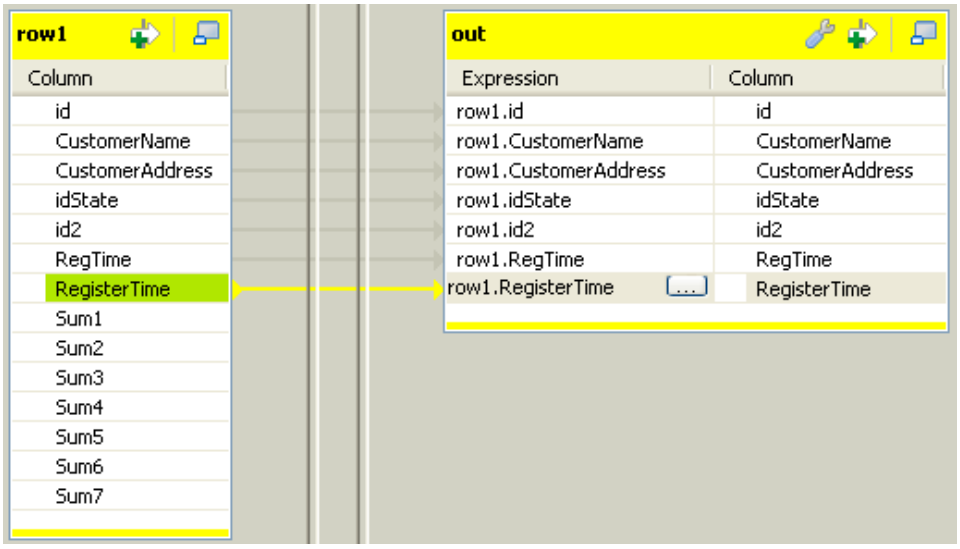
- In the **Header**, **Footer** and **Limit** fields, type in the number of headers and footers to ignore, and the number of rows to which processing should be limited.

- Click the **[...]** button next to the **Edit schema** field, and set the schema manually.

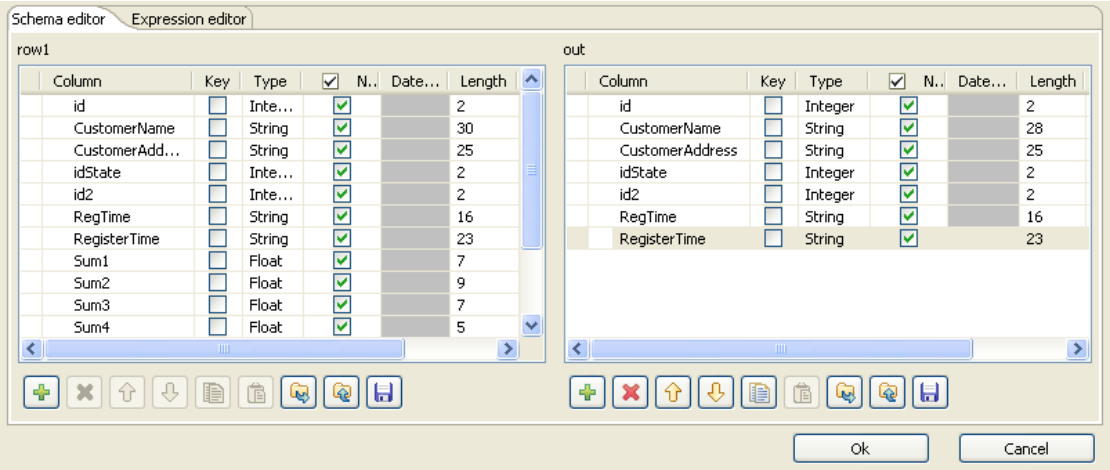
The schema is as follows:



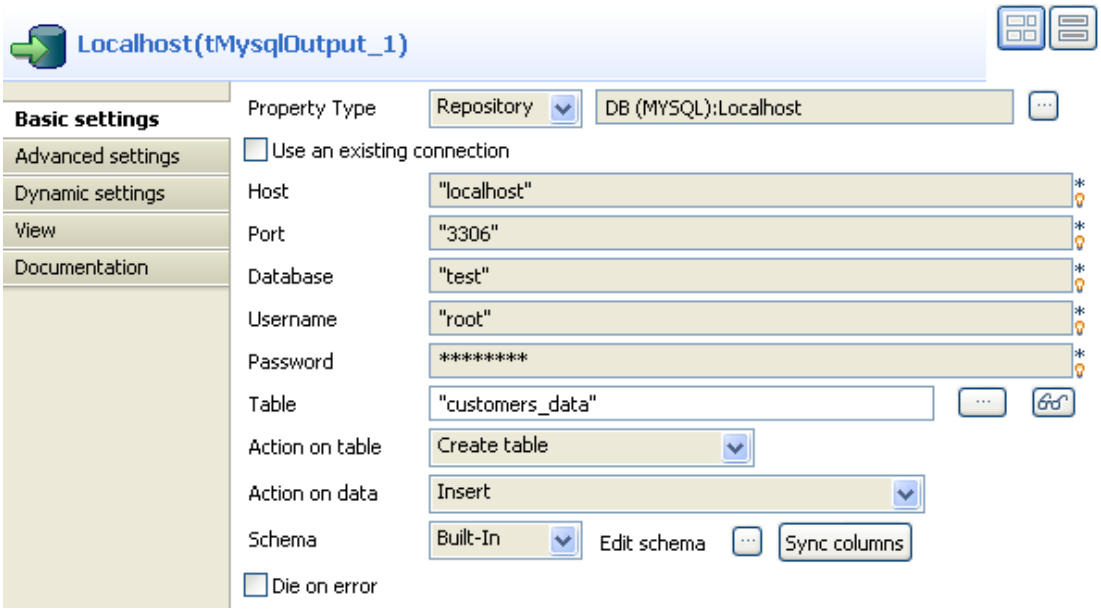
- Double-click the **tMap** component to open its editor.



- Select the *id*, *CustomerName*, *CustomerAddress*, *idSate*, *id2*, *RegTime* and *RegisterTime* columns on the table on the left and drop them on the **out** table, on the right.

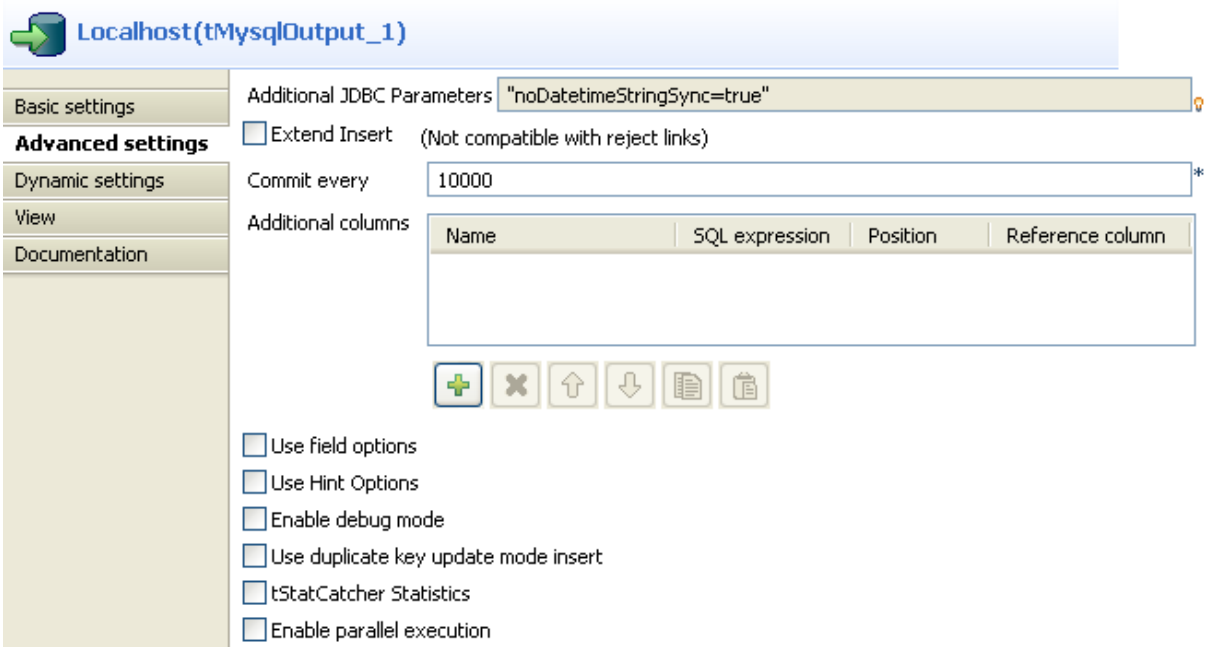


- In the **Schema editor** area, at the bottom of the **tMap** editor, in the right table, change the length of the **CustomerName** column to *28* to create an error. Thus, any data for which the length is greater than 28 will create errors, retrieved with the **Reject** link.
- Click **OK**.
- In the workspace, double-click the output **Localhost** component to display its **Component** view.

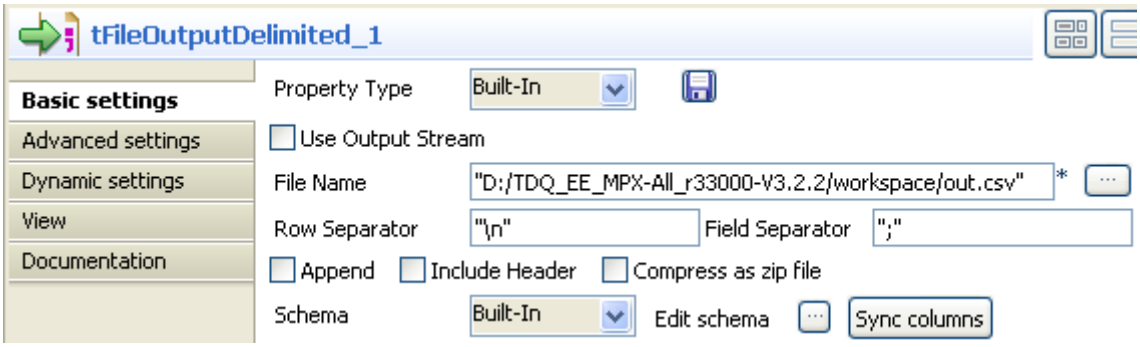


*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

- Define the DB connection details in the corresponding fields.
- In the **Table** field, type in the name of the table to be created. In this scenario, we call it *customers_data*.
- In the **Action on data** list, select the **Create table** option.
- Click the **Sync columns** button to retrieve the schema from the previous component.
- Make sure the **Die on error** check box isn't selected, so that the Job can be executed despite the error you just created.
- Click the **Advanced settings** tab of the **Component** view to set the advanced parameters of the component.



- Deselect the **Extend Insert** check box which enables you to insert rows in batch, because this option is not compatible with the **Reject** link.
- Double-click the **tFileOutputDelimited** component to set its properties in the **Component** view.



- Click the **[...]** button next to the **File Name** field to fill in the path and name of the output file.
- Click the **Sync columns** button to retrieve the schema of the previous component.
- Save your Job and press **F6** to execute it.

1	Griffith Paving	talend@apres	7	41	##########	2001-01-17 06:26	Data truncation: Data too long
5	Terrinni & Sor	770 Exmoor F	5	9	##########	1982-04-19 10:26	Data truncation: Data too long
10	Elle Hypnosis	2032 Northbro	1	7	##########	1975-06-10 20:20	Data truncation: Data too long
15	Darcy Frame	1633 McGove	21	28	##########	2001-01-07 20:40	Data truncation: Data too long
19	Salt & Peppe	965 Marion P	44	33	##########	2006-01-24 10:53	Data truncation: Data too long
31	Arthur M. Fei	1220 Heather	37	24	##########	1974-07-24 20:48	Data truncation: Data too long
36	Got It All Ele	629 Kincaid A	22	28	##########	1993-10-22 15:26	Data truncation: Data too long
53	Crossroads C	2267 Anderso	5	14	##########	1977-10-24 17:53	Data truncation: Data too long
54	Jay's Cabinet	2024 St.John	48	48	2006-02-29 0	2001-01-10 07:48	Data truncation: Data too long
57	Lee Stairwork	2644 Princeto	41	46	##########	1992-04-15 09:41	Data truncation: Data too long
58	Lambert & Lo	662 Lyons Ci	45	17	##########	2002-11-22 04:26	Data truncation: Data too long
70	janice Mann A	3181 Barkwo	28	35	##########	1983-04-20 15:20	Data truncation: Data too long
72	Michael Mont	1077 Court A	9	9	##########	2000-10-07 00:20	Data truncation: Data too long
83	Dean Barnett	2922 Twin Oa	18	46	##########	2006-03-23 10:14	Data truncation: Data too long
96	Doerner's Aw	616 Cobblest	26	28	##########	1993-05-03 04:33	Data truncation: Data too long
##	Auto Interior (	844 Spruce S	48	40	##########	2006-04-25 13:14	Data truncation: Data too long
##	First National	465 Stratford	5	41	##########	2003-09-23 01:54	Data truncation: Data too long
##	Terrinni & Sor	1859 Green E	12	44	##########	1986-12-26 05:48	Data truncation: Data too long

The data in error are sent to the delimited file, as well as the error type met. Here, we have: **Data truncation**.

tMysqlOutputBulk

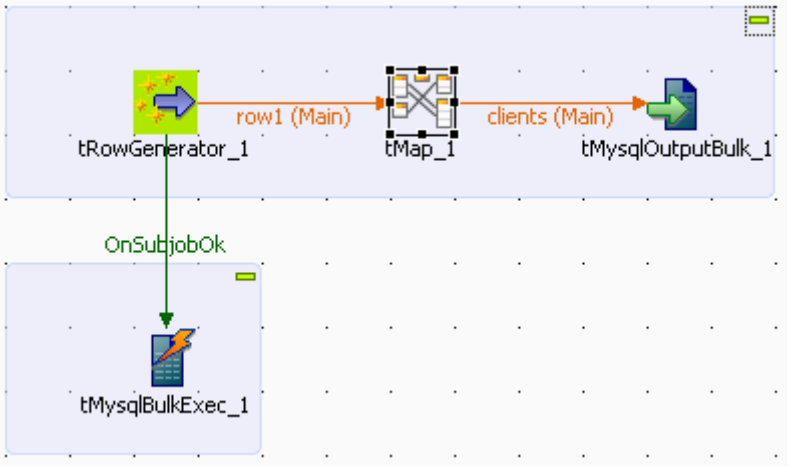
tMysqlOutputBulk properties

The **tMysqlOutputBulk** and **tMysqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT statement used to feed a database. These two steps are fused together in the **tMysqlOutputBulkExec** component, detailed in a separate section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

Component family	Databases/MySQL	
Function	Writes a file with columns based on the defined delimiter and the MySql standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the MySQL database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: "\n" on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Text enclosure	Character used to enclose the text.
	Create directory if does not exist	This check box is selected by default. It creates a directory to hold the output table if required.
	Custom the flush buffer size	Customize the amount of memory used to temporarily store output data. In the **Row number field**, enter the number of rows after which the memory is to be freed again.
	Records contain NULL value	This check box is selected by default. It allows you to take account of NULL value fields. If you clear the check box, the NULL values will automatically be replaced with empty values.
	Check disk space	Select the this check box to throw an exception during execution if the disk is full.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This component is to be used along with **tMySQlBulkExec** component. Used together they offer gains in performance while feeding a MySQL database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Inserting transformed data in MySQL database

This scenario describes a four-component job which aims at fueling a database with data contained in a file, including transformed data. Two steps are required in this job, first step is to create the file, that will then be used in the second step. The first step includes a tranformation phase of the data included in the file.

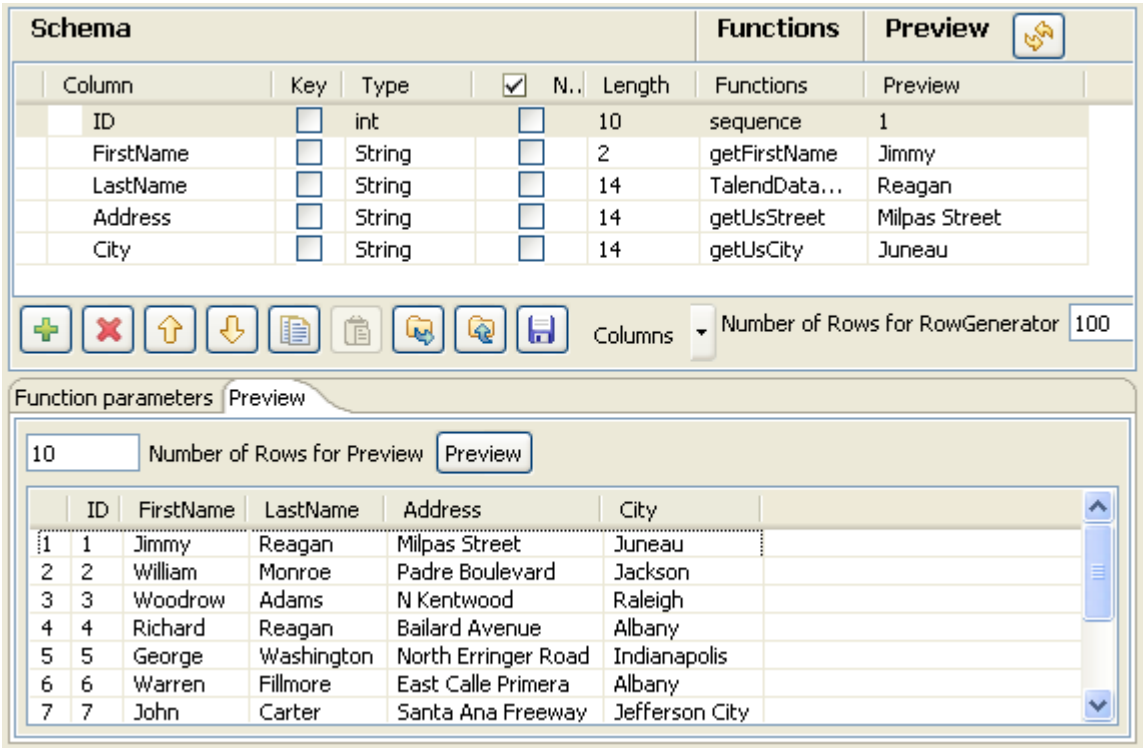


Dropping and linking components

1. Drag and drop a **tRowGenerator**, a **tMap**, a **tMysqlOutputBulk** as well as a **tMysqlBulkExec** component.
2. Connect the main flow using **row Main** links.
3. And connect the start component (**tRowgenerator** in this example) to the **tMysqlBulkExec** using a **trigger** connection, of type **OnComponentOk**.

Configuring the components

1. A **tRowGenerator** is used to generate random data. Double-click on the **tRowGenerator** component to launch the editor.
2. Define the schema of the rows to be generated and the nature of data to generate. In this example, the *clients* file to be produced will contain the following columns: *ID*, *First Name*, *Last Name*, *Address*, *City* which all are defined as string data but the ID that is of integer type.

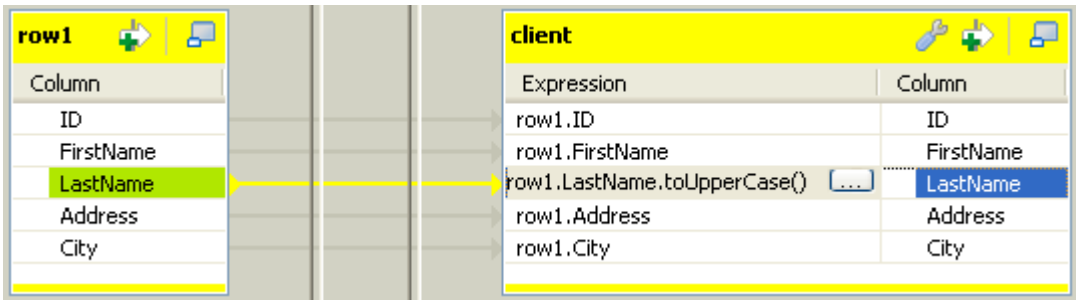


Some schema information don't necessarily need to be displayed. To hide them away, click on **Columns** list button next to the toolbar, and uncheck the relevant entries, such as **Precision** or **Parameters**.

Use the plus button to add as many columns to your schema definition.

Click the Refresh button to preview the first generated row of your output.

3. Then select the **tMap** component to set the transformation.
4. Drag and drop all columns from the input table to the output table.

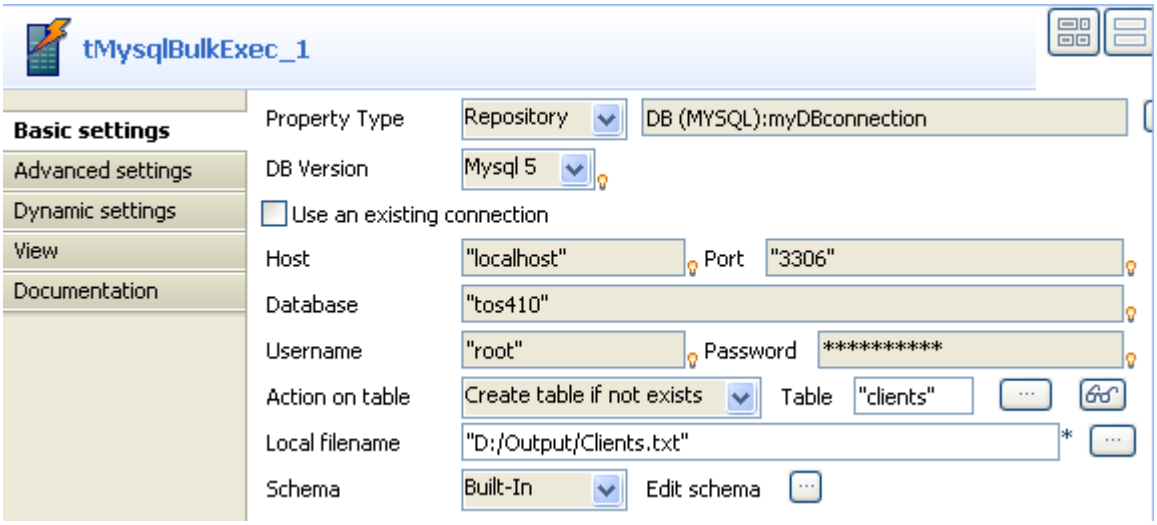


5. Apply the transformation on the *LastName* column by adding `.toUpperCase()` in its expression field.

   Then, click **OK** to validate the transformation.
6. Double-click on the **tMysqlOutputBulk** component.
7. Define the name of the file to be produced in **File Name** field. If the delimited file information is stored in the **Repository**, select it in **Property Type** field, to retrieve relevant data. In this use case the file name is *clients.txt*.

   The schema is propagated from the **tMap** component, if you accepted it when prompted.
8. In this example, don't include the header information as the table should already contain it.
9. Click OK to validate the output.

10. Then double-click on the **tMysqlBulkExec** component to set the INSERT query to be executed.
11. Define the database connection details.



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

12. Set the table to be filled in with the collected data, in the **Table** field.
13. Fill in the column delimiters in the **Field terminated by** area.
14. Make sure the encoding corresponds to the data encoding.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

Resultset 1

ID	First Name	Last Name	Address	City
1	Martin	REAGAN	Hutchinson Rd	Dover¶
2	Herbert	REAGAN	Bailard Avenue	Frankfort¶
3	Franklin	WASHINGTON	Bayshore Freeway	Denver¶
4	Franklin	CARTER	Burnett Road	Frankfort¶
5	Woodrow	MCKINLEY	San Ysidro Blvd	Des Moines¶
6	Bill	QUINCY	Fairview Avenue	Topeka¶
7	Bill	BUREN	Calle Real	Jefferson City¶
8	Andrew	ARTHUR	Santa Ana Freeway	Indianapolis¶
9	Woodrow	FORD	N Harrison St	Augusta¶
10	Calvin	COOLIDGE	Harbor Dr	Frankfort¶

The *clients* database table is filled with data from the file including upper-case *last name* as transformed in the job.

For simple Insert operations that don't include any transformations, the use of **tMysqlOutputBulkExec** allows you to skip a step in the process and thus improves performance.

Related topic: section *tMysqlOutputBulkExec properties*

tMysqlOutputBulkExec

tMysqlOutputBulkExec properties

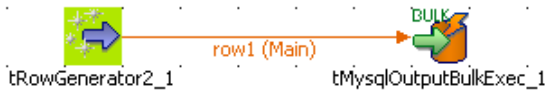
The **tMysqlOutputBulk** and **tMysqlBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT statement used to feed a database. These two steps are fused together in the **tMysqlOutputBulkExec** component.

Component family	Databases/MySQL	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it improves performance during Insert operations to a MySQL database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of MySQL that you are using.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
***tCreateTable** can be used as a substitute for this function.*	*Action on table*	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if doesn't exist**: The table is created if it does not already exist. **Clear a table**: The table content is deleted.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must already exist for the insert operation to succeed
	Local FileName	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select the check box for this option to append new rows to the end of the file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Row separator	String (ex: "\n"on Unix) to distinguish rows.

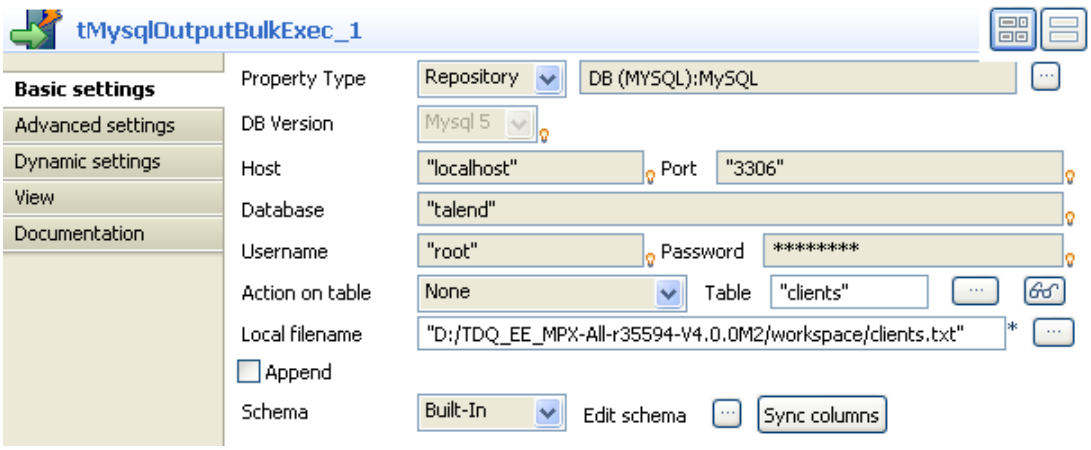
	Field separator	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped
	Text enclosure	Character used to enclose the text.
	Create directory if does not exist	This check box is selected by default. It creates a directory to hold the output table if required.
	Custom the flush buffer size	Customize the amount of memory used to temporarily store output data. In the **Row number field**, enter the number of rows after which the memory is to be freed again.
	Action on data	On the data of the table defined, you can carry out the following opertaions: **Insert records in table**: Add new records to the table. **Update records in table**: Make changes to existing records. **Replace records in table**: Replace existing records with new one. **Ignore records in table:** Ignore existing records or insert the new ones.
	Records contain NULL value	This check box is selected by default. It allows you to take account of NULL value fields. If you clear the check box, the NULL values will automatically be replaced with empty values.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect the log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Scenario: Inserting data in MySQL database

This scenario describes a two-component Job which carries out the same operation as the one described for section *tMysqlOutputBulk properties* and section *tMysqlBulkExec properties*, although no data is transformed.

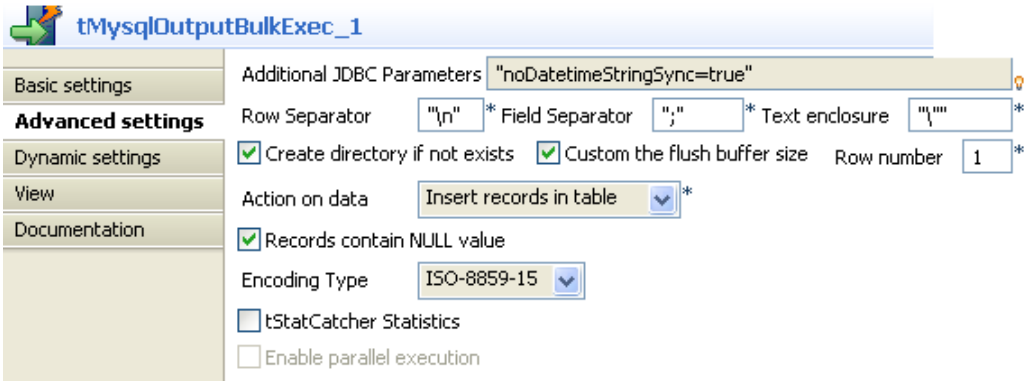


- Drop a **tRowGenerator** and a **tMysqlOutputBulkExec** component from the **Palette** to the design workspace.
- Connect the components using a link such as **Row > Main**.
- Set the **tRowGenerator** parameters the same way as in section *Scenario: Inserting transformed data in MySQL database*. The schema is made of four columns including: *ID*, *First Name*, *Last Name*, *Address* and *City*.
- In the workspace, double-click the **tMysqlOutputBulkExec** to display the **Component** view and set the properties.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

- Define the database connection details in the corresponding fields.
- In the **Action on table** field, select the **None** option as you want to insert the data into a table which already exists.
- In the **Table** field, enter the name of the table you want to populate, the name being *clients* in this example.
- In the **Local filename** field, indicate the access path and the name of the file which contains the data to be added to the table. In this example, the file is *clients.txt*.
- Click on the **Advanced settings** tab to define the component's advanced parameters.



- In the **Action on data** list, select the **Insert records in table** to insert the new data in the table.
- Press **F6** to run the Job.

The result should be pretty much the same as in section *Scenario: Inserting transformed data in MySQL database*, but the data might differ as these are regenerated randomly everytime the Job is run.

tMysqlRollback

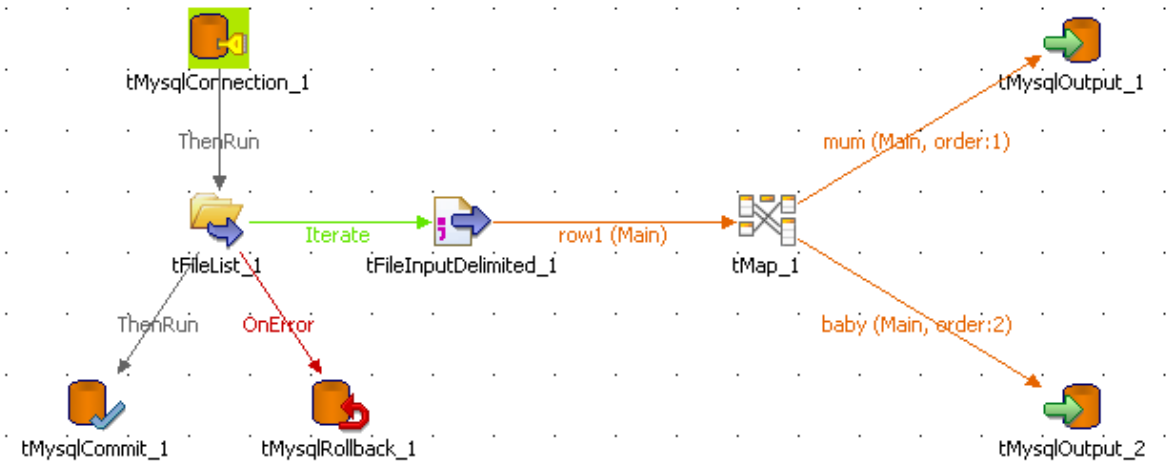
tMysqlRollback properties

This component is closely related to **tMysqlCommit** and **tMysqlConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection** and **tMysqlCommit** components.	
Limitation	n/a	

Scenario: Rollback from inserting data in mother/daughter tables

Based on section *Scenario: Inserting data in mother/daughter tables*, insert a rollback function in order to prevent unwanted commit.



- Drag and drop a **tMysqlRollback** to the design workspace and connect it to the Start component.
- Set the Rollback unique field on the relevant DB connection.

This complementary element to the job ensures that the transaction will not be partly committed.

tMysqlRow

tMysqlRow properties

Component family	Databases/MySQL	
Function	**tMysqlRow** is the specific component for this database query. It executes the SQL query stated in the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tMysqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the MySQL version that you are using.
	Use an existing connection	Select this check box and click the relevant **tMysqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder

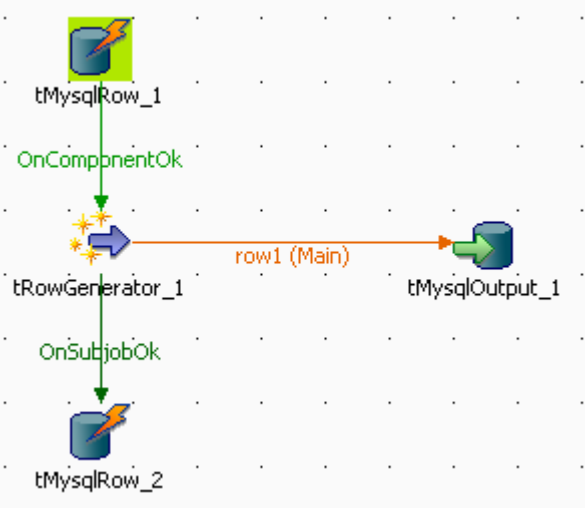
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the **Query** field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query in a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Scenario 1: Removing and regenerating a MySQL table index

This scenario describes a four-component job that removes a table index, applies a select insert action onto a table then regenerates the index.

Linking the components

1. Select and drop the following components onto the design workspace: **tMysqlRow** (x2), **tRowGenerator**, and **tMysqlOutput**.
2. Link the first **tMysqlRow** to **tRowGenerator** using an **OnComponentOk** connection.
3. Link **tRowGenerator** to **tMysqlOutput** using a **Row** > **Main** connection.
4. Link **tRowGenerator** to the second **tMysqlRow** using an **OnSubjobOk** connection.



Configuring the components

1. Select the **tMysqlRow** to fill in the DB **Basic settings**.
2. Define the DB connection details in the corresponding fields.
3. Click the **[...]** button next to **Edit schema** and define the schema columns.
4. Propagate the properties and schema details onto the other components of the Job.
5. Type in the following SQL statement to alter the database entries: drop index <index_name> on <table_name>
6. Select the second **tMysqlRow** component, check the DB properties and schema.
7. Type in the SQL statement to recreate an index on the table using the following statement: create index <index_name> on <table_name> (<column_name>)

   The **tRowGenerator** component is used to generate automatically the columns to be added to the DB output table defined.
8. Select the **tMysqlOutput** component and fill in the DB connection properties. The table to be fed is named: *comprehensive*.
9. The schema should be automatically inherited from the data flow coming from the **tLogRow**. Edit the schema to check its structure and check that it corresponds to the schema expected on the DB table specified.

   The **Action on table** is *None* and the **Action on data** is *Insert*.

   No additional Columns is required for this job.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run the job.

   If you manage to watch the action on DB data, you can notice that the index is dropped at the start of the job and recreated at the end of the insert action.

   Related topics: section *tDBSQLRow properties*.

Scenario 2: Using PreparedStatement objects to query data

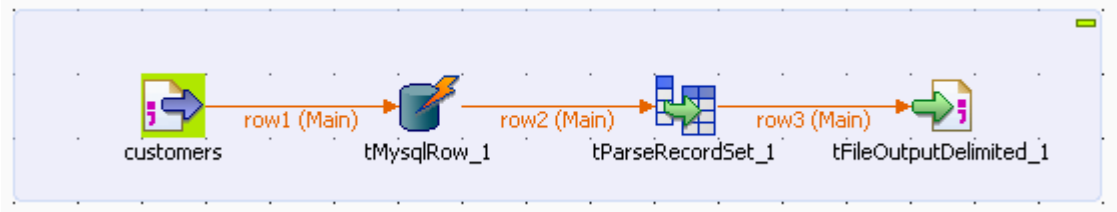
This scenario describes a four component job which allows you to link a table column with a client file. The MySQL table contains a list of all the American States along with the State ID, while the file contains the customer information including the ID of the State in which they live. We want to retrieve the name of the State for each client, using an SQL query. In order to process a large volume of data quickly, we use a PreparedStatement object which means that the query is executed only once rather than against each row in turn. Then each row is sent as a parameter. Note that PreparedStatement object can also be used in avoiding SQL injection.

For this scenario, we use a file and a database for which we have already stored the connection and properties in the Rerpository metadata.

*This scenario is for reference only. For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to drop all the required components from the **Palette** and complete the relevant settings manually in the specific Job.*

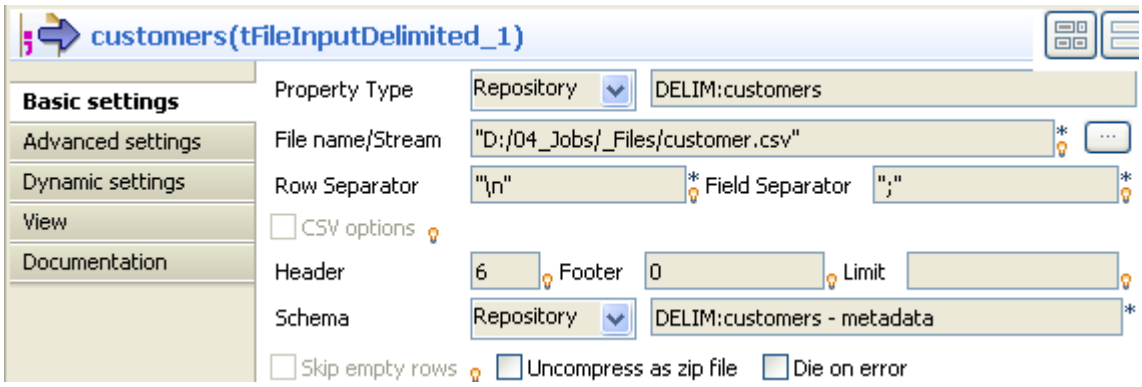
Linking the components

1. In the **Repository**, expand the **Metadata** and **File delimited** nodes.
2. Select the metadata which corresponds to the client file and slide the metadata onto the workspace. Here, we are using the *customers* metadata.
3. Double-click **tFileInputDelimited** in the **Components** dialog box to add **tFileInputDelimited** to the workspace, with the relevant fields filled by the metadata file.
4. Drop **tMysqlRow**, **tParseRecordSet** and **tFileOutputDelimited** onto the workspace.
5. Link **tFileInputDelimited** to **tMysqlRow** using a **Row** > **Main** connection.
6. Link **tMysqlRow** to **tParseRecordSet** using a **Row** > **Main** connection.
7. Link **tParseRecordSet** to **tFileOutputDelimited** using a **Row** > **Main** connection.

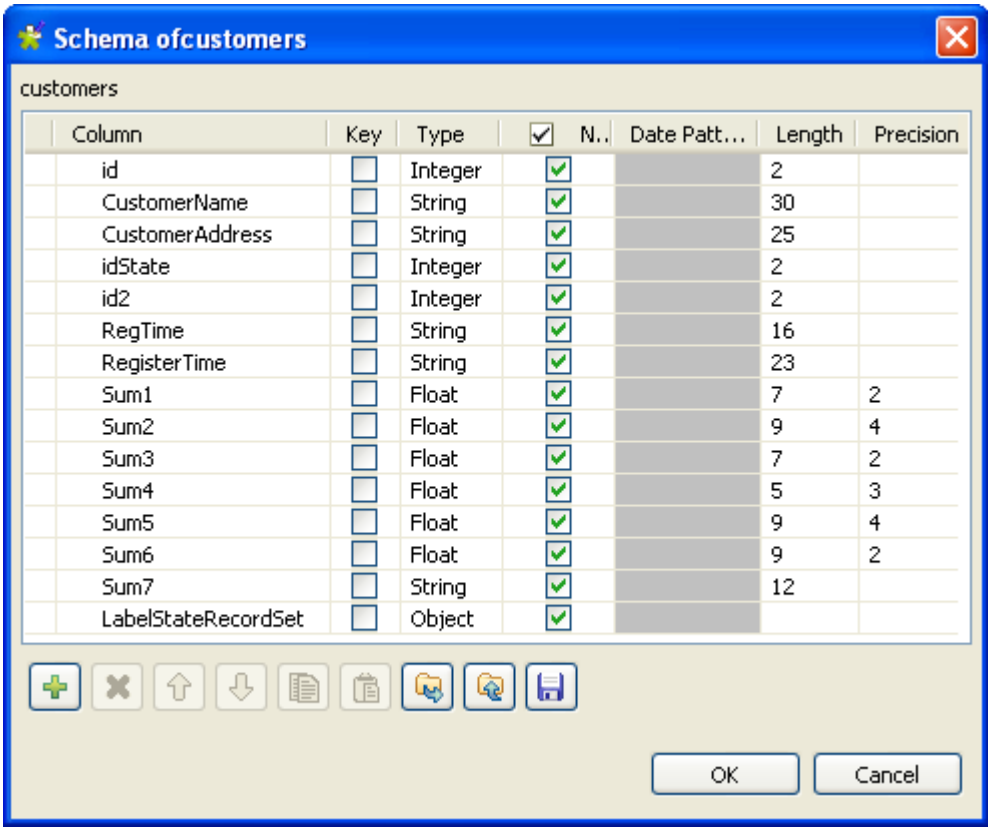


Configuring the components

1. Double-click **tFileInputDelimited** to open its **Basic settings** view.



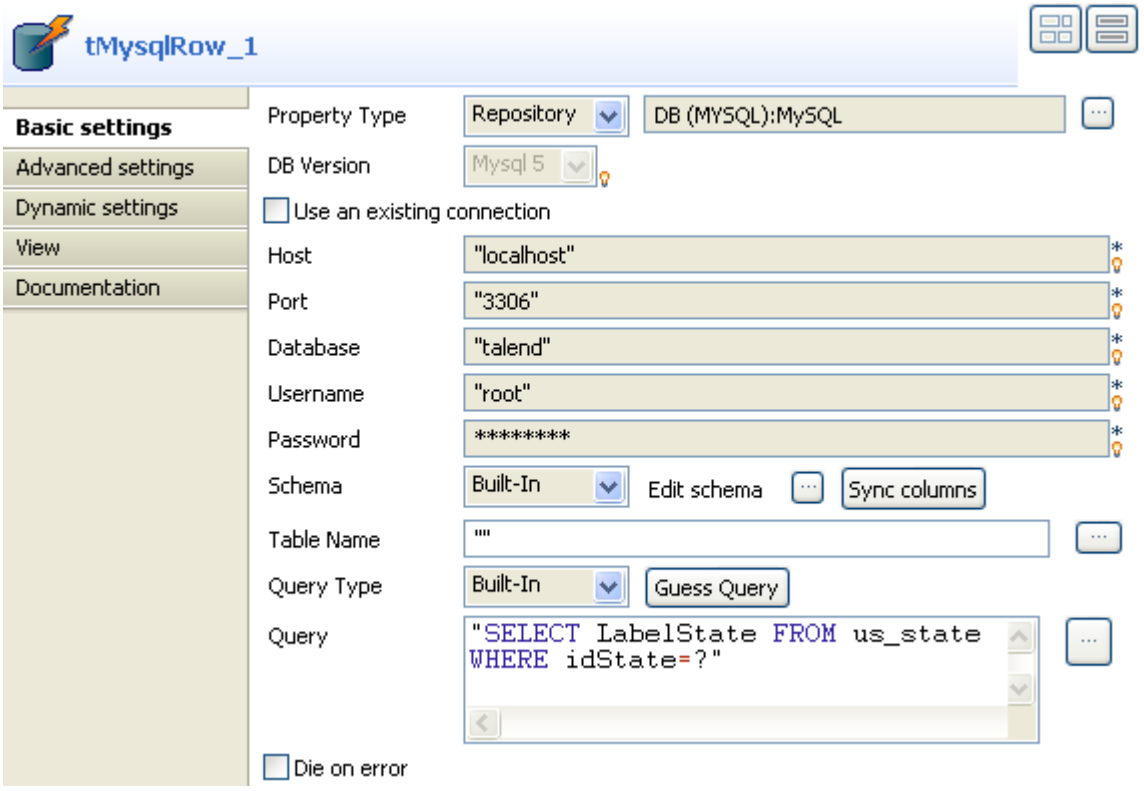
2. In the **Schema** list, select **Built-in** so that you can modify the component's schema. Then click on **[...]** next to the **Edit schema** field to add a column into which the name of the State will be inserted.



3. Click on the **[+]** button to add a column to the schema. Rename this column *LabelStateRecordSet* and select **Object** from the **Type** list. Click **OK** to save your modifications.

   From the **Palette**, select the **tMysqlRow**, **tParseRecordSet** and **tFileOutputDelimited** components and drop them onto the workspace.

4. Double click **tMysqlRow** to set its properties in the **Basic settings** tab of the **Component** view.



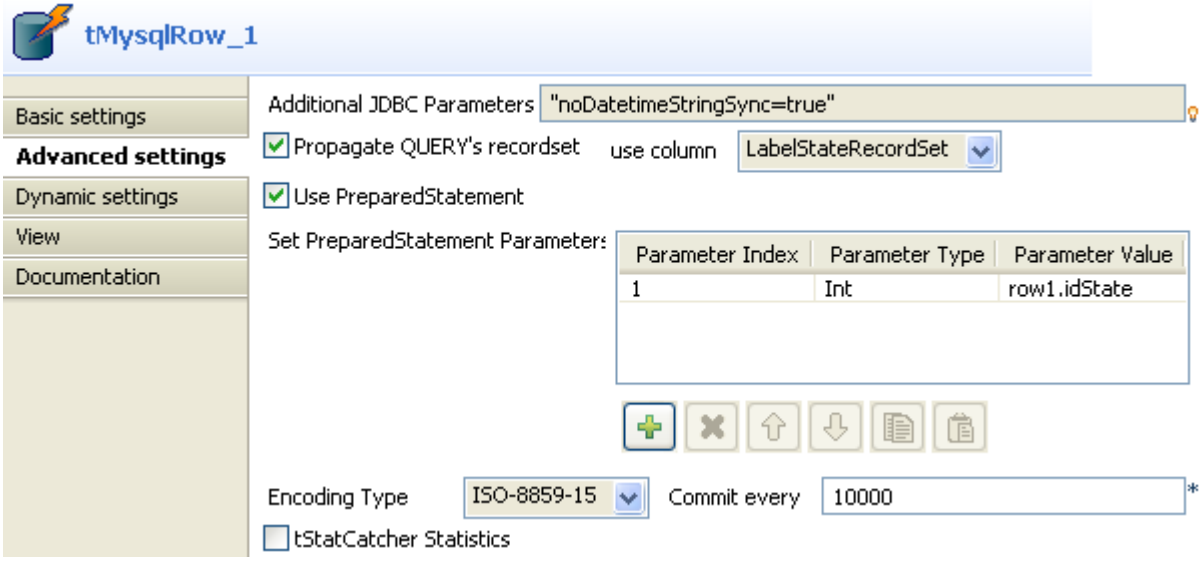
5. In the **Property Type** list, select **Repository** and click on the **[...]** button to select a database connection from the metadata in the Repository. The **DB Version**, **Host**, **Port**, **Database**, **Username** and **Password** fields are completed automatically. If you are using the **Built-in** mode, complete these fields manually.

6. From the **Schema** list, select **Built-in** to set the schema properties manually and add the *LabelStateRecordSet* column, or click directly on the **Sync columns** button to retrieve the schemma from the preceding component.

7. In the **Query** field, enter the SQL query you want to use. Here, we want to retrieve the names of the American States from the *LabelState* column of the MySQL table, *us_state*: `"SELECT LabelState FROM us_state WHERE idState=?"`.

   The question mark, “?”, represents the parameter to be set in the **Advanced settings** tab.

8. Click **Advanced settings** to set the components advanced properties.



9. Select the **Propagate QUERY's recordset** check box and select the *LabelStateRecordSet* column from the **use column** list to insert the query results in that column.

   Select the **Use PreparedStatement** check box and define the parameter used in the query in the **Set PreparedStatement Parameters** table.

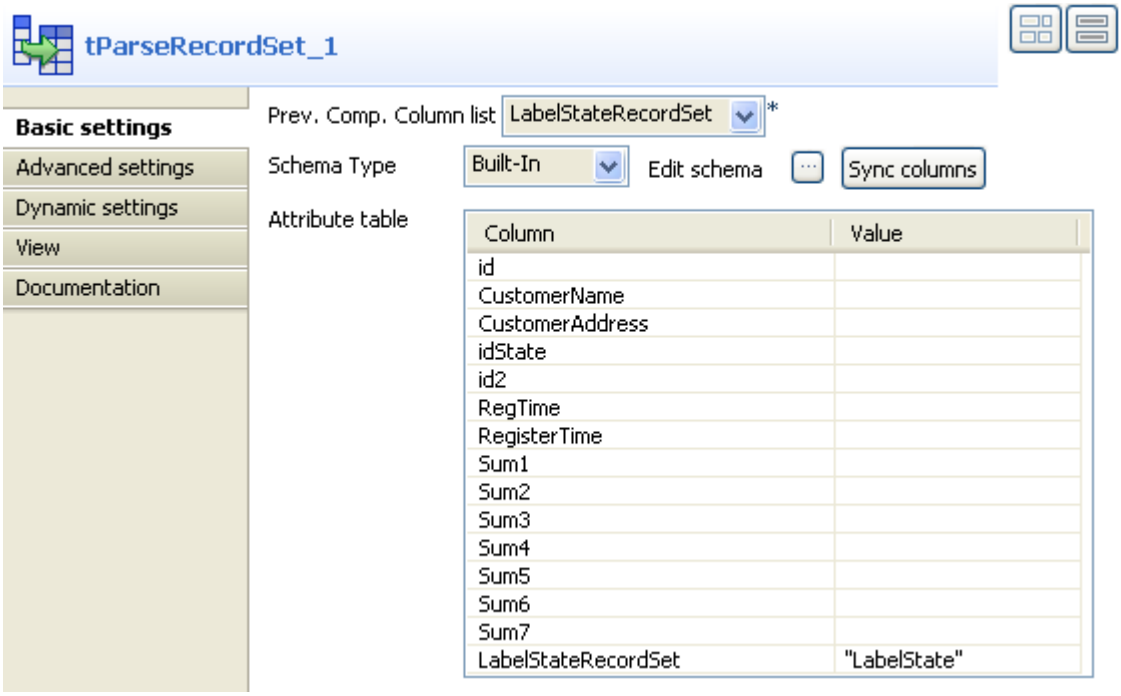
   Click on the **[+]** button to add a parameter.

   In the **Parameter Index** cell, enter the parameter position in the SQL instruction. Enter "*1*" as we are only using one parameter in this example.

   In the **Parameter Type** cell, enter the type of parameter. Here, the parameter is a whole number, hence, select **Int** from the list.

   In the **Parameter Value** cell, enter the parameter value. Here, we want to retrieve the name of the State based on the State ID for every client in the input file. Hence, enter "`row1.idState`".

10. Double click **tParseRecordSet** to set its properties in the **Basic settings** tab of the **Component** view.

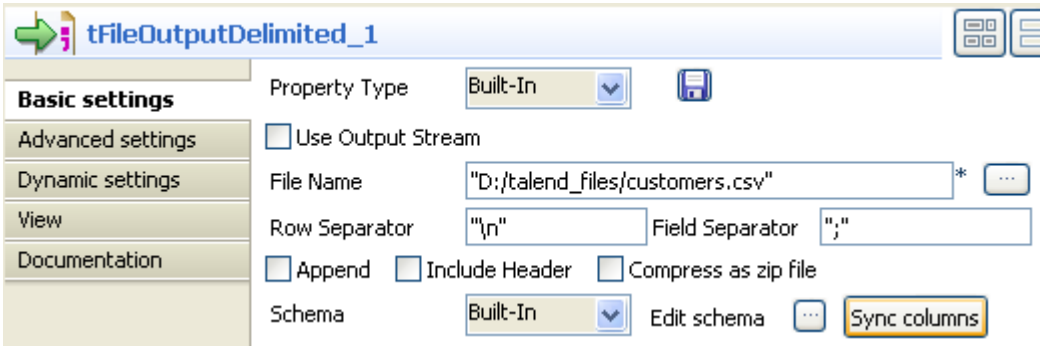


11. From the **Prev. Comp. Column list**, select the preceding components column for analysis. In this example, select *LabelStateRecordSet*.

    Click on the **Sync columns** button to retrieve the schema from the preceding component. The **Attribute table** is automatically completed with the schema columns.

    In the **Attribute table**, in the **Value** field which corresponds to the *LabelStateRecordSet*, enter the name of the column containing the State names to be retrieved and matched with each client, within double quotation marks. In this example, enter "*LabelState*".

12. Double click **tFileOutputDelimited** to set its properties in the **Basic settings** tab of the **Component** view.



13. In the **File Name field**, enter the access path and name of the output file.

    Click **Sync columns** to retrieve the schema from the preceding component.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run it.

customers.csv

	A	B	C	D	E	F	G	H	O
1	1	Griffith Paving	talend@apres	7	41	03/11/2006 09:20	2001-01-17 06:	67852.0	Connecticut
2	2	Bill's Dive Sh	511 Maple Av	35	5	19/11/2004 15:48	2002-06-07 09:	88792.0	Ohio
3	3	Childress Chi	662 Lyons Ci	1	28	16/02/2005 08:27	1990-04-01 21:	35340.0	Alabama
4	4	Facelift Kitch	220 Vine Ave	41	15	22/08/2002 09:55	1972-04-23 18:	6097.0	South Dakota
5	5	Terrinni & So	770 Exmoor F	5	9	28/06/2001 09:15	1982-04-19 10:	5146.0	California
6	6	Kermit the Pe	1860 Parksid	28	15	17/08/2003 10:07	2006-05-27 17:	16087.0	Nevada
7	7	Tub's Furnitur	807 Old Trail	15	9	27/08/2000 03:13	1970-03-27 23:	53216.0	Iowa
8	8	Toggle & Mye	618 Sheriden	9	15	24/03/2006 23:07	2005-08-02 01:	74168.0	Florida
9	9	Childress Chi	788 Tennyson	12	33	10/09/2001 06:33	1994-05-03 11:	92176.0	Idaho
10	10	Elle Hypnosis	2032 Northbr	1	7	11/01/1977 03:07	1975-06-10 20:	48498.0	Alabama

A column containing the name of the American State corrresponding to each client is added to the file.

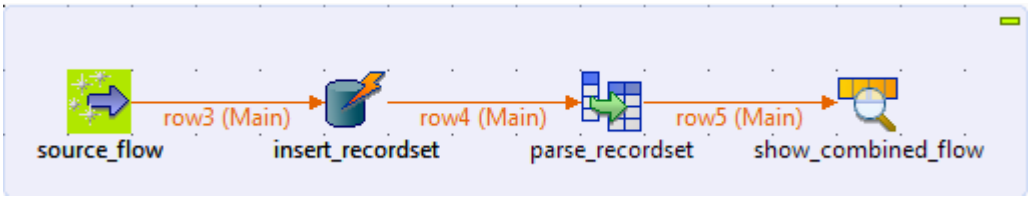
Scenario 3: Combining two flows for selective output

In this scenario, a flow generated by **tFixedFlowInput** is combined with a flow from the Mysql database. The source flow contains *id* and *age* fields while the Mysql table contains *id* and *name*. We want to retrieve the *age* data of the source flow and combine it with the *id* and *name* records from the Mysql table based on *id* matching. One thing that is worth noting is that the input schema is different from the output one at **tMysqlRow** in the Job.

Linking the components

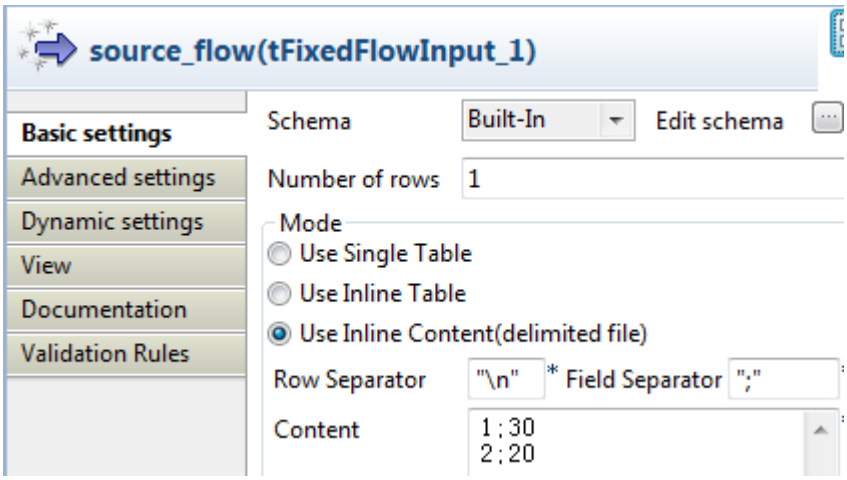
1. Drop **tFixedFlowInput**, **tMysqlRow**, **tParseRecordSet** and **tLogRow** onto the workspace.
2. Rename **tFixedFlowInput** as **source_flow**, **tMysqlRow** as **insert_recordset**, **tParseRecordSet** as **parse_recordset** and **tLogRow** as **show_combined_flow**.

3. Link **tFixedFlowInput** to **tMysqlRow** using a **Row** > **Main** connection.
4. Link **tMysqlRow** to **tParseRecordSet** using a **Row** > **Main** connection.
5. Link **tParseRecordSet** to **tLogRow** using a **Row** > **Main** connection.



Configuring the components

1. Double-click **tFixedFlowInput** to open its **Basic settings** view.

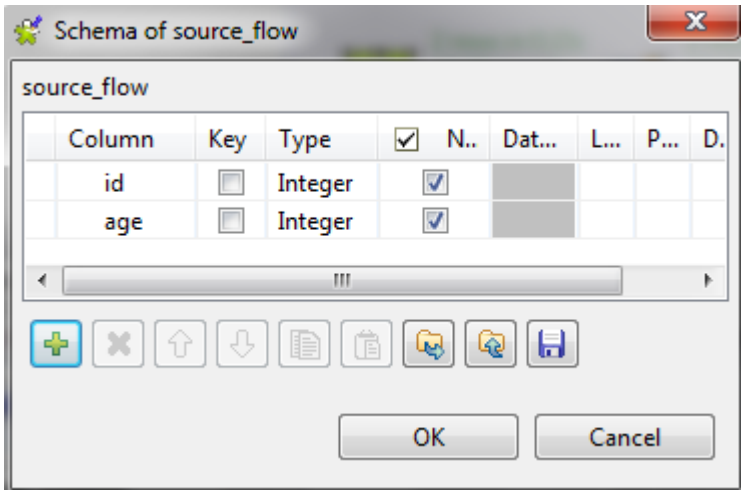


2. Select **Use Inline Content (delimited file)** in the **Mode** area.

   In the **Content** field, enter the data to be transferred:

```
1;30
2;20
```

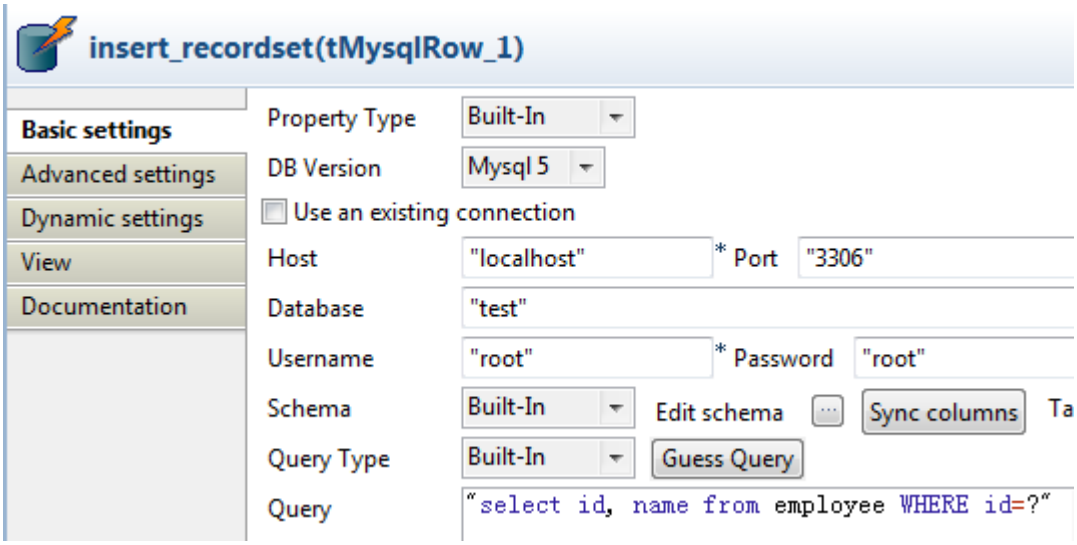
3. Double-click the **[...]** button next to **Edit schema** to open the schema editor.



   Click the **[+]** button to add two columns, namely *id* and *age*, with the type of *Integer*.

Click **Ok** to close the editor.

4. Double-click **tMysqlRow** to open its **Basic settings** view.



5. In the **Host** and **Port** fields, enter the connection details.

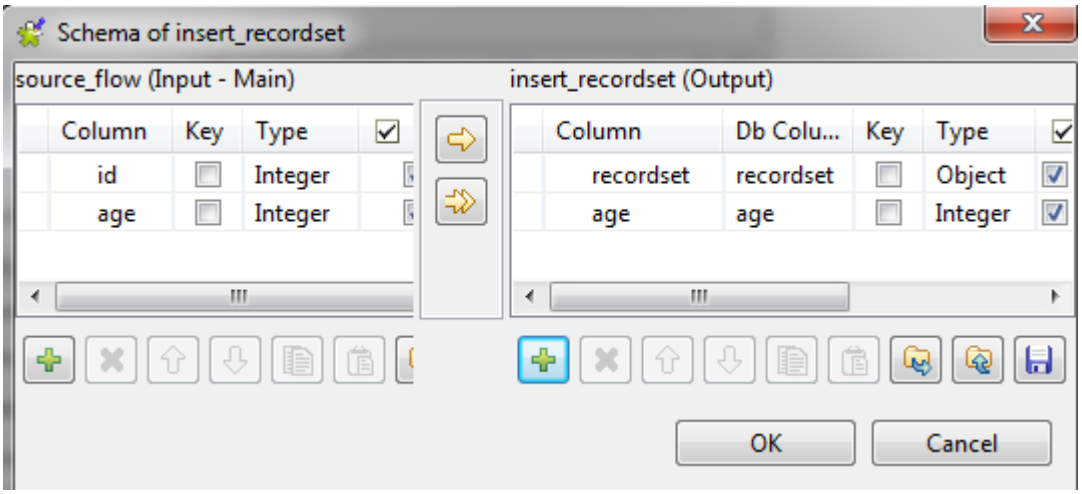
   In the **Database** field, enter the database name.

   In the **Username** and **Password** fields, enter the authentication details.

   In the **Query** field, enter the SQL query to retrieve the *id* and *name* data from the Mysql table *employee*: `"select id, name from employee WHERE id=?"`.

   The question mark, "?", represents the parameter to be set in the **Advanced settings** tab.

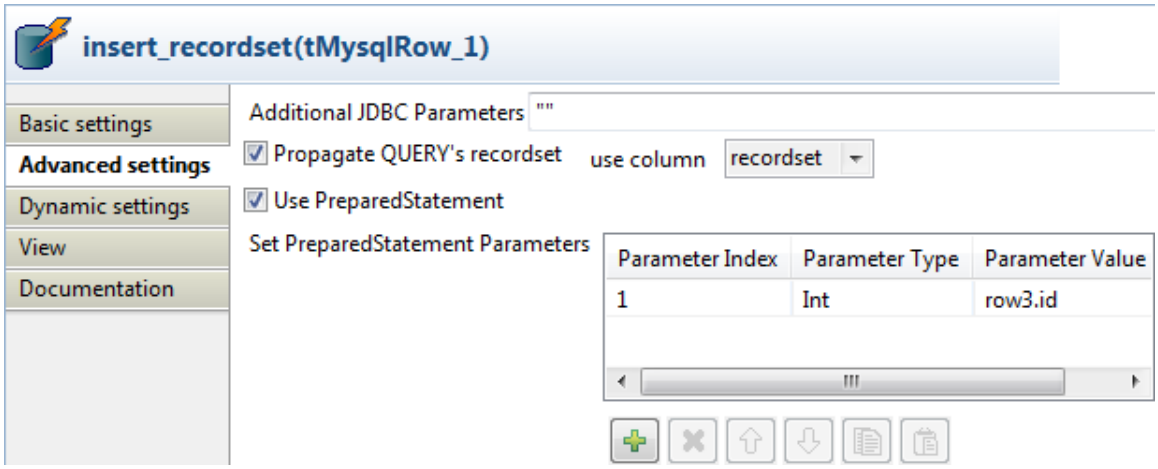
6. Click the **[...]** button next to **Edit schema** to open the schema editor.



7. Click the **[+]** button to add two columns in the right part, namely *recordset* and *age*, with the type of *Object* and *Integer*. Note that *recordset* is intended to hold the query results of the Mysql table, namely the *id* and *name* fields.

   Click **OK** to close the editor.

8. Click the **Advanced settings** tab for further setup.



9. Select the **Propagate QUERY's recordset** check box and choose *recordset* from the **use column** list to insert the query results in that column.

   Select the **Use PreparedStatement** check box and define the parameter used in the query in the **Set PreparedStatement Parameters** table.

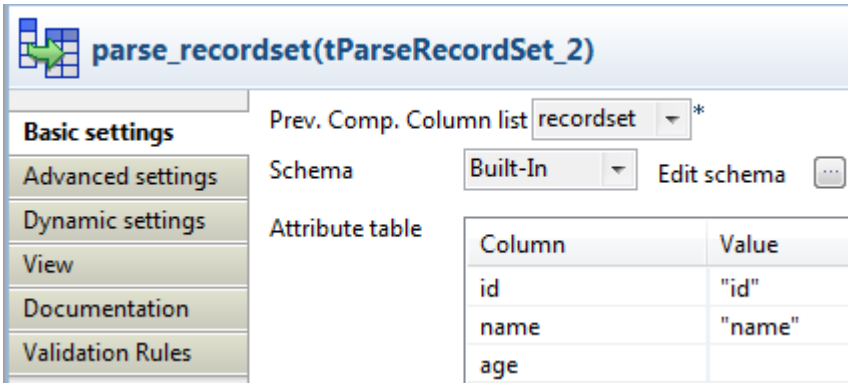
10. Click on the **[+]** button to add a line.

    In the **Parameter Index** cell, enter the parameter position in the SQL instruction. Enter "*1*" as we are only using one parameter in this example.

    In the **Parameter Type** cell, enter the type of parameter. Here, the parameter is an integer. Hence, select **Int** from the list.

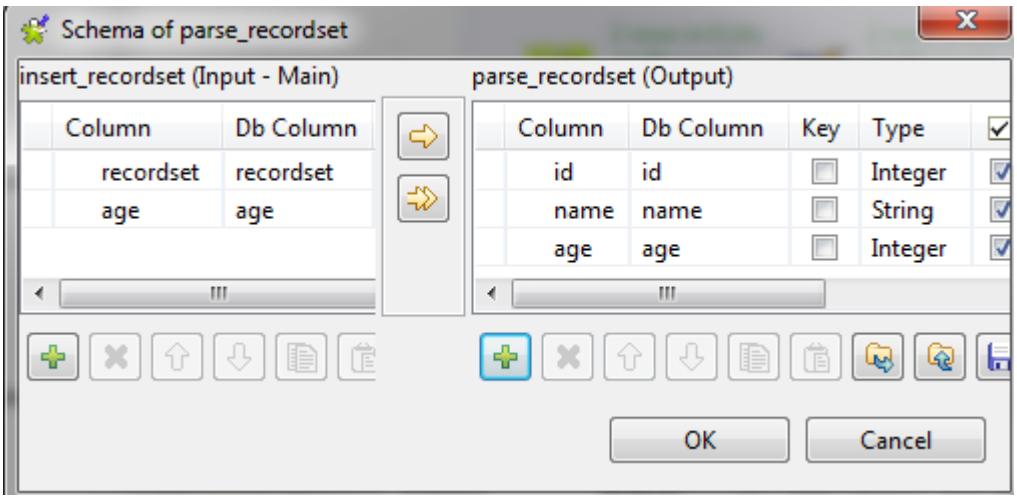
    In the **Parameter Value** cell, enter the parameter value. Here, we want to retrieve the *id* and *name* from the *employee* table based on the *id* value from the source flow. Hence, enter *row3.id*.

11. Double-click **tParseRecordSet** to open its **Basic settings** view.



    From the **Prev. Comp. Column list**, select the column to parse, namely *recordset*.

12. Click the **[...]** button next to **Edit schema** to open the schema editor.

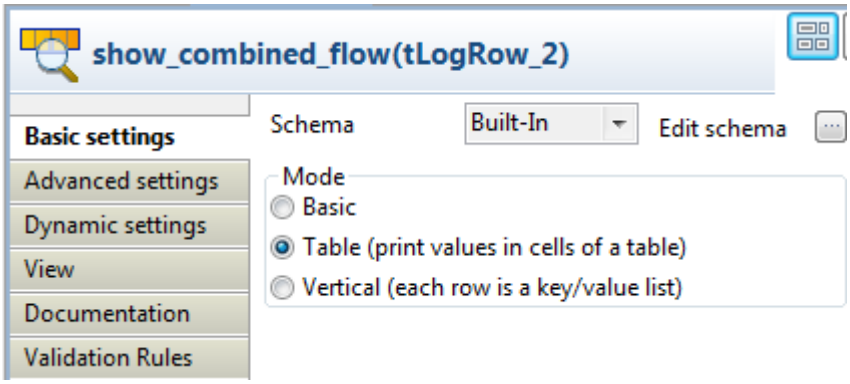


Click the **[+]** button to add three columns in the right part, namely *id*, *name* and *age*, with the type of *Integer*, *String* and *Integer*. Note that the *id* and *name* fields are intended to hold the parsed data of *recordset*.

Click **OK** to close the editor.

In the **Attribute table**, in the **Value** fields which correspond to *id* and *name*, enter the name of the column in the Mysql table to be retrieved, namely *"id"* and *"name"*.

13. Double-click **tLogRow** to open its **Basic settings** view.



In the **Mode** area, select **Table (print values in cells of a table** for better display.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run the Job.

```
[statistics] connecting to socket on port 4030
[statistics] connected
.--+--------+---.
|show_combined_flow|
|=-+--------+--=|
|id|name    |age|
|=-+--------+--=|
|1 |Andy    |30 |
|2 |Anderson|20 |
'--+--------+---'

[statistics] disconnected
```

tMysqlSCD

tMysqlSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tMysqlSCD*.

tMysqlSCDELT

tMysqlSCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tMysqlSCDELT*.

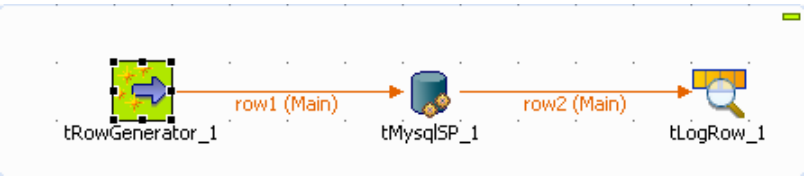
tMysqlSP

tMysqlSP Properties

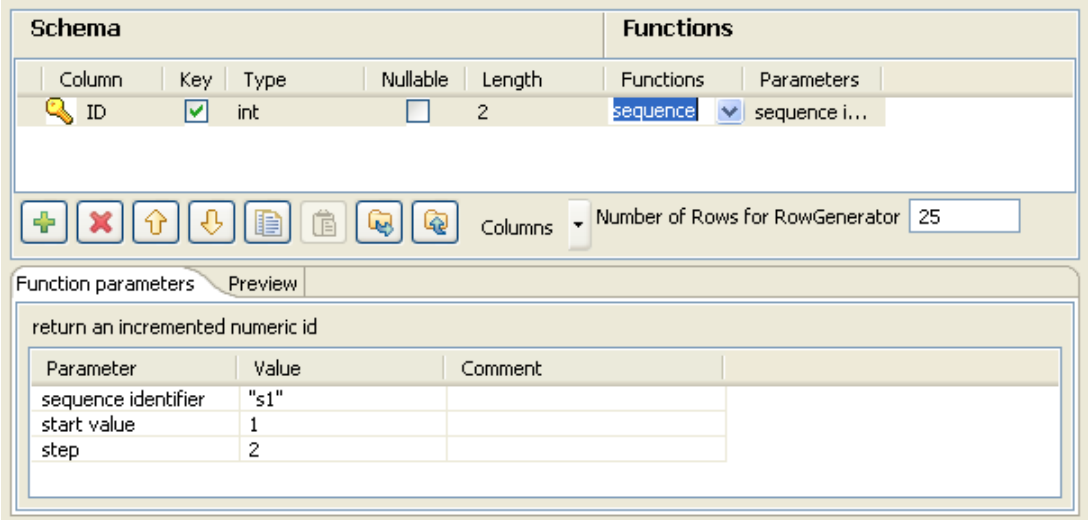
Component family	Databases/Mysql	
Function	**tMysqlSP** calls the database stored procedure.	
Purpose	**tMysqlSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Select this check box, if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the **tPostgresqlCommit** component if you want to analyze a set of records from a database table or DB query and return single records.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Scenario: Finding a State Label using a stored procedure

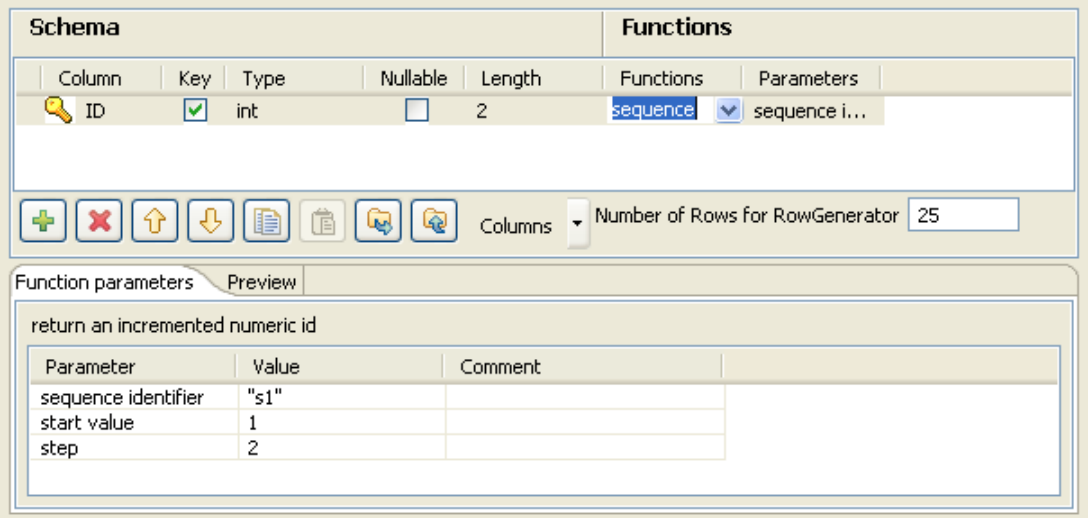
The following job aims at finding the State labels matching the odd State IDs in a Mysql two-column table. A stored procedure is used to carry out this operation.



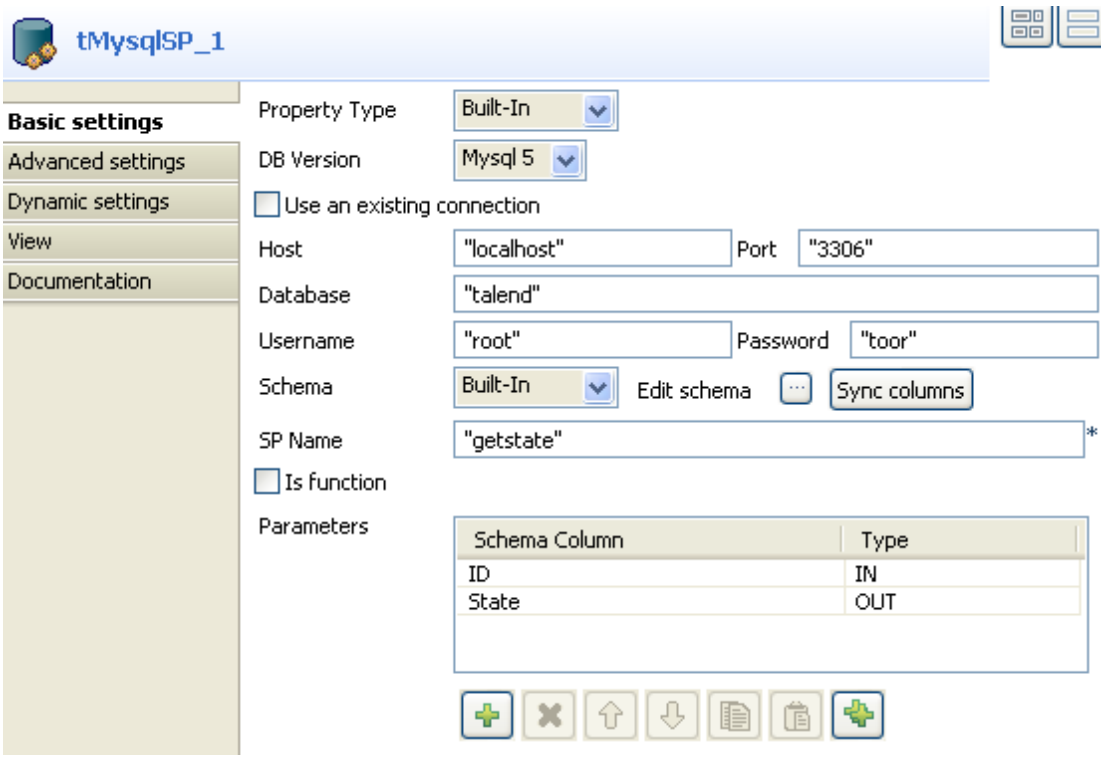
- Drag and drop the following components used in this example: **tRowGenerator**, **tMysqlSP**, **tLogRow**.
- Connect the components using the **Row Main** link.
- The **tRowGenerator** is used to generate the odd id number. Double-click on the component to launch the editor.



- Click on the **Plus** button to add a column to the schema to generate.
- Select the **Key** check box and define the **Type** to **Int**.
- The **Length** equals to 2 digits max.
- Use the preset function called **sequence** but customize the Parameters in the lower part of the window.



- Change the **Value** of **step** from *1* to *2* for this example, still starting from 1.
- Set the **Number of generated rows** to 25 in order for all the odd State id (of 50 states) to be generated.
- Click **OK** to validate the configuration.
- Then select the **tMysqlSP** component and define its properties.

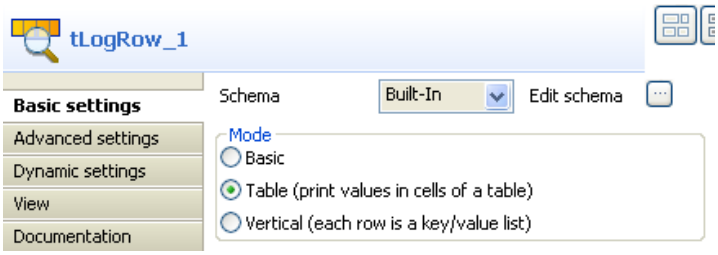


- Define the DB connection details in the corresponging fields.
- Click **Sync Column** to retrieve the generated schema from the preceding component.
- Then click **Edit Schema** and add an extra column to hold the State Label to be output, in addition to the ID.
- Type in the name of the procedure in the **SP Name** field as it is called in the Database. In this example, *getstate*. The procedure to be executed states as follows:

```
DROP PROCEDURE
```

```
IF EXISTS `talend`.`getstate` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `getstate`(IN pid INT, OUT
pstate VARCHAR(50))
BEGIN
SELECT LabelState INTO pstate FROM us_states WHERE idState = pid;
END $$
```

- In the **Parameters** area, click the plus button to add a line to the table.
- Set the **Column** field to *ID*, and the **Type** field to *IN* as it will be given as input parameter to the procedure.
- Add a second line and set the **Column** field to *State* and the **Type** to *Out* as this is the output parameter to be returned.
- Eventually, set the **tLogRow** component properties.



- Synchronize the schema with the preceding component.
- And select the **Print values in cells of a table** check box for reading convenience.
- Then save your Job and execute it.

```
Starting job MysqlSP at 17:24 23/08/2007.
.--+--------------.
|     tLogRow_1    |
|=-+--------------=|
|ID|State          |
|=-+--------------=|
|1 |Alabama        |
|3 |Arizona        |
|5 |California     |
|7 |Connecticut    |
|9 |Florida        |
|11|Hawaii         |
|13|Illinois       |
|15|Iowa           |
|17|Kentucky       |
|19|Maine          |
|21|Massachusetts  |
|23|Minnesota      |
|25|Missouri       |
|27|Nebraska       |
|29|New Hampshire  |
|31|New Mexico     |
|33|North Carolina |
```

The output shows the state labels corresponding to the odd state ids as defined in the procedure.

Check the **tPostgresqlCommit** component if you want to analyze a set of records from a database table or DB query and return single records.

tMysqlTableList

tMysqlTableList Properties

Component family	Databases/MySQL	
Function	Iterates on a set of table names through a defined Mysql connection.	
Purpose	Lists the names of a given set of Mysql tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection**.	
Limitation	n/a	

Related scenario

For **tMysqlTableList** related scenario, see section *Scenario: Iterating on a DB table and listing its column names*.

tOracleBulkExec

tOracleBulkExec properties

The **tOracleOutputBulk** and **tOracleBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tOracleOutputBulkExec** component, detailed in a separate section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

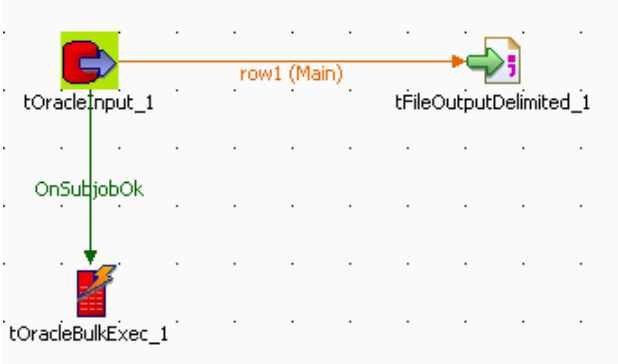
Component family	Databases/Oracle	
Function	**tOracleBulkExec** inserts, appends, replaces or truncate data in an Oracle database.	
Purpose	As a dedicated component, it allows gains in performance during operations performed on data of an Oracle database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when you are using the component **tOracleConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers
	DB Version	Select the Oracle version in use
	Host	IP address of the database server
	Port	Port number listening the database server
	Database	Database name.
	Schema	Schema name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created.

		Create a table if doesn't exist: The table is created if it does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Data file name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Action on data	On the data of the table defined, you can perform: **Insert**: Inserts rows to an empty table. If duplicates are found, Job stops. **Update**: Update the existing data of the table. **Append**: Adds rows to the existing data of the table **Replace**: Overwrites some rows of the table **Truncate**: Drops table entries and inserts new input flow data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
Advanced settings	*Advanced separator (for number)*	Select this check box to change the separator used for the numbers.
	Use existing control file	Select this check box if you use a control file (.ctl) and specify its path in the **.ctl file name** field.
	Record format	Define the record format: **Default**: format parameters are set by default. **Stream**: set **Record terminator**. **Fixed**: set the **Record length**. **Variable**: set the **Field size of the record length**.
	Specify .ctl file's INTO TABLE clause manually	Select this check box to manually fill in the **INTO TABLE clause** of the control file.
	Fields terminated by	Character, string or regular expression to separate fields: **None**: no separator is used. **Whitespace**: the separator used is a space. **EOF (used for loading LOBs from lobfile)**: the separator used is an EOF character (End Of File). **Other terminator**: Set another terminator in the **Field terminator** field.
	Use fields enclosure	Select this check box if you want to use enclosing characters for the text: **Fields enclosure (left part)**: character delimiting the left of the field. **Field enclosure (right part)**: character delimiting the right of the field.
	Use schema's Date Pattern to load Date field	Select this check box to use the date pattern of the schema in the date field.
	Specify field condition	Select this check box to define data loading condition.
	Preserve blanks	Select this check box to preserve the blanks.
	Trailing null columns	Select this check box to load null columns.
	Load options	Click + to add data loading options:

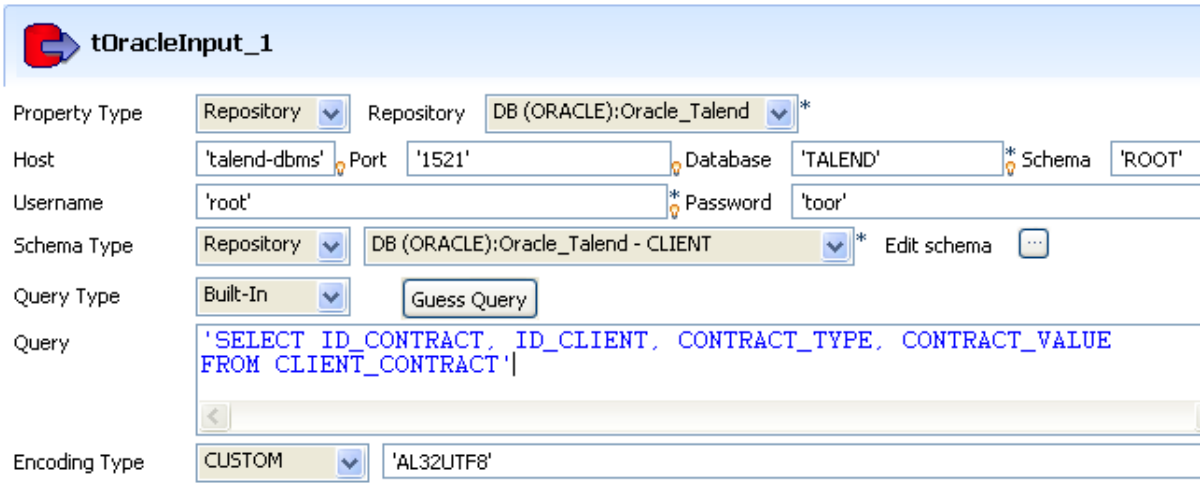
		Parameter: select a loading parameter from the list. **Value**: enter a value for the parameter selected.
	NLS Language	In the list, select the language used for the data that are not used in Unicode.
	Set Parameter NLS_TERRITORY	Select this check box to modify the territory conventions used for day and weeks numbering. Your OS value is the default value used.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for database data handling.
	Output	Select the type of output for the standard output of the Oracle database: **to console**, **to global variable**.
	Convert columns and table names to uppercase	Select this check box to uppercase the names of the columns and the name of the table.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This dedicated component offers performance and flexibility of Oracle DB query handling.	

Scenario: Truncating and inserting file data into Oracle DB

This scenario describes how to truncate the content of an Oracle DB and load an input file content. The related job is composed of three components that respectively creates the content, output this content into a file to be loaded onto the Oracle database after the DB table has been truncated.

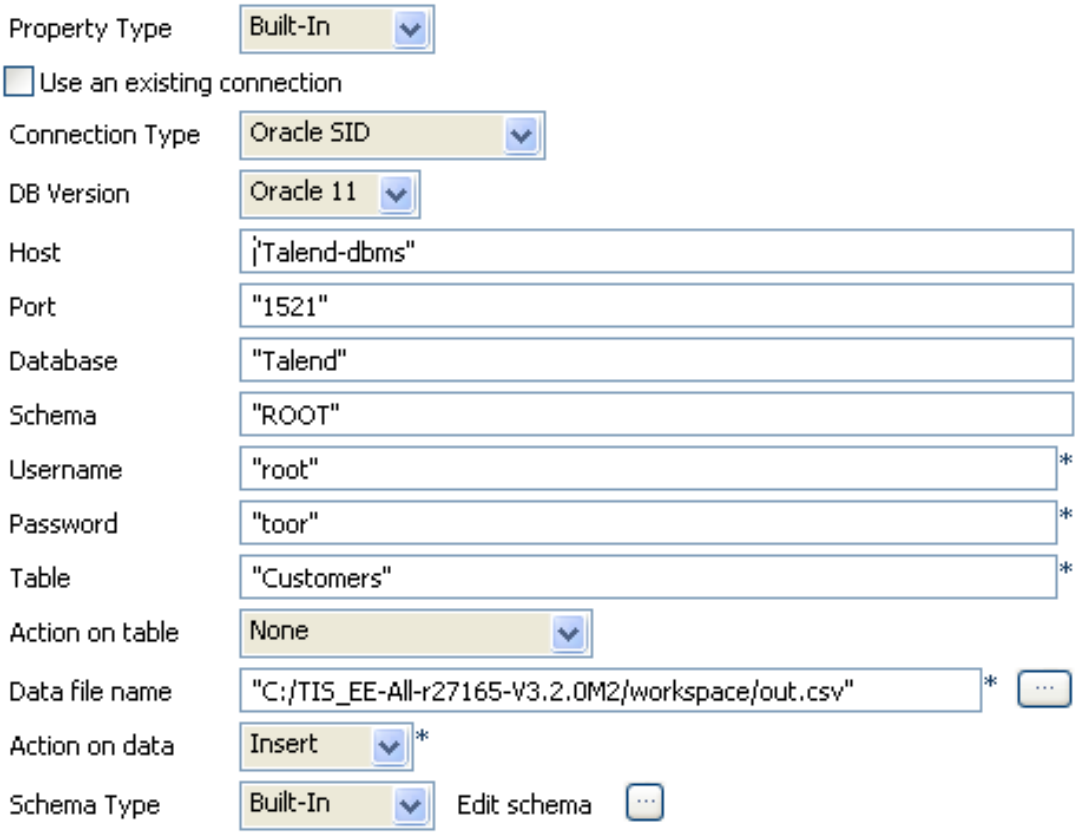


- Drop the following components: **tOracleInput**, **tFileOutputDelimited**, **tOracleBulkExec** from the **Palette** to the design workspace
- Connect the **tOracleInput** with the **tFileOutputDelimited** using a **row main** link.
- And connect the **tOracleInput** to the **tOracleBulkExec** using a **OnSubjobOk** trigger link.
- Define the Oracle connection details. We recommend you to store the DB connection details in the Metadata repository in order to retrieve them easily at any time in any job.



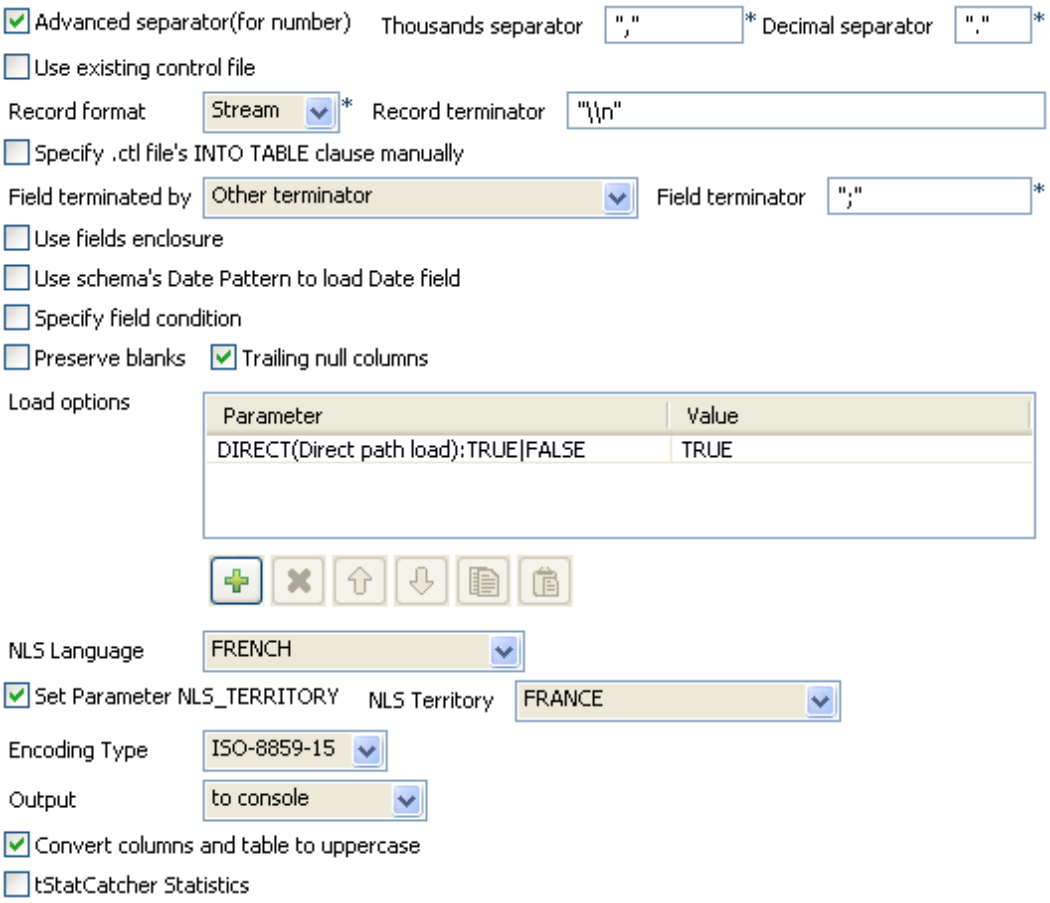
*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

- Define the schema, if it isn't stored either in the **Repository**. In this example, the schema is as follows: *ID_Contract, ID_Client, Contract_type, Contract_Value*.
- Define the **tFileOutputDelimited** component parameters, including output **File Name**, **Row separator** and **Fields delimiter**.
- Then double-click on the **tOracleBulkExec** to define the DB feeding properties.



- In the **Property Type**, select **Repository** mode if you stored the database connection details under the **Metadata** node of the **Repository** or select **Built-in** mode to define them manually. In this scenario, we use the **Built-in** mode.

- Thus, set the connection parameters in the following fields: **Host**, **Port**, **Database**, **Schema**, **Username**, and **Password**.
- Fill in the name of the **Table** to be fed and the **Action on data** to be carried out, in this use case: **insert**.
- In the **Schema** field, select **Built-in** mode, and click **[...]** button next to the **Edit schema** field to describe the structure of the data to be passed on to the next component.
- Click the **Advanced settings** view to configure the advanced settings of the component.



- Select the Use an existing control file check box if you want to use a control file (.ctl) storing the status of the physical structure of the database. Or, fill in the following fields manually: **Record format**, **Specify .ctl file's INTO TABLE clause manually**, **Field terminated by**, **Use field enclosure**, **Use schema's Date Pattern to load Date field**, **Specify field condition**, **Preserve blanks**, **Trailing null columns**, **Load options**, **NLS Language** et **Set Parameter NLS_TERRITORY** according to your database.
- Define the **Encoding** as in preceding steps.
- For this scenario, in the **Output** field, select **to console** to output the standard output f the database to the console.

Press **F6** to run the job. The log output displays in the **Run** tab and the table is fed with the parameter file data.

Related topic: see section *Scenario: Inserting data in MySQL database*.

tOracleClose

tOracleClose properties

Function	**tOracleClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tOracleCommit

tOracleCommit Properties

This component is closely related to **tOracleConnection** and **tOracleRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Oracle	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tOracleCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleConnection** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleCommit** related scenario, see section *tMysqlConnection*

tOracleConnection

tOracleConnection Properties

This component is closely related to **tOracleCommit** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

Component family	Databases/Oracle	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Connection type	Drop-down list of available drivers: **Oracle OCI**: Select this connection type to use Oracle Call Interface with a set of C-language software APIs that provide an interface to the Oracle database. **Oracle RAC**: Select this connection type to access a clustered database. **Oracle Service Name**: Select this connection type to use the TNS alias that you give when you connect to the remote database. **WALLET**: Select this connection type to store credentials in an Oracle wallet. **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Select the Oracle version in use.
	Use tns file	Select this check box to use the metadata of a context included in a tns file. One tns file may have many contexts. **TNS File**: Enter the path to the tns file manually or browse to the file by clicking the three-dot button next to the filed. **Select a DB Connection in Tns File**: Click the three-dot button to display all the contexts held in the tns file and select the desired one.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating. You can set the encoding parameters through this field.

	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime.
Usage	This component is to be used along with Oracle components, especially with **tOracleCommit** and **tOracleRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleCommit** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleConnection** related scenario, see section *tMysqlConnection*

tOracleInput

tOracleInput properties

Component family	Databases/Oracle	
Function	**tOracleInput** reads a database and extracts fields based on a query.	
Purpose	**tOracleInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tOracleConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers: **Oracle OCI**: Select this connection type to use Oracle Call Interface with a set of C-language software APIs that provide an interface to the Oracle database. **Oracle RAC**: Select this connection type to access a clustered database. **Oracle Service Name**: Select this connection type to use the TNS alias that you give when you connect to the remote database. **WALLET**: Select this connection type to store credentials in an Oracle wallet. **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Select the Oracle version in use.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Oracle schema	Oracle schema name.

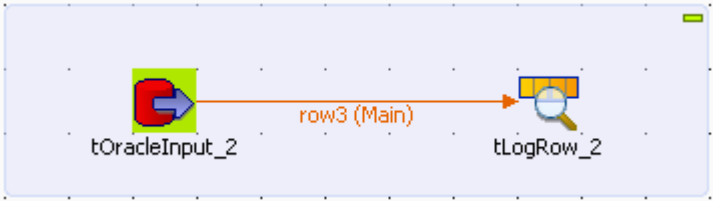
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Database table name.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	No null values	Check this box to improve the performance if there are no null values.
Usage	This component covers all possible SQL queries for Oracle databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario 1: Using context parameters when reading a table from an Oracle database

In this scenario, we will read a table from an Oracle database, using a context parameter to refer to the table name.

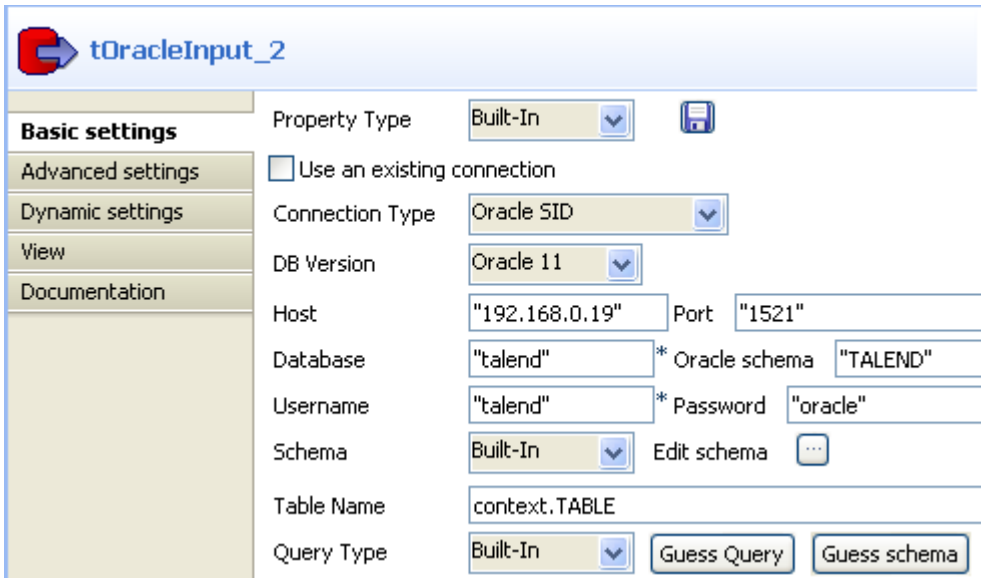
Dragging and dropping components and linking them together

1. Drop **tOracleInput** and **tLogRow** from the **Palette** onto the workspace.
2. Link **tOracleInput** to **tLogRow** using a **Row > Main** connection.



Configuring the components

1. Double-click **tOracleInput** to open its **Basic Settings** view in the **Component** tab.



2. In the **Host** field, enter the Oracle database serverse's IP address, *"192.168.0.19"* in this example.

   In the **Port** field, enter the port number, *"1521"* in this example.

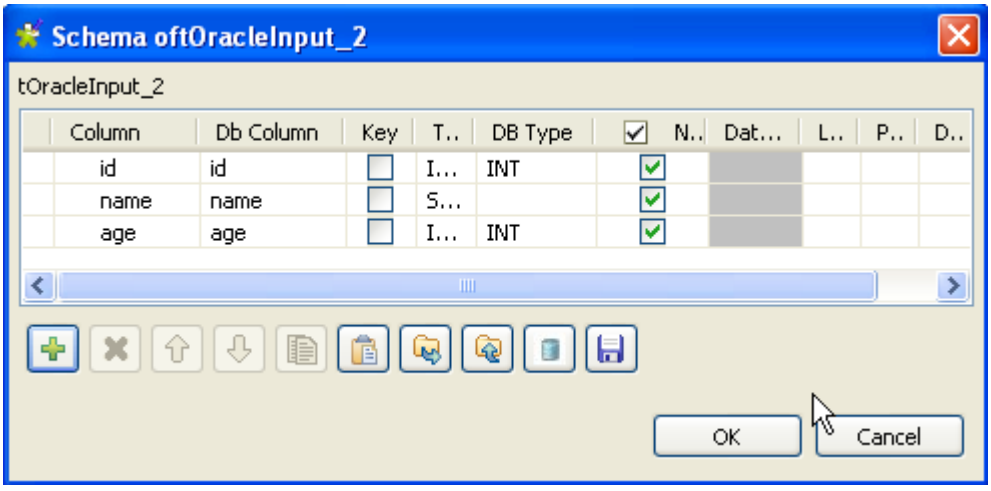
   In the **Database** field, enter the database name, *"talend"* in this example.

   In the **Oracle schema** field, enter the Oracle schema name, *"TALEND"* in this example.

   In the **Username** and **Password** fields, enter the authentication details, respectively *"talend"* and *"oracle"* in this example.

3. Set the **Schema** as **Built-In** and click **Edit schema** to define the desired schema.

   The schema editor opens:



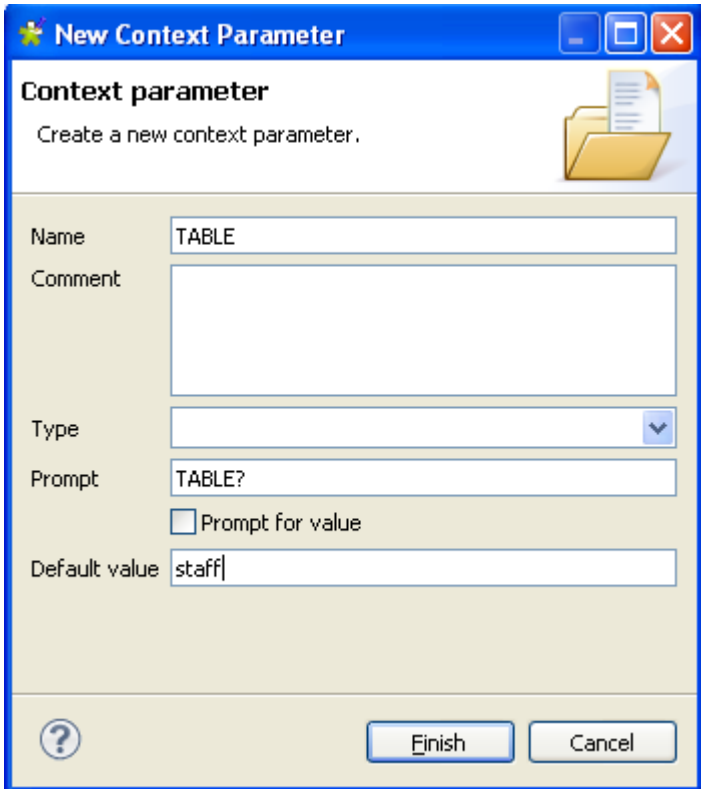
4. Click the **[+]** button to add the rows that you will use to define the schema, three columns in this example: *id*, *name* and *age*.

   Under **Column**, click the fields to enter the corresponding column names.

   Click the fields under **Type** to define the type of data.

Click **OK** to close the schema editor.

5. Put the cursor in the **Table Name** field and press **F5** for context parameter setting.



For more information about context settings, see *Talend Open Studio for Big Data User Guide*.

6. Keep the default setting in the **Name** field and type in the name of the database table in the **Default value** field, *staff* in this use case.

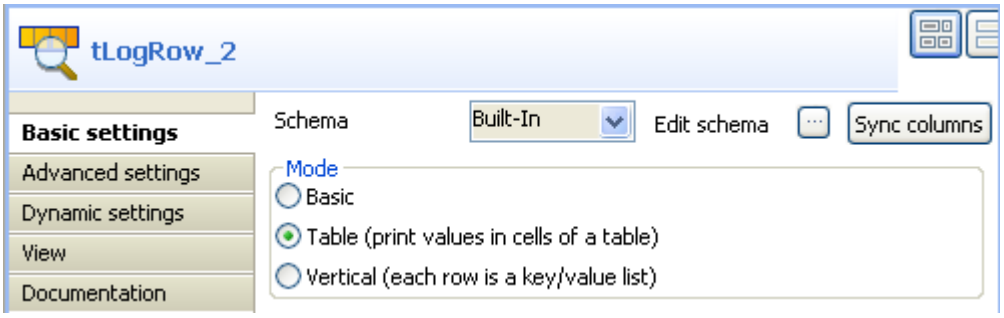
7. Click **Finish** to validate the setting.

The context parameter *context.TABLE* automatically appears in the **Table Name** field.

8. In the **Query type** list, select **Built-In**. Then, click **Guess Query** to get the query statement.

```
"SELECT
  TALEND."+context.TABLE+".id,
  TALEND."+context.TABLE+".name,
  TALEND."+context.TABLE+".age
FROM TALEND."+context.TABLE
```

9. Double-click **tLogRow** to set its **Basic Settings** in the **Component** tab.

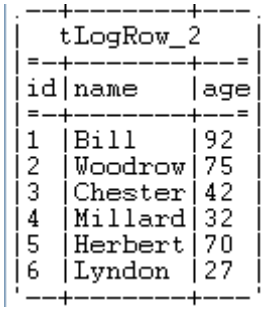


10. In the **Mode** area, select **Table (print values in cells of a table)** for a better display of the results.

11. Save the Job.

Executing the Job

The results below can be found after **F6** is pressed to run the Job.



Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tOracleOutput

tOracleOutput properties

Component family	Databases/Oracle	
Function	**tOracleOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tOracleOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a **tOracleConnection** component. When you deselect it, a check box appears (selected by default and followed by a field) in the **Advanced settings**, **Batch Size**, which enables you to define the number of lines in each processed batch. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers: **Oracle OCI**: Select this connection type to use Oracle Call Interface with a set of C-language software APIs that provide an interface to the Oracle database. **Oracle RAC**: Select this connection type to access a clustered database. **Oracle Service Name**: Select this connection type to use the TNS alias that you give when you connect to the remote database. **WALLET**: Select this connection type to store credentials in an Oracle wallet. **Oracle SID**: Select this connection type to uniquely identify a particular database on a system.
	DB Version	Select the Oracle version in use.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.

	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.

Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Use Hint Options	Select this check box to activate the hint configuration area which helps you optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */`. - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	Convert columns and table to uppercase	Select this check box to set the names of columns and table in upper case.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch. *This option is available only when you do not **Use an existing connection** in **Basic settings**.*
	Support null in "SQL WHERE" statement	Select this check box to validate null in "SQL WHERE" statement.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Oracle database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMysqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tOracleOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.

- section *Scenario 1: Adding a new column and altering data in a DB table*.

tOracleOutputBulk

tOracleOutputBulk properties

The **tOracleOutputBulk** and **tOracleBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tOracleOutputBulkExec** component, detailed in a separate section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

Component family	Databases/Oracle	
Function	Writes a file with columns based on the defined delimiter and the Oracle standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Oracle database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Advanced separator (for number)*	Select this check box to change data separators for numbers: **Thousands separator**: define separators you want to use for thousands. **Decimal separator**: define separators you want to use for decimals.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to separate rows.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Bulk file parameters	Set the parameters **Buffer Size** and **StringBuilder Size** for a performance gain according to the memory size.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is to be used along with **tOracleBulkExec** component. Used together they offer gains in performance while feeding a Oracle database.	

Related scenarios

For use cases in relation with **tOracleOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.

- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tOracleOutputBulkExec

tOracleOutputBulkExec properties

The **tOracleOutputBulk** and **tOracleBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tOracleOutputBulkExec** component.

Component family	Databases/Oracle	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to an Oracle database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tOracleConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	List of available drivers
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operations is carried out. **Drop and create the table**: The table is removed and created again.

		Create a table: The table does not exist and gets created. **Create table if doesn't exist**: The table is created if does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if required.
	Append	Select this check box to add the new rows at the end of the file.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Truncate**: Remove all entries from table.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
	Field separator	Character, string or regular expression to separate fields.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Advanced settings	*Advanced separator (for number)*	Select this check box to change data separators for numbers: **Thousands separator**: define separators you want to use for thousands. **Decimal separator**: define separators you want to use for decimals.
	Use existing control file	Select this check box and browse to the .ctl control file you want to use.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: "\n"on Unix) to separate rows.
	Specify .ctl file's INTO TABLE clause manually	Select this check box to enter manually the INTO TABLE clause of the control file directly into the code.
	Use schema's Date Pattern to load Date field	Select this check box to use the date model indicated in the schema for dates.
	Specify field condition	Select this check box to define a condition for loading data.
	Preserve blanks	Select this check box to preserve blank spaces.
	Trailing null columns	Select this check box to load data with all empty columns.
	Load options	Click **+** to add data loading options: **Parameter**: select a loading parameter from the list. **Value**: enter a value for the parameter selected.
	NLS Language	From the drop-down list, select the language for your data if the data is not in Unicode.
	Set Parameter NLS_TERRITORY	Select this check box to modify the conventions used for date and time formats. The default value is that of the operating system.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

	Set Oracle Encoding Type	Select this check box to type in the characterset next to the **Oracle Encoding Type** field.
	Output	Select the type of output for the standard output of the Oracle database: **to console**, **to global variable**.
	Convert columns and table names to uppercase	Select this check box to put columns and table names in upper case.
	Bulk file parameters	Set the parameters **Buffer Size** and **StringBuilder Size** for a performance gain according to the memory size.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tOracleOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tOracleRollback

tOracleRollback properties

This component is closely related to **tOracleCommit** and **tOracleConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleConnection** and **tOracleCommit**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleRollback** related scenario, see section *tMysqlRollback*

tOracleRow

tOracleRow properties

Component family	Databases/Oracle	
Function	**tOracleRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tOracleRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tOracleConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers.
	DB Version	Select the Oracle version in use.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased. You can also use PreparedStatement to avoid SQL injection. For a detailed scenario of utilizing this feature, see section *Scenario 2: Using PreparedStatement objects to query data*.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

- section *Scenario 2: Using PreparedStatement objects to query data*.

tOracleSCD

tOracleSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tOracleSCD*.

tOracleSCDELT

tOracleSCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tOracleSCDELT*.

tOracleSP

tOracleSP Properties

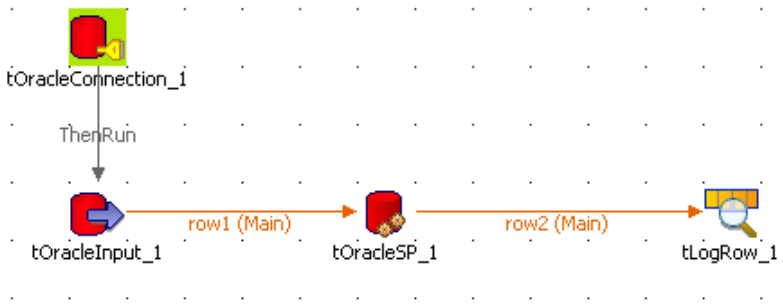
Component family	Databases/Oracle	
Function	**tOracleSP** calls the database stored procedure.	
Purpose	**tOracleSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Use an existing connection*	Select this check box to use an established connection from **tOracleConnection**. Once you select it, the **Component list** field appears allowing you to choose the **tOracleConnection** component to be used from those already established on the studio workspace. For more information on **tOracleConnection**, see section *tOracleConnection*. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Connection type	Drop-down list of available drivers.
	Property type	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	SP Name	Type in the exact name of the Stored Procedure (or Function)

	Is Function / Return result in	Select this check box, if the stored procedure is a function and one value only is to be returned. Select on the list the schema column, the value to be returned is based on.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure.
		Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameter is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the **tPostgresqlCommit** component if you want to analyze a set of records from a database table or DB query and return single records.
		The **Custom Type** is used when a **Schema Column** you want to use is user-defined. Two **Custom Type** columns are available in the **Parameters** table. In the first **Custom Type** column: - Select the check box in the **Custom Type** column when the corresponding **Schema Column** you want to use is of user-defined type. - If all listed **Schema Columns** in the **Parameters** table are of custom type, you can select the check box before **Custom Type** once for them all.
		Select a database type from the **DB Type** list to map the source database type to the target database type: **- Auto-Mapping**: Map the source database type to the target database type automatically.(default) **- CLOB**: Character large object **- BLOB**: Binary large object **- DECIMAL**: Decimal numeric object **- NUMERIC**: Character 0 to 9
		In the second **Custom Type** column, you can precise what the custom type is. The type may be: - **STRUCT**: used for one element. - **ARRAY**: used for a collection of elements.
		In the **Custom name** column, specify the name of the custom type that you have given to this type. *When an OUT parameter uses the custom type, make sure that its corresponding Schema Column has chosen the Object type in the schema table.*
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	NLS Language	In the list, select the language used for the data that are not used in Unicode.
	NLS Territory	Select the conventions used for date and time formats. The default value is that of the operating system.

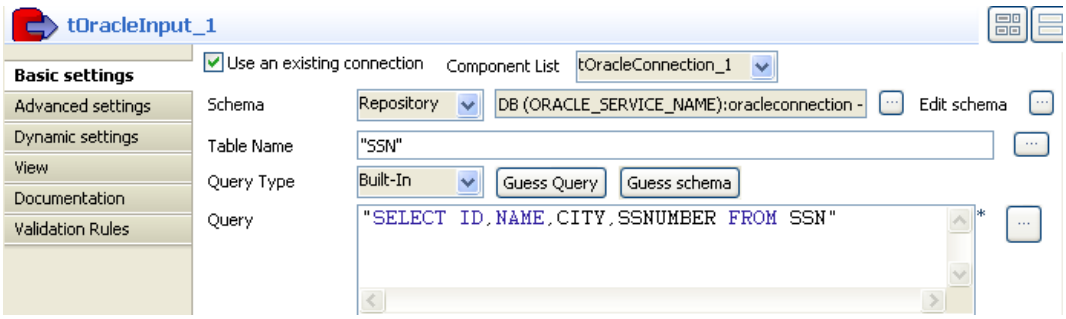
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax. When the parameters set in this component are of **Custom Type**, the **tJava** family components should be placed before the component in order for users to define values for the custom-type parameters, or after the component so as to read and output the **Out**-type custom parameters.	

Scenario: Checking number format using a stored procedure

The following job aims at connecting to an Oracle Database containing Social Security Numbers and their holders' name, calling a stored procedure that checks the SSN format of against a standard ###-##-#### format. Then the verification output results, 1 for valid format and 0 for wrong format get displayed onto the execution console.



- Drag and drop the following components from the **Palette**: **tOracleConnection**, **tOracleInput**, **tOracleSP** and **tLogRow**.
- Link the **tOracleConnection** to the **tOracleInput** using a **Then Run** connection as no data is handled here.
- And connect the other components using a **Row Main** link as rows are to be passed on as parameter to the SP component and to the console.
- In the **tOracleConnection**, define the details of connection to the relevant Database. You will then be able to reuse this information in all other DB-related components.
- Then select the **tOracleInput** and define its properties.



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

- Select the **Use an existing connection** check box and select the **tOracleConnection** component in the list in order to reuse the connection details that you already set.

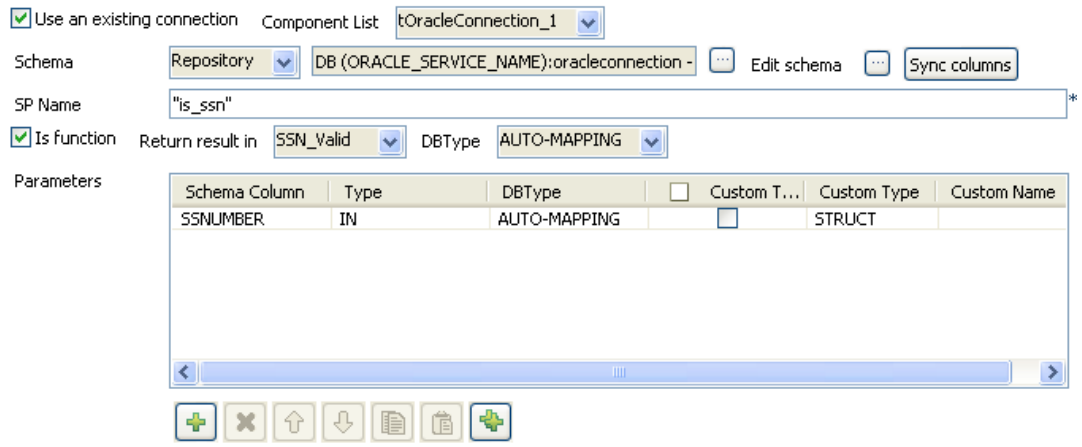
- Select **Repository** as **Property type** as the Oracle schema is defined in the DB Oracle connection entry of the Repository. If you haven't recorded the Oracle DB details in the **Repository**, then fill in the Schema name manually.
- Then select **Repository** as **Schema**, and retrieve the relevant schema corresponding to your Oracle DB table.

	ID	NAME	CITY	SSNUMBER
1	1	Jack	LA	123-45-6789
2	2	Tom	NYC	123-A5-6789
3	3	Bill	SF	123=45-6789
4	4	Jana	NYC	236-52-2956
5	5	Brandon	SLC	561-52-B267

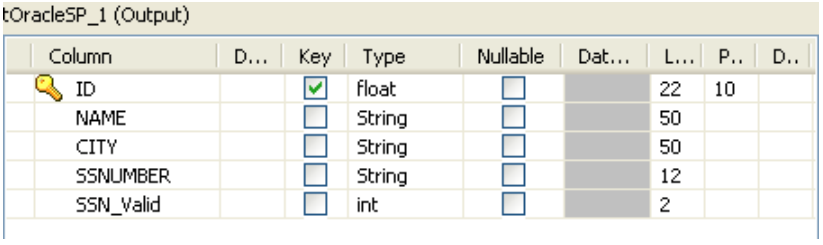
- In this example, the SSN table has a four-column schema that includes *ID*, *NAME*, *CITY* and *SSNUMBER*.
- In the **Query** field, type in the following Select query or select it in the list, if you stored it in the Repository.

```
select ID, NAME, CITY, SSNUMBER from SSN
```

- Then select the **tOracleSP** and define its **Basic settings**.



- Like for the **tOracleInput** component, select **Repository** in the **Property type** field and select the **Use an existing connection** check box, then select the relevant entries in the respective list.
- The schema used for the **tOracleSP** slightly differs from the input schema. Indeed, an extra column (*SSN_Valid*) is added to the Input schema. This column will hold the format validity status (*1* or *0*) produced by the procedure.

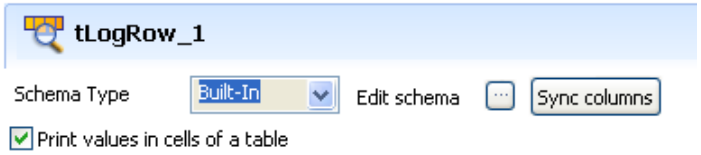


- In the **SP Name** field, type in the exact name of the stored procedure (or function) as called in the Database. In this use case, the stored procedure name is *is_ssn*.
- The basic function used in this particular example is as follows:

```
CREATE OR REPLACE FUNCTION is_ssn(string_in VARCHAR2)
RETURN PLS_INTEGER
IS
-- validating ###-##-#### format
```

```
BEGIN
  IF TRANSLATE(string_in, '0123456789A', 'AAAAAAAAAAB') =
     'AAA-AA-AAAA' THEN
    RETURN 1;
  END IF;
  RETURN 0;
END is_ssn;
/
```

- As a return value is expected in this use case, the procedure acts as a function, so select the **Is function** check box.
- The only return value expected is based on the *ssn_valid* column, hence select the relevant list entry.
- In the **Parameters** area, define the input and output parameters used in the procedure. In this use case, only the *SSNumber* column from the schema is used in the procedure.
- Click the plus sign to add a line to the table and select the relevant column (*SSNumber*) and type (*IN*).
- Then select the **tLogRow** component and click Sync Column to make sure the schema is passed on from the preceding **tOracleSP** component.



- Select the **Print values in cells of a table** check box to facilitate the output reading.
- Then save your job and press **F6** to run it.

```
Starting job OracleSP at 15:14 23/08/2007.
.---+--------+----+-----------+---------.
|               tLogRow_1                |
|=--+--------+----+-----------+---------=|
|ID |NAME    |CITY|SSNUMBER   |SSN_Valid|
|=--+--------+----+-----------+---------=|
|1.0|Jack    |LA  |123-45-6789|1        |
|2.0|Tom     |NYC |123-A5-6789|0        |
|3.0|Bill    |SF  |123=45-6789|0        |
|4.0|Jana    |NYC |236-52-2956|1        |
|5.0|Brandon|SLC |561-52-B267|0        |
'---+--------+----+-----------+---------'

Job OracleSP ended at 15:14 23/08/2007. [exit code=0]
```

On the console, you can read the output results. All input schema columns are displayed eventhough they are not used as parameters in the stored procedure.

The final column shows the expected return value, i.e. whether the SS Number checked is valid or not.

Check the **tPostgresqlCommit** component if you want to analyze a set of records from a database table or DB query and return single records.

tOracleTableList

tOracleTableList properties

Component family	Databases/Oracle	
Function	**tOracleTableList** iterates on a set of tables through a defined Oracle connection.	
Purpose	This component lists the names of specified Oracle tables using a SELECT statement based on a WHERE clause.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection is planned for the current Job.
	Where clause for table name selection	Enter the WHERE clause that will be used to identify the tables to iterate on.
Usage	This component is to be used along with other Oracle components, especially with **tOracleConnection**.	
Limitation	n/a	

Related scenarios

No scenario is available for this component yet.

tPostgresqlBulkExec

tPostgresqlBulkExec properties

tPostgresqlOutputBulk and **tPostgresqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tPostgresqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading in the database.

Component family	Databases/Postgresql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tPostgresqlBulkExec** offers gains in performance while carrying out the Insert operations to a Postgresql database	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tPostgrresqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again.

		Create table: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component.In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
Advanced settings	*Action on data*	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, job stops. **Bulk Update**: Make simultaneous changes to multiple entries.
	Copy the OID for each row	Retrieve the ID item for each row.
	Contains a header line with the names of each column in the file	Specify that the table contains header.
	File type	Select the type of file being handled.
	Null string	String displayed to indicate that the value is null..
	Fields terminated by	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped.
	Text enclosure	Character used to enclose text.
	Activate standard_conforming_string	Activate the variable.
	Force not null for columns	Define the columns nullability **Force not null**: Select the check box next to the column you want to define as not null.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tPostgresqlOutputBulk** component. Used together, they can offer gains in performance while feeding a Postgresql database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tPostgresqlBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresqlCommit

tPostgresqlCommit Properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually does not make much sense to use these components independently in a transaction.

Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tPostgresqlConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tPostgresqlCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlConnection** and **tPostgresqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlConnection** and **tPostgresqlRollback**. It usually does not make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tPostgresqlClose

tPostgresqlClose properties

Component family	Databases/Postgresql	
Function	**tPostgresqlClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tPostgresqlConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlConnection** and **tPostgresqlCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tPostgresqlConnection

tPostgresqlConnection Properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually doesn't make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

Component family	Databases/Postgresql	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlCommit** and **tPostgresqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually doesn't make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlConnection** related scenario, see section *tMysqlConnection*

tPostgresqlInput

tPostgresqlInput properties

Component family	Databases/ PostgreSQL	
Function	**tPostgresqlInput** reads a database and extracts fields based on a query.	
Purpose	**tPostgresqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tPostgresqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Postgresql databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tPostgresqlOutput

tPostgresqlOutput properties

Component family	Databases/Postgresql	
Function	**tPostgresqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tPostgresqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tPostgresqlConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is cleared by default, meaning to skip the row on error and to complete the process for error-free rows.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use save point	Select this check box to use savepoints in the transaction. This check box will not be available if you select: • the **Die on error** check box in the **Basic settings** view, or • the **Use Batch Size** check box in the **Advanced settings** view. This check box will not work if you: • type in *0* in the **Commit every** field, or • select the **Use an existing connection** check box in the **Basic settings** view while the **Auto Commit** mode is activated in the database connection component.
	Use field options	Select this check box to customize a request, especially when there is double action on data.

	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Ensure that the Nullable check box is selected for the corresponding columns in the schema.
	Use Batch Size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Postgresql database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tPostgresqlOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tPostgresqlOutputBulk

tPostgresqlOutputBulk properties

The **tPostgresqlOutputBulk** and **tPostgresqlBulkExec** components are generally used together as part of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tPostgresqlOutputBulkExec** component, detailed in a separate section. The advantage of having two separate steps is that it makes it possible to transform data before it is loaded in the database.

Component family	Databases/Postgresql	
Function	Writes a file with columns based on the defined delimiter and the Postgresql standards	
Purpose	Prepares the file to be used as parameters in the INSERT query to feed the Postgresql database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* User Guide.
	Append	Select this check box to add the new rows at the end of the file
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header to the file.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tPostgresqlBulkExec** component. Used together they offer gains in performance while feeding a Postgresql database.	

Related scenarios

For use cases in relation with **tPostgresqlOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresqlOutputBulkExec

tPostgresqlOutputBulkExec properties

The **tPostgresqlOutputBulk** and **tPostgresqlBulkExec** components are generally used together as part of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tPostgresqlOutputBulkExec** component.

Component family	Databases/Postgresql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Postgresql database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* **User Guide**
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Advanced settings	*Action on data*	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, job stops. **Bulk Update**: Make simultaneous changes to multiple entries.

	Copy the OID for each row	Retrieve the ID item for each row.
	Contains a header line with the names of each column in the file	Specify that the table contains header.
	Encoding	Select the encoding from the list or select **CUSTOM** and define it manually. This field is compulsory for DB data handling.
	File type	Select the type of file being handled.
	Null string	String displayed to indicate that the value is null..
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Fields terminated by	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped.
	Text enclosure	Character used to enclose text.
	Activate standard_conforming_string	Activate the variable.
	Force not null for columns	Define the columns nullability **Force not null**:: Select the check box next to the column you want to define as not null.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular tranformation is required on the data to be loaded onto the database.	

Related scenarios

For use cases in relation with **tPostgresqlOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresqlRollback

tPostgresqlRollback properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tPostgresqlConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlConnection** and **tPostgresqlCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlConnection** and **tPostgresqlCommit**. It usually does not make much sense to use one of them without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlRollback** related scenario, see section *tMysqlRollback*

tPostgresqlRow

tPostgresqlRow properties

Component family	Databases/Postgresql	
Function	**tPostgresqlRow** is the specific component for the database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tPostgresqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tPostgresqlConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema using CDC and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tPostgresqlSCD

tPostgresqlSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tPostgresqlSCD*.

tPostgresqlSCDELT

tPostgresqlSCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tPostgresqlSCDELT*.

tSybaseBulkExec

tSybaseBulkExec Properties

The **tSybaseOutputBulk** and **tSybaseBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tSybaseOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Database name
	Username and *Password*	DB user authentication data.
	Bcp Utility	Name of the utility to be used to copy data over to the Sybase server.
	Server	IP address of the database server for the Bcp utility connection.
	Batch size	Number of lines in each processed batch.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out.

		Drop and create a table: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Use an interface file*	Select this check box to specify an interface file in the field **Interface file**.
	Additional JDBC parameters	Specify additional connection properties in the existing DB connection, to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters.
	Action on data	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, Job stops. **Bulk Update**: Make simultaneous changes to multiple entries.
	Field Terminator	Character, string or regular expression to separate fields. *With the row/field separators compliant with the Sybase syntax, this component allows for the use of Sybase-orientated characters, such as \x09.*
	Row Terminator	String (ex: “\n” in Unix) to separate lines.
	Head row	Number of head lines to be ignored in the beginning of a file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Output	Select the type of output for the standard output of the Sybase database: **to console**, **to global variable**.
	tStataCatcher statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	As opposed to the Oracle dedicated bulk component, no action on data is possible using this Sybase dedicated component. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tSybaseBulkExec** related topics, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tSybaseClose

tSybaseClose properties

Function	**tSybaseClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tSybaseConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Sybase components, especially with **tSybaseConnection** and **tSybaseCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tSybaseCommit

tSybaseCommit Properties

This component is closely related to **tSybaseConnection** and **tSybaseRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Sybase	
Function	**tSybaseCommit** validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tSybaseConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tSybaseCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Sybase components, especially with **tSybaseConnection** and **tSybaseRollback**.	
Limitation	n/a	

Related scenario

This component is closely related to **tSybaseConnection** and **tSybaseRollback**. It usually does not make much sense to use one of these without using a **tSybaseConnection** component to open a connection for the current transaction.

For **tSybaseCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tSybaseConnection

tSybaseConnection Properties

This component is closely related to **tSybaseCommit** and **tSybaseRollback**. It usually does not make much sense to use one of these without using a **tSybaseConnection** component to open a connection for the current transaction.

Component family	Databases/Sybase	
Function	**tSybaseConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Sybase components, especially with **tSybaseCommit** and **tSybaseRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a **tSybaseConnection** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tSybaseInput

tSybaseInput Properties

Component family	Databases/Sybase	
Function	**tSybaseInput** reads a database and extracts fields based on a query.	
Purpose	**tSybaseInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the Sybase schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Sybase databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tSybaseIQBulkExec

tSybaseIQBulkExec Properties

Component family	Databases/Sybase IQ	
Function	**tSybaseIQBulkExec** uploads a bulk file in a Sybase IQ database.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase IQ database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	The available Sybase versions are: - **Sybase IQ 12**; - **Sybase IQ 15**. • *For **Sybase IQ 12**, the file to be bulk-loaded must be located on the same machine as the **Sybase IQ 12** server.* • *For **Sybase IQ 15**, the file to be bulk-loaded can be located on a remote machine. However, this means certain setup on the **Sybase IQ 15** server. For details, see Sybase IQ client-side load support enhancements.* The **Sybase IQ 15** version is connected to via ODBC while the **Sybase IQ 12** version is via JDBC, so the fields to be completed on the **Basic settings** view vary slightly between the alternative versions.
	Use an existing connection *Sybase IQ 12 only.*	Select this check box and click the relevant **tSybaseConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host *Sybase IQ 12 only.*	Database server IP address
	Port	Listening port number of DB server.

	Sybase IQ 12 only.	
	Data Source *Sybase IQ 15 only.*	Select the type of the data source to be used and complete the corresponding DSN information in the field alongside. The available types are: - **DSN**; - **FILEDSN**. When the **FILEDSN** type is used, a three-dot button appears next to the **Data Source** field to allow you to browse to the data source file of interest.
	Database	Database name
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Local filename	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Additional JDBC Parameters*	Specify additional connection properties in the existing DB connection, to allow specific character set support.
	Lines terminated by	Character or sequence of characters used to separate lines.
	Field Terminated by	Character, string or regular expression to separate fields. With the row/field separators compliant with the Sybase syntax, this component allows the use of Sybase-oriented separators, such as *\x09*.
	Use enclosed quotes	Select this check box to use data enclosure characters.
	Use fixed length	Select this check box to set a fixed width for data lines.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This dedicated component offers performance and flexibility of Sybase IQ DB query handling.	
Limitation	As opposed to the Oracle dedicated bulk component, no action on data is possible using this Sybase dedicated component. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

	The jodbc.jar also needs to be installed separately in the **Modules** view of the **Integration** perspective in your studio. *The DBMS's Client application should be installed on the same machine as your studio for successful bulk operations.*

Related scenarios

For **tSybaseIQBulkExec** related topics, see:

- section *Scenario: Bulk-loading data to a Sybase IQ 12 database*.
- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tSybaseIQOutputBulkExec

tSybaseIQOutputBulkExec properties

Component family	Databases/Sybase IQ	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase IQ database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	DB Version	The available Sybase versions are: - **Sybase IQ 12**; - **Sybase IQ 15**. • *For **Sybase IQ 12**, the file to be bulk-loaded must be located on the same machine as the **Sybase IQ 12** server.* • *For **Sybase IQ 15**, the file to be bulk-loaded can be located on a remote machine. However, this means certain setup on the **Sybase IQ 15** server. For details, see Sybase IQ client-side load support enhancements.* The **Sybase IQ 15** version is connected to via ODBC while the **Sybase IQ 12** version is via JDBC, so the fields to be completed on the **Basic settings** view vary slightly between the alternative versions.
	Use an existing connection *Sybase IQ 12 only.*	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.

	Sybase IQ 12 only.	
	Port *Sybase IQ 12 only.*	Listening port number of DB server.
	Data Source *Sybase IQ 15 only.*	Select the type of the data source to be used and complete the corresponding DSN information in the field alongside. The available types are: - **DSN**; - **FILEDSN**. When the **FILEDSN** type is used, a three-dot button appears next to the **Data Source** field to allow you to browse to the data source file of interest.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append the file	select this check box to add the new rows at the end of the records.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Additional JDBC Parameters*	Specify additional connection properties in the existing DB connection, to allow specific character set support.
	Fields terminated by	Character, string or regular expression to separate fields. *As a combination of **tSybaseOutputBulk** and **tSybaseIQBulkExec**, this component does not allow the use of Sybase-oriented row/field separators, such as \x09. To achieve the desired effect (for example, displaying fields in the tabular form), you need to use **tSybaseOutputBulk** and **tSybaseIQBulkExec** together to replace **tSybaseIQOutputBulkExec**, with \t used in the former component and \x09 used in the latter.*
	Lines terminated by	Character or sequence of characters used to separate lines.
	Use enclose quotes	Select this check box to use data enclosure characters.
	Include Head	Select this heck box to include the column header.

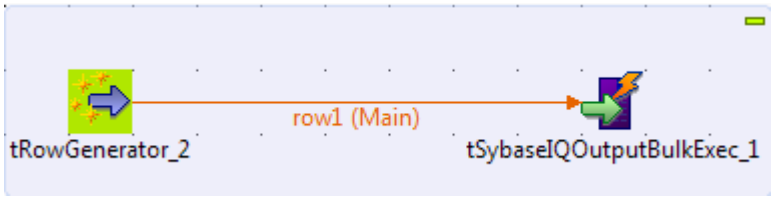
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*. The jodbc.jar also needs to be installed separately in the **Modules** view of the **Integration** perspective in your studio. *The DBMS's Client application should be installed on the same machine as your studio for successful bulk operations.*	

Scenario: Bulk-loading data to a Sybase IQ 12 database

This scenario saves data from a **tRowGenerator** to a file and then bulk-loads the data to a Sybase IQ 12 database.

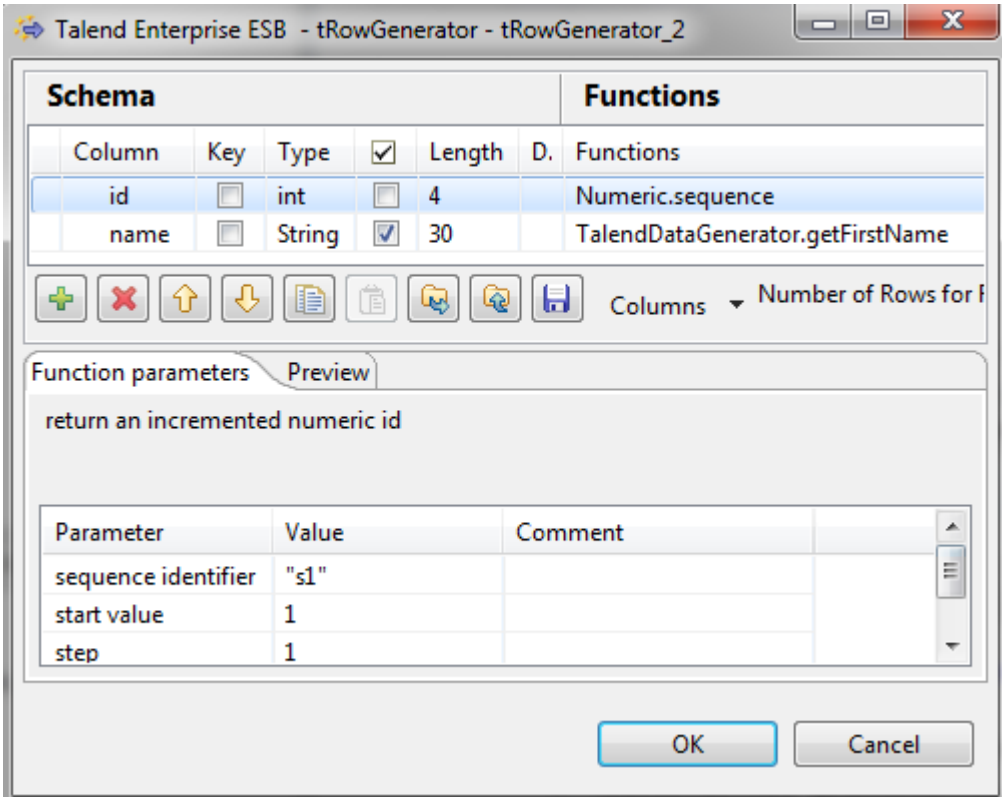
Linking the components

1. Drop **tRowGenerator** and **tSybaseIQOutputBulkExec** onto the workspace.
2. Link **tRowGenerator** to **tSybaseIQOutputBulkExec** using a **Row** > **Main** connection.

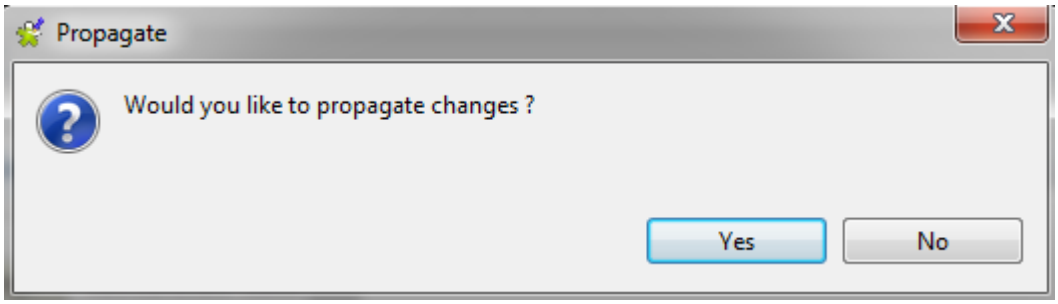


Configuring the components

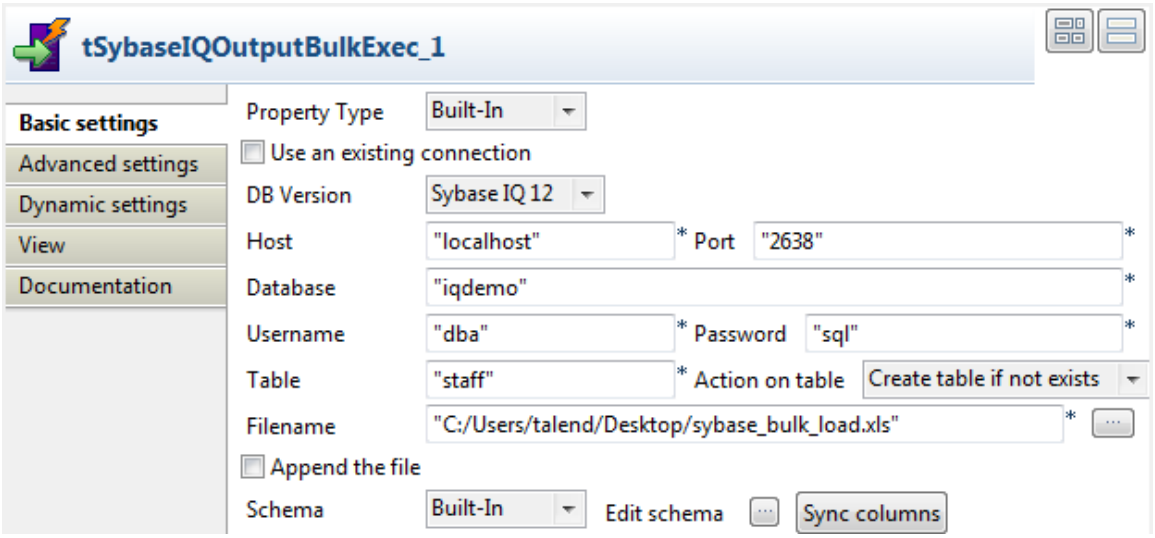
1. Double-click **tRowGenerator** to open its schema editor.



2. Click the **[+]** button to add two columns, namely *id* and *name*.
3. Select the type for *id* and *name*, respectively *int* and *String*.
4. Set the length for *id* and *name*, respectively *4* and *30*.
5. Select the function for *id* and *name*, respectively `Numeric.sequence` and `TalendDataGenerator.getFirstName`.
6. Click **Ok** to close the editor and click **Yes** on the pop-up below to propagate changes:



7. Double-click **tSybaseIQOutputBulkExec** to open its **Basic settings** view.



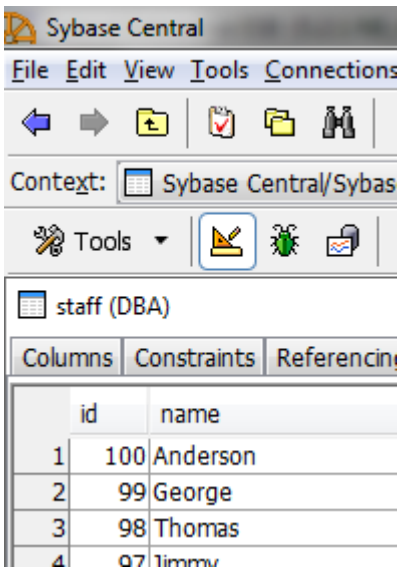
8. In the **Host** and **Port** fields, enter the connection details.
9. In the **Database** field, enter the database name.
10. In the **Username** and **Password** fields, enter the authentication credentials.
11. In the **Table** field, enter the table name.
12. In the **Action on table** list, select **Create table if not exists**.
13. In the **Filename** field, enter the full path of the file to hold the data.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run the Job.

```
Starting job usecase_sybase at 14:50 14/11/2012.


[statistics] connecting to socket on port 3643
[statistics] connected
[statistics] disconnected
Job usecase_sybase ended at 14:50 14/11/2012. [exit code=0]
```

3. In the Sybase Central console, open the table *staff* to check the data:



As shown above, the table is created with data inserted.

Related scenarios

For use cases in relation with **tSybaseIQOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tSybaseOutput

tSybaseOutput Properties

Component family	Databases/Sybase	
Function	**tSybaseOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tSybaseOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the Sybase schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

		Truncate table: The table content is deleted. You do not have the possibility to rollback the operation.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed.

		This check box is available only when you have selected the **Insert**, the **Update** or the **Delete** option in the **Action on data** field.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Sybase database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tSybaseOutput**, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tSybaseOutputBulk

tSybaseOutputBulk properties

The **tSybaseOutputBulk** and **tSybaseBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tSybaseOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases/Sybase	
Function	Writes a file with columns based on the defined delimiter and the Sybase standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Sybase database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields. *Fully in line with the Java syntax, this component does not allow the use of Sybase-orientated row/field separators, such as \x09.*
	Include header	Select this check box to include the column header in the file.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level
Usage	This component is to be used along with **tSybaseBulkExec** component. Used together they offer gains in performance while feeding a Sybase database.	

Related scenarios

For use cases in relation with **tSybaseOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.

- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tSybaseOutputBulkExec

tSybaseOutputBulkExec properties

The **tSybaseOutputBulk** and **tSybaseBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tSybaseOutputBulkExec** component.

Component family	Databases/Sybase	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file in which the properties are stored. The fields that follow are completed automatically using the data retrieved.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Bcp utility	Name of the utility to be used to copy data over to the Sybase server.
	Batch row number	Number of lines in each processed batch.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again.

		Create a table: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Append	Select this check box to add the new rows at the end of the records.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created and stored the schema in the Repository, hence can be reused. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Advanced settings	*Use an interface file*	Select this check box to specify an interface file in the field **Interface file**.
	Additional JDBC parameters	Specify additional connection properties in the existing DB connection, to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters.
	Action on data	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, job stops. **Bulk Update**: Make simultaneous changes to multiple entries.
	Field terminator	Character, string or regular expression to separate fields. *As a combination of **tSybaseOutputBulk** and **tSybaseBulkExec**, this component does not allow the use of Sybase-oriented row/field separators, such as \x09. To achieve the desired effect (for example, displaying fields in the tabular form), you need to use **tSybaseOutputBulk** and **tSybaseBulkExec** together to replace **tSybaseOutputBulkExec**, with \t used in the former component and \x09 used in the latter.*
	DB Row terminator	String (ex: “\n”on Unix) to distinguish rows in the DB.
	First row NO. of file	Type in the number of the file row where the action should start at.
	FILE Row terminator	Character, string or regular expression to separate fields in a file.
	Include Head	Select this heck box to include the column header.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Output	Select the type of output for the standard output of the Sybase database: **to console,** **to global variable.**
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tSybaseOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tSybaseRollback

tSybaseRollback properties

This component is closely related to **tSybaseCommit** and **tSybaseConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Sybase	
Function	**tSybaseRollback** cancels the transaction committed in the connected DB.	
Purpose	This component avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tSybaseConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Sybase components, especially with **tSybaseConnection** and **tSybaseCommit**.	
Limitation	n/a	

Related scenarios

For **tSybaseRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tSybaseRow

tSybaseRow Properties

Component family	Databases/Sybase	
Function	**tSybaseRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tSybaseRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the sybase schema.
	Username and *Password*	DB user authentication data.
	Table Name	Name of the table to be processed.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.

		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tSybaseRow** related topics, see:

- section *Scenario 3: Combining two flows for selective output* .
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tSybaseSCD

tSybaseSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tSybaseSCD*.

tSybaseSCDELT

tSybaseSCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tSybaseSCDELT*.

tSybaseSP

tSybaseSP properties

Component family	Databases/Sybase	
Function	**tSybaseSP** calls the database stored procedure.	
Purpose	**tSybaseSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Select this check box, if a value is to be returned. Select on the list the schema column, the value to be returned is based on.
	Timeout Interval	Maximum waiting time for the results of the stored procedure.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure.

		Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the **tPostgresqlCommit** component if you want to analyze a set of records from a database table or DB query and return single records.
Advanced settings	*Additional JDBC Parameters*	Fill in additional connection properties for the DB connection you are creating. This option is available when the checkbox **Use an existing connection** is not selected in the **Basic settings**.
	Use Multiple SELECT Procedure	Select this check box to use procedures which contain multiple SELECT statements.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topic, see section *Scenario: Finding a State Label using a stored procedure*.

Check section *tMysqlConnection* as well if you want to analyze a set of records from a database table or DB query and return single records.

Databases - appliance/datawarehouse components

This chapter describes connectors for specific databases oriented to the processing of large volume of data. These connectors cover various needs, including: opening connections, reading and writing tables, committing transactions as a whole, and performing rollback for error handling. These components can be found in the **Palette** of the **Integration** perspective of the **Talend** Studio.

Other types of database connectors, such as connectors for traditional databases and database management, are documented in *Databases - traditional components* and *Databases - other components*.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tGreenplumBulkExec

tGreenplumBulkExec Properties

The **tGreenplumOutputBulk** and **tGreenplumBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT statement used to feed a database. These two steps are fused together in the **tGreenplumOutputBulkExec** component, detailed in a separate section. The advantage using a two step process is that it makes it possible to transform data before it is loaded in the database.

Component Family	Databases/Greenplum	
Function	**tGreenplumBulkExec** performs an Insert action on the data.	
Purpose	**tGreenplumBulkExec** is a component which is specifically designed to improve performance when loading data in ParAccel database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box if you use a configured **tGreenplumConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created.

		Create a table if not exists: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Filename	Path and name of the file to be processed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Action on data*	Select the operation you want to perform: **Bulk insert Bulk update** The details asked will be different according to the action chosen.
	Copy the OID for each row	Retrieve the ID item for each row.
	Contains a header line with the names of each column in the file	Specify that the table contains header.
	File type	Select the file type to process.
	Null string	String displayed to indicate that the value is null.
	Fields terminated by	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Force not null for columns	Define the columns nullability **Force not null**:: Select the check box next to the column you want to define as not null.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is generally used with a **tGreenplumOutputBulk** component. Used together they offer gains in performance while feeding a Greenplum database.	

Related scenarios

For more information about **tGreenplumBulkExec**, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tGreenplumClose

tGreenplumClose properties

Component family	Databases/Greenplum	
Function	**tGreenplumClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tGreenplumConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Greenplum components, especially with **tGreenplumConnection** and **tGreenplumCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tGreenplumCommit

tGreenplumCommit Properties

This component is closely related to **tGreenplumConnection** and **tGreenplumRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Greenplum	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tGreenplumConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tGreenplumCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Greenplum components, especially with **tGreenplumConnection** and **tGreenplumRollback** components.	
Limitation	n/a	

Related scenarios

This component is closely related to **tGreenplumConnection** and **tGreenplumRollback**. It usually doesn't make much sense to use one of these without using a **tGreenplumConnection** component to open a connection for the current transaction.

For **tGreenplumCommit** related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*.
- section *tMysqlConnection*.

tGreenplumConnection

tGreenplumConnection properties

This component is closely related to **tGreenplumCommit** and **tGreenplumRollback**. It usually does not make much sense to use one of these without using a **tGreenplumConnection** to open a connection for the current transaction.

Component family	Databases/Greenplum	
Function	**tGreenplumConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with Greenplum components, especially with **tGreenplumCommit** and **tGreenplumRollback**.	
Limitation	n/a	

Related scenarios

This component is closely related to **tGreenplumCommit** and **tGreenplumRollback**. It usually does not make much sense to use one of these without using a **tGreenplumConnection** component to open a connection for the current transaction.

For **tGreenplumConnection** related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*.
- section *tMysqlConnection*.

tGreenplumGPLoad

This component invokes Greenplum's gpload utility to insert records into a Greenplum database. This component can be used either in standalone mode, loading from an existing data file, or connected to an input flow to load data from the connected component.

tGreenplumGPLoad properties

Component family	Databases/Greenplum	
Function	**tGreenplumGPLoad** inserts data into a Greenplum database table using Greenplum's gpload utility.	
Purpose	This component is used to bulk load data into a Greenplum table either from an existing data file, an input flow, or directly from a data flow in streaming mode through a named-pipe.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the Greenplum database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table into which the data is to be inserted.
	Action on table	On the table defined, you can perform one of the following operations before loading the data: **None**: No operation is carried out. **Clear table**: The table content is deleted before the data is loaded. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop and create table**: The table is removed and created again. **Drop table if exists and create**: The table is removed if it already exists and created again. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries. **Merge**: Updates or adds data to the table. *It is necessary to specify at least one column as a primary key on which the **Update** and **Merge** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). To define the **Update**/**Merge options**, select in the **Match Column** column the check boxes corresponding to the column names that*

		*you want to use as a base for the **Update** and **Merge** operations, and select in the **Update Column** column the check boxes corresponding to the column names that you want to update. To define the **Update condition**, type in the condition that will be used to update the data.*
	Schema and Edit schema	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Data file	Full path to the data file to be used. If this component is used in standalone mode, this is the name of an existing data file to be loaded into the database. If this component is connected with an input flow, this is the name of the file to be generated and written with the incoming data to later be used with gpload to load into the database. This field is hidden when the **Use named-pipe** check box is selected.
	Use named-pipe	Select this check box to use a named-pipe. This option is only applicable when the component is connected with an input flow. When this check box is selected, no data file is generated and the data is transferred to gpload through a named-pipe. This option greatly improves performance in both Linux and Windows. This component on named-pipe mode uses a JNI interface to create and write to a named-pipe on any Windows platform. Therefore the path to the associated JNI DLL must be configured inside the java library path. The component comes with two DLLs for both 32 and 64 bit operating systems that are automatically provided in the Studio with the component.
	Named-pipe name	Specify a name for the named-pipe to be used. Ensure that the name entered is valid.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Use existing control file (YAML formatted)*	Select this check box to provide a control file to be used with the gpload utility instead of specifying all the options explicitly in the component. When this check box is selected, **Data file** and the other gpload related options no longer apply. Refer to Greenplum's gpload manual for details on creating a control file.
	Control file	Enter the path to the control file to be used, between double quotation marks, or click **[...]** and browse to the control file. This option is passed on to the gpload utility via the *-f* argument.
	CSV mode	Select this check box to include CSV specific parameters such as **Escape char** and **Text enclosure**.
	Field separator	Character, string, or regular expression used to separate fields. *This is gpload's delim argument. The default value is |. To improve performance, use the default value.*
	Escape char	Character of the row to be escaped.
	Text enclosure	Character used to enclose text.
	Header (skips the first row of data file)	Select this check box to skip the first row of the data file.
	Additional options	Set the gpload arguments in the corresponding table. Click **[+]** as many times as required to add arguments to the table. Click the **Parameter** field and choose among the arguments from the list. Then click the corresponding **Value** field and enter a value between quotation marks.
		LOCAL_HOSTNAME: The host name or IP address of the local machine on which gpload is running. If this machine is configured with multiple network interface cards (NICs), you can specify the

		host name or IP of each individual NIC to allow network traffic to use all NICs simultaneously. By default, the local machine's primary host name or IP is used.
		PORT (gpfdist port): The specific port number that the gpfdist file distribution program should use. You can also specify a **PORT_RANGE** to select an available port from the specified range. If both **PORT** and **PORT_RANGE** are defined, then **PORT** takes precedence. If neither **PORT** or **PORT_RANGE** is defined, an available port between 8000 and 9000 is selected by default. If multiple host names are declared in **LOCAL_HOSTNAME**, this port number is used for all hosts. This configuration is desired if you want to use all NICs to load the same file or set of files in a given directory location.
		PORT_RANGE: Can be used instead of **PORT (gpfdist port)** to specify a range of port numbers from which gpload can choose an available port for this instance of the gpfdist file distribution program.
		NULL_AS: The string that represents a null value. The default is `\N` (backslash-N) in TEXT mode, and an empty value with no quotation marks in CSV mode. Any source data item that matches this string will be considered a null value.
		FORCE_NOT_NULL: In CSV mode, processes each specified column as though it were quoted and hence not a NULL value. For the default null string in CSV mode (nothing between two delimiters), this causes missing values to be evaluated as zero-length strings.
		ERROR_LIMIT (2 or higher): Enables single row error isolation mode for this load operation. When enabled and the error limit count is not reached on any Greenplum segment instance during input processing, all good rows will be loaded and input rows that have format errors will be discarded or logged to the table specified in **ERROR_TABLE** if available. When the error limit is reached, input rows that have format errors will cause the load operation to abort. Note that single row error isolation only applies to data rows with format errors, for example, extra or missing attributes, attributes of a wrong data type, or invalid client encoding sequences. Constraint errors, such as primary key violations, will still cause the load operation to abort if encountered. When this option is not enabled, the load operation will abort on the first error encountered.
		ERROR_TABLE: When **ERROR_LIMIT** is declared, specifies an error table where rows with formatting errors will be logged when running in single row error isolation mode. You can then examine this error table to see error rows that were not loaded (if any).
	Log file	Browse to or enter the access path to the log file in your directory.
	Encoding	Define the encoding type manually in the field.
	Specify gpload path	Select this check box to specify the full path to the gpload executable. You must check this option if the gpload path is not specified in the PATH environment variable.
	Full path to gpload executable	Full path to the gpload executable on the machine in use. It is advisable to specify the gpload path in the PATH environment variable instead of selecting this option.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded on to the database. This component can be used as a standalone or an output component.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related use case, see section *Scenario: Inserting data in MySQL database*.

tGreenplumInput

tGreenplumInput properties

Component family	Databases/Greenplum	
Function	**tGreenplumInput** reads a database and extracts fields based on a query.	
Purpose	**tGreenplumInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Guess schema	Click the **Guess schema** button to retrieve the table schema.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for FireBird databases.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario: Mapping data using a simple implicit join*.
- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also related topic: section *Scenario: Dynamic context use in MySQL DB insert*.

tGreenplumOutput

tGreenplumOutput Properties

Component Family	Databases/Greenplum	
Function	**tGreenplumOutput** writes, updates, modifies or deletes the data in a database.	
Purpose	**tGreenplumOutput** executes the action defined on the table and/or on the data of a table, according to the input flow form the previous component.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box if you use a configured **tGreenplumConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column** , select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Greenplum databases. It allows you to carry out actions on a table or on the data of a table in a Greenplum database. It enables you to create a reject flow, with a **Row** > **Reject**s link filtering the data in error. For a usage example, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related scenarios

For related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*.
- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tGreenplumOutputBulk

tGreenplumOutputBulk properties

The **tGreenplumOutputBulk** and **tGreenplumBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tGreenplumOutputBulkExec** component, detailed in a separate section. The advantage of using a two step process is that it makes it possible to transform data before it is loaded in the database.

Component family	Databases/Greenplum	
Function	Writes a file with columns based on the defined delimiter and the Greenplum standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Greenplum database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the records
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check to include the column header.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStaCatcher statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tGreenplumBulkExec** component. Used together they offer gains in performance while feeding a Greenplum database.	

Related scenarios

For use cases in relation with **tGreenplumOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tGreenplumOutputBulkExec

tGreenplumOutputBulkExec properties

The **tGreenplumOutputBulk** and **tGreenplumBulkExec** components are used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tGreenplumOutputBulkExec** component.

Component family	Databases/Greenplum	
Function	Executes the action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Greenplum database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database name	Name of the database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted. You have the possibility to rollback the operation.
	File Name	Name of the file to be processed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Action on data*	Select the operation you want to perform: **Bulk insert Bulk update** The details asked will be different according to the action chosen.
	Copy the OID for each row	Retrieve the ID item for each row.
	Contains a header line with the names of each column in the file	Specify that the table contains header.

	File type	Select the file type to process.
	Null string	String displayed to indicate that the value is null.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Fields terminated by	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Force not null for columns	Define the columns nullability **Force not null**: Select the check box next to the column you want to define as not null.
	tStatCatcherStatistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tGreenplumOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tGreenplumRollback

tGreenplumRollback properties

This component is closely related to **tGreenplumCommit** and **tGreenplumConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Greenplum	
Function	**tGreenplumRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tGreenplumConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Greenplum components, especially with **tGreenplumConnection** and **tGreenplumCommit**.	
Limitation	n/a	

Related scenarios

For **tGreenplumRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tGreenplumRow

tGreenplumRow Properties

Component Family	Databases/Greenplum	
Function	**tGreenplumRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tGreenplumRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tFirebirdConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username et *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder.

	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For a related scenario, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tGreenplumSCD

tGreenplumSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tGreenplumSCD*.

tIngresBulkExec

tIngresBulkExec properties

tIngresOutputBulk and **tIngresBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tIngresOutputBulkExec** component, detailed in another section. The advantage of using two components is that data can be transformed before it is loaded in the database.

Component family	Databases/Ingres	
Function	Executes the Insert action on the data provided.	
Purpose	Inserts data in bulk to a table in the Ingres DBMS for performance gain.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Table	Name of the table to be filled.
	VNode	Name of the virtual node.
	Database	Name of the database.
	Action on table	Actions that can be taken on the table defined: **None**: No operation made to the table. **Truncate**: Delete all the rows in the table and release the file space back to the operating system.
	File name	Path and name of the file to be processed.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
	Delete Working Files After Use	Select this check box to delete the files that are created during the execution.
Advanced settings	*Field Separator*	Character, string or regular expression to separate fields.
	Row Separator	String (ex: “\n”on Unix) to separate rows
	Null Indicator	Value of the null indicator.
	Session User	User of the defined session (the connection to the database).
	Rollback	Enable or disable rollback.
	On Error	Policy of error handling: **Continue**: Continue the execution. **Terminate**: Terminate the execution.
	Reject Row File	Path and name of the file that holds the rejected rows. Available when **Continue** is selected from the **On Error** list.
	Error Count	Number of errors to trigger the termination of the execution. Available when **Terminate** is selected from the **On Error** list.
	Allocation	Number of pages initially allocated to the table or index.
	Extend	Number of pages by which a table or index grows.
	Fill Factor	Specify the percentage (from 1 to 100) of each primary data page that must be filled with rows, under ideal conditions. For example,

		if you specify a fillfactor of 40, the DBMS Server fills 40% of each of the primary data pages in the restructured table with rows.
	Min Pages/*Max Pages*	Specify the minimum/maximum number of primary pages a hash table must have. The Min. pages and Max. pages must be at least 1.
	Leaf Fill	A bulk copy from can specify a leaffill value. This clause specifies the percentage (from 1 to 100) of each B-tree leaf page that must be filled with rows during the copy. This clause can be used only on tables with a B-tree storage structure.
	Non Leaf Fill	A bulk copy from can specify a nonleaffill value. This clause specifies the percentage (from 1 to 100) of each B-tree non-leaf index page that must be filled with rows during the copy. This clause can be used only on tables with a B-tree storage structure.
	Row Estimate	Specify the estimated number of rows to be copied from a file to a table during a bulk copy operation.
	Trailing WhiteSpace	Selected by default, this check box is designed to trim the trailing white spaces and applies only to such data types as VARCHAR, NVARCHAR and TEXT.
	Encoding	List of the encoding schemes.
	Output	Where to output the error message: **to console**: Message output to the console. **to global variable**: Message output to the global variable.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	Deployed along with **tIngresOutputBulk**, **tIngresBulkExec** feeds the given data in bulk to the Ingres database for performance gain.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Loading data to a table in the Ingres DBMS*

tIngresClose

tIngresClose properties

Component family	Databases/Ingres	
Function	**tIngresClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tIngresConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Ingres components, especially with **tIngresConnection** and **tIngresCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tIngresCommit

tIngresCommit Properties

This component is closely related to **tIngresConnection** and **tIngresRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Ingres	
Function	Validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tIngresConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tIngresCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Ingres components, especially with **tIngresConnection** and **tIngresRollback**.	
Limitation	n/a	

Related scenario

For **tIngresCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tIngresConnection

tIngresConnection Properties

This component is closely related to **tIngresCommit** and **tIngresRollback**. It usually does not make much sense to use one of these without using a **tIngresConnection** component to open a connection for the current transaction.

Component family	Databases/Ingres	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Ingres components, especially with **tIngresCommit** and **tIngresRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tIngresConnection** related scenario, see section *Scenario: Loading data to a table in the Ingres DBMS*.

tIngresInput

tIngresInput properties

Component family	Databases/Ingres	
Function	**tIngresInput** reads a database and extracts fields based on a query.	
Purpose	**tIngresInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Ingres databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*
- section *Scenario 2: Using StoreSQLQuery variable*.

See also the scenario for **tContextLoad**: section *Scenario: Dynamic context use in MySQL DB insert*.

tIngresOutput

tIngresOutput properties

Component family	Databases/Ingres	
Function	**tIngresOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tIngresOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Ingres database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tIngresOutputBulk

tIngresOutputBulk properties

tIngresOutputBulk and **tIngresBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tIngresOutputBulkExec** component.

Component family	Databases/Ingres	
Function	Prepares a file with the schema defined and the data coming from the preceding component.	
Purpose	Prepares the file whose data is inserted in bulk to the Ingres DBMS for performance gain.	
Basic settings	*Property Type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Path and name of the file that will hold the incoming data.
	Append the File	Select this check box to add the new rows at the end of the file.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
Advanced settings	*Field Separator*	Character, string or regular expression to separate fields.
	Row Separator	String (ex: “\n”on Unix) to separate rows.
	Include Header	Select this check box to include the column header in the file.
	Encoding	List of encoding schemes.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	Deployed along with **tIngresBulkExec**, **tIngresOutputBulk** is intended to save the incoming data to a file, whose data is then inserted in bulk to an Ingres database by **tIngresBulkExec** for performance gain.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Loading data to a table in the Ingres DBMS*,

tIngresOutputBulkExec

tIngresOutputBulkExec properties

tIngresOutputBulk and **tIngresBulkExec** are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tIngresOutputBulkExec** component.

Component family	Databases/Ingres	
Function	Prepares an output file and uses it to feed a table in the Ingres DBMS.	
Purpose	Inserts data in bulk to a table in the Ingres DBMS for performance gain.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Table	Name of the table to be filled.
	VNode	Name of the virtual node.
	Database	Name of the database.
	Action on table	Actions that can be taken on the table defined: **None**: No operation made to the table. **Truncate**: Delete all the rows in the table and release the file space back to the operating system.
	File name	Path and name of the file that will hold the incoming data.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
	Delete Working Files After Use	Select this check box to delete the files that are created during the execution.
Advanced settings	*Field Separator*	Character, string or regular expression to separate fields.
	Row Separator	String (ex: “\n”on Unix) to separate rows
	On Error	Policy of error handling: **Continue**: Continue the execution. **Terminate**: Terminate the execution.
	Reject Row File	Path and name of the file that holds the rejected rows. Available when **Continue** is selected from the **On Error** list.
	Error Count	Number of errors to trigger the termination of the execution. Available when **Terminate** is selected from the **On Error** list.
	Rollback	Enable or disable rollback.
	Null Indicator	Value of the null indicator.
	Session User	User of the defined session (the connection to the database).
	Allocation	Number of pages initially allocated to the table or index.
	Extend	Number of pages by which a table or index grows.
	Fill Factor	Specify the percentage (from 1 to 100) of each primary data page that must be filled with rows, under ideal conditions. For example, if you specify a fillfactor of 40, the DBMS Server fills 40% of each of the primary data pages in the restructured table with rows.

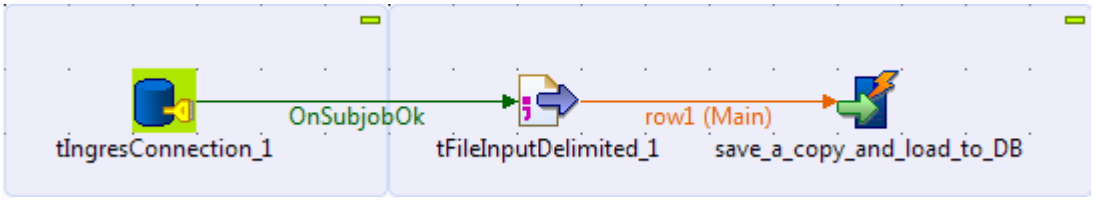
	Min Pages/Max Pages	Specify the minimum/maximum number of primary pages a hash table must have. The Min. pages and Max. pages must be at least 1.
	Leaf Fill	A bulk copy from can specify a leaffill value. This clause specifies the percentage (from 1 to 100) of each B-tree leaf page that must be filled with rows during the copy. This clause can be used only on tables with a B-tree storage structure.
	Non Leaf Fill	A bulk copy from can specify a nonleaffill value. This clause specifies the percentage (from 1 to 100) of each B-tree non-leaf index page that must be filled with rows during the copy. This clause can be used only on tables with a B-tree storage structure.
	Row Estimate	Specify the estimated number of rows to be copied from a file to a table during a bulk copy operation.
	Trailing WhiteSpace	Selected by default, this check box is designed to trim the trailing white spaces and applies only to such data types as VARCHAR, NVARCHAR and TEXT.
	Output	Where to output the error message: **to console**: Message output to the console. **to global variable**: Message output to the global variable.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	Usually deployed along with **tIngresConnection** or **tIngresRow**, **tIngresOutputBulkExec** prepares an output file and feeds its data in bulk to the Ingres DBMS for performance gain.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Loading data to a table in the Ingres DBMS

In this scenario, a **tIngresOutputBulkExec** component is deployed to prepare an output file with the employee data from a .csv file and then use that output file to feed a table in an Ingres database.

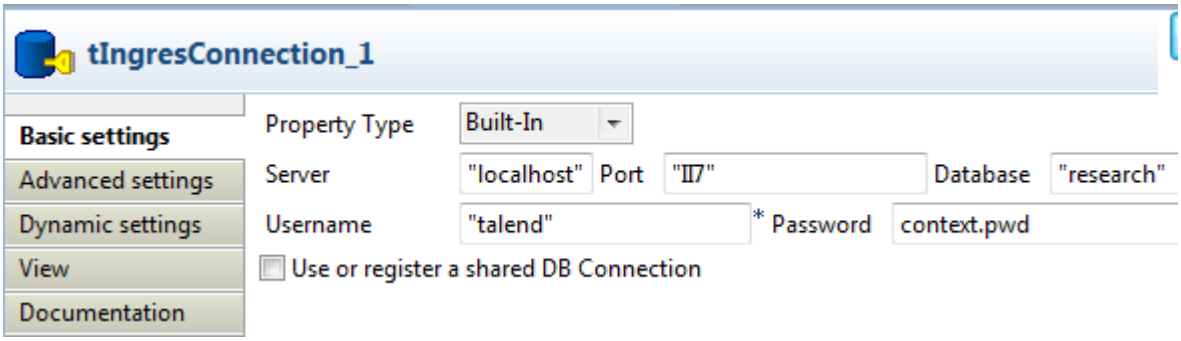
Dragging and dropping components

1. Drop **tIngresConnection**, **tFileInputDelimited** and **tIngresOutputBulkExec** from the **Palette** onto the workspace.
2. Rename **tIngresOutputBulkExec** as **save_a_copy_and_load_to_DB**.
3. Link **tIngresConnection** to **tFileInputDelimited** using an **OnSubjobOk** trigger.
4. Link **tFileInputDelimited** to **tIngresOutputBulkExec** using a **Row > Main** connection.



Configuring the components

1. Double-click **tIngresConnection** to open its **Basic settings** view in the **Component** tab.



2. In the **Server** field, enter the address of the server where the Ingres DBMS resides, for example *"localhost"*.

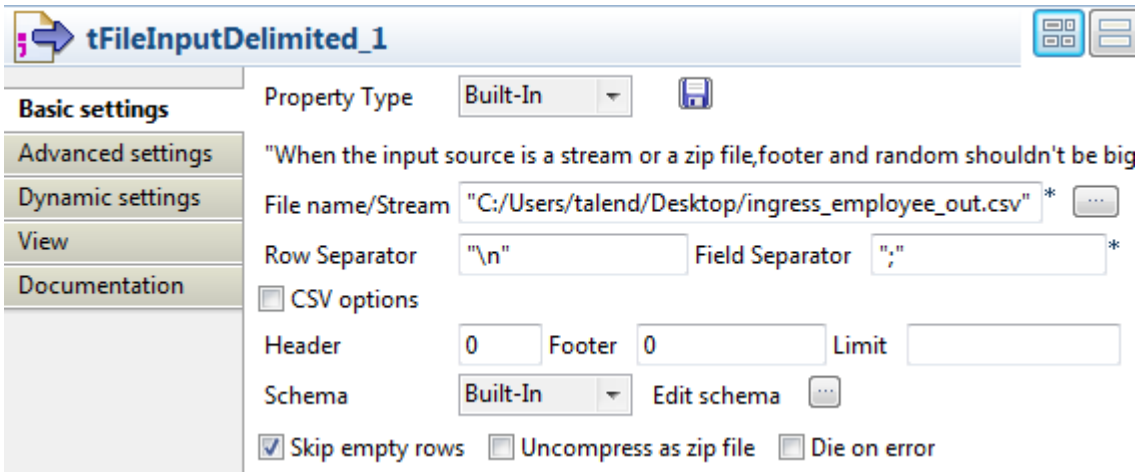
   Keep the default settings of the **Port** field.

3. In the **Database** field, enter the name of the Ingres database, for example *"research"*.

4. In the **Username** and **Password** fields, enter the authentication credentials.

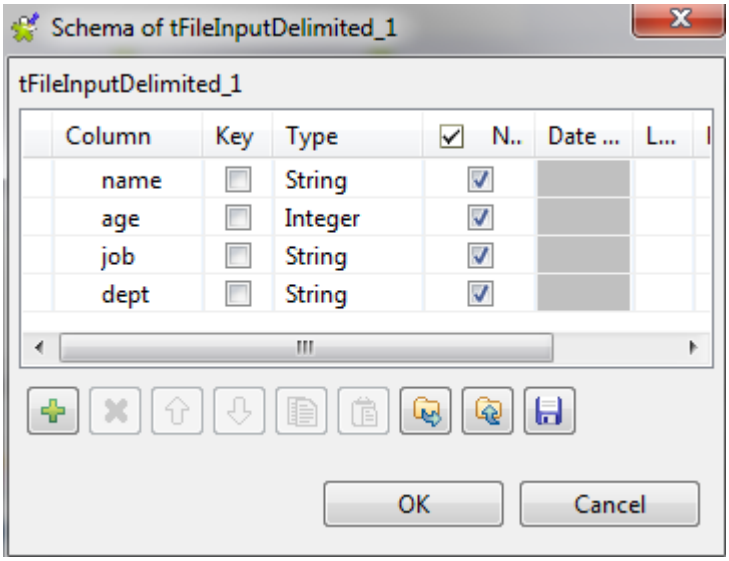
   A context variable is used for the password here. For more information on context variables, see *Talend Open Studio for Big Data User Guide*.

5. Double-click **tFileInputDelimited** to open its **Basic settings** view in the **Component** tab.



6. Select the source file by clicking the **[...]** button next to the **File name/Stream** field.

7. Click the **[...]** button next to the **Edit schema** field to open the schema editor.



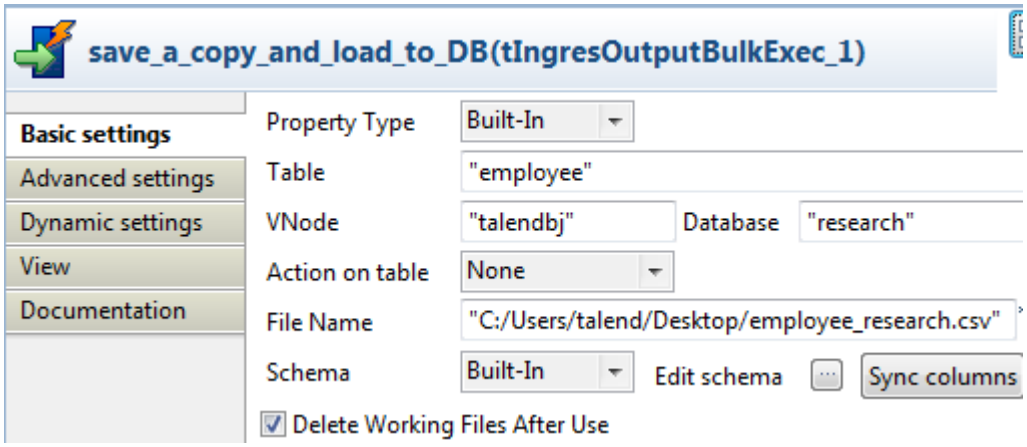
8. Click the **[+]** button to add four columns, for example *name*, *age*, *job* and *dept*, with the data type as *string*, *Integer*, *string* and *string* respectively.

   Click **OK** to close the schema editor.

   Click **Yes** on the pop-up window that asks whether to propagate the changes to the subsequent component.

   Leave other default settings unchanged.

9. Double-click **tIngresOutputBulkExec** to open its **Basic settings** view in the **Component** tab.



10. In the **Table** field, enter the name of the table for data insertion.

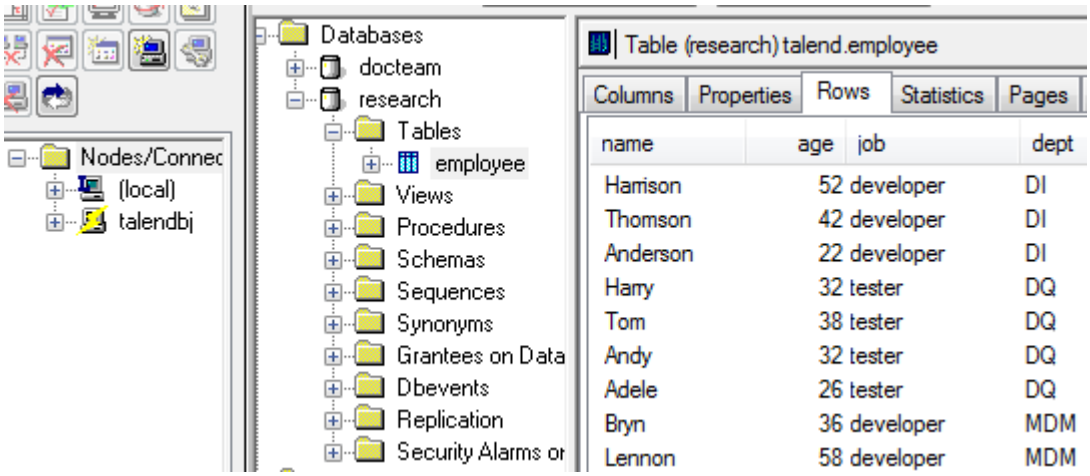
11. In the **VNode** and **Database** fields, enter the names of the VNode and the database.

12. In the **File Name** field, enter the full path of the file that will hold the data of the source file.

Executing the Job

1. Press **Ctrl+S** to save the Job.

2. Press **F6** to run the Job.



As shown above, the employee data is written to the table *employee* in the database *research* on the node *talendbj*. Meanwhile, the output file *employee_research.csv* has been generated at *C:/Users/talend/Desktop*.

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*,
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tIngresRollback

tIngresRollback properties

This component is closely related to **tIngresCommit** and **tIngresConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Ingres	
Function	**tIngresRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tIngresConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Ingres components, especially with **tIngresConnection** and **tIngresCommit**.	
Limitation	n/a	

Related scenarios

For **tIngresRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tIngresRow

tIngresRow properties

Component family	Databases/Ingres	
Function	**tIngresRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tIngresRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced Settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.

	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tIngresSCD

tIngresSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tIngresSCD*.

tNetezzaBulkExec

tNetezzaBulkExec properties

Component family	Databases/Netezza	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tNetezzaBulkExec** offers gains in performance while carrying out the Insert operations to a Netezza database	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when you are using the component **tNetezzaConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Field Separator*	Character, string or regular expression to separate fields.
	Require quotes (“) around data files	Select this check box to use data enclosure characters.
	Row Separator	String (ex: “\n”on Unix) to distinguish rows.

	Escape character	Character of the row to be escaped.
	Date format / Date delimiter	Use Date format to distinguish the way years, months and days are represented in a string. Use Date delimiter to specify the separator between date values.
	Time format/ Time delimiter	Use Time format to distinguish the time is represented in a string. Use Time delimiter to specify the separator between time values.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Max Errors	Enter the maximum error limit that will not stop the process.
	Skip Rows	Enter the number of rows to be skipped.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when non particular transformation is required on the data to be loaded on to the database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tNetezzaBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tNetezzaClose

tNetezzaClose properties

Component family	Databases/Netezza	
Function	**tNetezzaClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tNetezzaConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaConnection** and **tNetezzaCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tNetezzaCommit

tNetezzaCommit Properties

This component is closely related to **tNetezzaConnection** and **tNetezzaRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Netezza	
Function	**tNetezzaCommit** validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tNetezzaConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tNetezzaCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaConnection** and **tNetezzaRollback**.	
Limitation	n/a	

Related scenario

This component is closely related to **tNetezzaConnection** and **tNetezzaRollback**. It usually does not make much sense to use one of these without using a **tNetezzaConnection** component to open a connection for the current transaction.

For **tNetezzaCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tNetezzaConnection

tNetezzaConnection Properties

This component is closely related to **tNetezzaCommit** and **tNetezzaRollback**. It usually does not make much sense to use one of these without using a **tNetezzaConnection** component to open a connection for the current transaction.

Component family	Databases/Netezza	
Function	**tNetezzaConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Additional JDBC Parameters	Specify additional connection properties for the DB connection you are creating.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaCommit** and **tNetezzaRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a **tNetezzaConnection** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tNetezzaInput

tNetezzaInput properties

Component family	Databases/Netezza	
Function	**tNetezzaInput** reads a database and extracts fields based on a query.	
Purpose	**tNetezzaInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a **tNetezzaConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.

Usage	This component covers all possible SQL queries for Netezza databases.
Limitiation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenarios

Related scenarios for **tNetezzaInput** are:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tNetezzaNzLoad

This component invokes Netezza's nzload utility to insert records into a Netezza database. This component can be used either in standalone mode, loading from an existing data file; or connected to an input row to load data from the connected component.

tNetezzaNzLoad properties

Component family	Databases/Netezza	
Function	**tNetezzaNzLoad** inserts data into a Netezza database table using Netezza's nzload utility.	
Purpose	To bulk load data into a Netezza table either from an existing data file, an input flow, or directly from a data flow in streaming mode through a named-pipe.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the Netezza database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table into which the data is to be inserted.
	Action on table	On the table defined, you can perform one of the following operations before loading the data: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear table**: The table content is deleted before the data is loaded. **Truncate table**: executes a truncate statement prior to loading the data to clear the entire content of the table.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Data file	Full path to the data file to be used. If this component is used on its own (not connected to another component with input flow) then this is the name of an existing data file to be loaded into the database. If it is connected, with an input flow to another component; this is the name of the file to be generated and written with the incoming data to later be used with nzload to load into the database.
	Use named-pipe	Select this check box to use a named-pipe instead of a data file. This option can only be used when the component is connected with an input flow to another component. When the check box is selected, no data file is generated and the data is transferred to nzload through

		a named-pipe. This option greatly improves performance in both Linux and Windows. This component on named-pipe mode uses a JNI interface to create and write to a named-pipe on any Windows platform. Therefore the path to the associated JNI DLL must be configured inside the java library path. The component comes with two DLLs for both 32 and 64 bit operating systems that are automatically provided in the Studio with the component.
	Named-pipe name	Specify a name for the named-pipe to be used. Ensure that the name entered is valid.
Advanced settings	*Use existing control file*	Select this check box to provide a control file to be used with the nzload utility instead of specifying all the options explicitly in the component. When this check box is selected, **Data file** and the other nzload related options no longer apply. Please refer to Netezza's nzload manual for details on creating a control file.
	Control file	Enter the path to the control file to be used, between double quotation marks, or click **[...]** and browse to the control file. This option is passed on to the nzload utility via the *-cf* argument.
	Field separator	Character, string or regular expression used to separate fields. *This is nzload's delim argument. If you do not use the **Wrap quotes around fields** option, you must make sure that the delimiter is not included in the data that's inserted to the database. The default value is \t or TAB. To improve performance, use the default value.*
	Wrap quotes around fields	This option is only applied to columns of *String*, *Byte*, *Byte[]*, *Char*, and *Object* types. Select either: **None**: do not wrap column values in quotation marks. **Single quote**: wrap column values in single quotation marks. **Double quote**: wrap column values in double quotation marks. *If using the **Single quote** or **Double quote**option, it is necessary to use \ as the **Escape char**.*
	Advanced options	Set the nzload arguments in the corresponding table. Click **[+]** as many times as required to add arguments to the table. Click the **Parameter** field and choose among the arguments from the list. Then click the corresponding **Value** field and enter a value between quotation marks.
Parameter	*-lf*	Name of the log file to generate. The logs will be appended if the log file already exists. If the parameter is not specified, the default name for the log file is '*<table_name>.<db_name>.nzlog*'. And it's generated under the current working directory where the job is running.
	-bf	Name of the bad file to generate. The bad file contains all the records that could not be loaded due to an internal Netezza error. The records will be appended if the bad file already exists. If the parameter is not specified, the default name for the bad file is '*<table_name>.<db_name>.nzbad*'. And it's generated under the current working directory where the job is running.
	-ouputDir	Directory path to where the log and the bad file are generated. If the parameter is not specified the files are generated under the current directory where the job is currently running.
	-logFileSize	Maximum size for the log file. The value is in MB. The default value is 2000 or 2GB. To save hard disk space, specify a smaller amount if your job runs often.
	-compress	Specify this option if the data file is compressed. Valid values are "**TRUE**" or "**FALSE**". Default value if "**FALSE**".

		This option is only valid if this component is used by itself and not connected to another component via an input flow.
	-skipRows <n>	Number of rows to skip from the beginning of the data file. Set the value to "1" if you like to skip the header row from the data file. The default value is "0". This option should only be used if this component is used by itself and not connected to another component via an input flow.
	-maxRows <n>	Maximum number of rows to load from the data file. This option should only be used if this component is used by itself and not connected to another component via an input flow.
	-maxErrors	Maximum number of error records to allow before terminating the load process. The default value is "1".
	-ignoreZero	Binary zero bytes in the input data will generate errors. Set this option to "NO" to generate error or to "YES" to ignore zero bytes. The default value is "NO".
	-requireQuotes	This option requires all the values to be wrapped in quotes. The default value is "FALSE". This option currently does not work with input flow. Use this option only in standalone mode with an existing file.
	-nullValue <token>	Specify the token to indicate a null value in the data file. The default value is "NULL". To improve slightly performance you can set this value to an empty field by specifying the value as single quotes: "\\'".
	-fillRecord	Treat missing trailing input fields as null. You do not need to specify a value for this option in the value field of the table. This option is not turned on by default, therefore input fields must match exactly all the columns of the table by default. Trailing input fields must be nullable in the database.
	-ctrlChar	Accept control chars in char/varchar fields (must escape NUL, CR and LF). You do not need to specify a value for this option in the value field of the table. This option is turned off by default.
	-ctInString	Accept un-escaped CR in char/varchar fields (LF becomes only end of row). You do not need to specify a value for this option in the value field of the table. This option is turned off by default.
	-truncString	Truncate any string value that exceeds its declared char/varchar storage. You do not need to specify a value for this option in the value field of the table. This option is turned off by default.
	-dateStyle	Specify the date format in which the input data is written in. Valid values are: "YMD", "Y2MD", "DMY", "DMY2", "MDY", "MDY2", "MONDY", "MONDY2". The default value is "YMD". The date format of the column in the component's schema must match the value specified here. For example if you want to load a DATE column, specify the date format in the component schema as "yyyy-MM-dd" and the -dateStyle option as "YMD". For more description on loading date and time fields, see section *Loading DATE, TIME and TIMESTAMP columns*.
	-dateDelim	Delimiter character between date parts. The default value is "-" for all date styles except for "MONDY[2]" which is " " (empty space). The date format of the column in the component's schema must match the value specified here.

	-y2Base	First year expressible using two digit year (Y2) dateStyle.
	-timeStyle	Specify the time format in which the input data is written in. Valid values are: "24HOUR" and "12HOUR". The default value is "24HOUR". For slightly better performance you should keep the default value. The time format of the column in the component's schema must match the value specified here. For example if you want to load a TIME column, specify the date format in the component schema as "HH:mm:ss" and the -timeStyle option as "24HOUR". For more description on loading date and time fields, see section *Loading DATE, TIME and TIMESTAMP columns*.
	-timeDelim	Delimiter character between time parts. The default value is ":". The time format of the column in the component's schema must match the value specified here.
	-timeRoundNanos	Allow but round non-zero digits with smaller than microsecond resolution.
	-boolStyle	Specify the format in which Boolean data is written in the data. The valid values are: "1_0", "T_F", "Y_N", "TRUE_FALSE", "YES". The default value is "1_0". For slightly better performance keep the default value.
	-allowRelay	Allow load to continue after one or more SPU reset or failed over. The default behaviour is not allowed.
	-allowRelay <n>	Specify number of allowable continuation of a load. Default value is "1".
	Encoding	Select the encoding type from the list.
	Specify nzload path	Select this check box to specify the full path to the nzload executable. You must check this option if the nzload path is not specified in the PATH environment variable.
	Full path to nzload executable	Full path to the nzload executable on the machine in use. It is advisable to specify the nzload path in the PATH environment variable instead of selecting this option.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when non particular transformation is required on the data to be loaded ont to the database. This component can be used as a standalone or an output component.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Loading DATE, TIME and TIMESTAMP columns

When this component is used with an input flow, the date format specified inside the component's schema must match the value specified for -dateStyle, -dateDelim, -timeStyle, and -timeDelim options. Please refer to following examples:

DB Type	Schema date format	-dateStyle	-dateDelim	-timeStyle	-timeDelim
DATE	"yyyy-MM-dd"	"YMD"	"-"	n/a	n/a
TIME	"HH:mm:ss"	n/a	n/a	"24HOUR"	":"
TIMESTAMP	"yyyy-MM-dd HH:mm:ss"	"YMD"	"-"	"24HOUR"	":"

Related scenario

For a related use case, see section *Scenario: Inserting data in MySQL database*.

tNetezzaOutput

tNetezzaOutput properties

Component family	Databases/Netezza	
Function	**tNetezzaOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tNetezzaOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the designed Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a **tNetezzaConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected the **Insert**, **Update** or the **Delete** option in the **Action on data** list.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.

	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Netezza database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tNetezzaOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tNetezzaRollback

tNetezzaRollback properties

This component is closely related to **tNetezzaCommit** and **tNetezzaConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Netezza	
Function	**tNetezzaRollback** cancels the transaction committed in the connected DB.	
Purpose	This component avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tNetezzaConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaConnection** and **tNetezzaCommit**.	
Limitation	n/a	

Related scenarios

For **tNetezzaRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tNetezzaRow

tNetezzaRow properties

Component family	Databases/Netezza	
Function	**tNetezzaRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means that the component implements a flow in the job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tNetezzaRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tNetezzaConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Enter the name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario 1: Removing and regenerating a MySQL table index*

tParAccelBulkExec

tParAccelBulkExec Properties

The **tParAccelOutputBulk** and **tParAccelBulkExec** components are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tParAccelOutputBulkExec** component, detailed in a different section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

Component Family	Databases/ParAccel	
Function	**tParAccelBulkExec** performs an Insert action on the data.	
Purpose	**tParAccelBulkExec** is a component which is specifically designed to improve performance when loading data in ParAccel database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box if you use a configured **tParAccelConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Database name.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created.

		Create a table if not exists: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Copy mode*	Select the copy mode you want to use from either: **Basic**: Standard mode, without optimisation. **Parallel**: Allows you to use several internal ParAccel APIs in order to optimise loading speed.
	Filename	Name and path of the file to be processed.
	File Type	Select the file type from the list.
	Field Layout	Select the field layout from the list.
	Field separator	Character, string or regular expression to separate fields.
	Explicit IDs	The ID is already present in the file to be loaded or will be set by the database.
	Remove Quotes	Select this check box to remove quotation marks from the file to be loaded.
	Max. Errors	Type in the maximum number of errors before your Job stops.
	Date Format	Type in the date format to be used.
	Time/Timestamp Format	Enter the date and hour format to be used.
	Additional COPY Options	Enter the specific, customized ParAccel option that you want to use.
	Log file	Browse to or enter the access path to the log file in your directory.
	Logging level	Select the information type you want to record in your log file.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL database queries. It allows you to carry out actions on a table or on the data of a table in a ParAccel database. It enables you to create a reject flow, with a **Row** > **Reject** link filtering the data in error. For a usage example, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a related scenario, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tParAccelClose

tParAccelClose properties

Component family	Databases/ParAccel	
Function	**tParAccelClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tParAccelConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with ParAccel components, especially with **tParAccelConnection** and **tParAccelCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tParAccelCommit

tParAccelCommit Properties

This component is closely related to **tParAccelConnection** and **tParAccelRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/ParAccel	
Function	Validates the data processed through the job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tParAccelConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tParAccelCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with ParAccel components, especially with **tParAccelConnection** and **tParAccelRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tParAccelConnection** and **tParAccelRollback**. It usually does not make much sense to use one of these without using a **tParAccelConnection** component to open a connection for the current transaction.

For **tParAccelCommit** related scenario, see section *tMysqlConnection*

tParAccelConnection

tParAccelConnection Properties

This component is closely related to **tParAccelCommit** and **tParAccelRollback**. It usually doesn't make much sense to use one of these without using a **tParAccelConnection** component to open a connection for the current transaction.

Component family	Databases/ParAccel	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with ParAccel components, especially with **tParAccelCommit** and **tParAccelRollback** components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

This component is closely related to **tParAccelCommit** and **tParAccelRollback**. It usually does not make much sense to use one of these without using a **tParAccelConnection** component to open a connection for the current transaction.

For **tParAccelConnection** related scenario, see section *tMysqlConnection*

tParAccelInput

tParAccelInput properties

Component family	Databases/ ParAccel	
Function	**tParAccelInput** reads a database and extracts fields based on a query.	
Purpose	**tParAccelInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tParAccelConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to sequence the fields properly in order to match the schema definition.
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Guess schema	Click the **Guess schema** button to retrieve the table schema.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.

	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for ParAccel databases. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

tParAccelOutput

tParAccelOutput Properties

Component Family	Databases/ParAccel	
Function	**tParAccelOutput** writes, updates, modifies or deletes the data in a database.	
Purpose	**tParAccelOutput** executes the action defined on the table and/or on the data of a table, according to the input flow form the previous component.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box if you use a configured **tParAccelConnection**. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Database name.
	Schema	Exact name of the schema.
	Username et *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL database queries. It allows you to carry out actions on a table or on the data of a table in a ParAccel database. It enables you to create a reject flow, with a **Row** > **Rejects** link filtering the data in error. For a usage example, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a related scenario, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tParAccelOutputBulk

tParAccelOutputBulk properties

The **tParAccelOutputBulk** and **tParAccelBulkExec** components are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tParAccelOutputBulkExec** component, detailed in a different section. The advantage of using two separate steps is that the data can be transformed before it is loaded in the database.

Component family	Databases/ParAccel	
Function	Writes a file with columns based on the defined delimiter and the ParAccel standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the ParAccel database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header.
	Encoding	Select the encoding type from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tParAccelBulkExec** component. Used together they offer gains in performance while feeding a ParAccel database.	

Related scenarios

For use cases in relation with **tParAccelOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tParAccelOutputBulkExec

tParAccelOutputBulkExec Properties

The **tParAccelOutputBulk** and **tParAccelBulkExec** components are generally used together in a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in **tParAccelOutputBulkExec**.

Component Family	Databases/ParAccel	
Function	**tParAccelOutputBulkExec** performs an Insert action on the data.	
Purpose	**tParAccelOutputBulkExec** is a component which is specifically designed to improve performance when loading data in ParAccel database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Database name.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Copy mode	Select the copy mode you want to use from either: **Basic**: Standard mode, without optimisation. **Parallel**: Allows you to use several internal ParAccel APIs in order to optimise loading speed.
	Filename	Name and path of the file to be processed.
Advanced settings	*File Type*	Select the file type from the list.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.

	Fields terminated by	Character, string or regular expression to separate fields.
	Append	Select this check box to add the new rows at the end of the file.
	Explicit IDs	The ID is already present in the file to be loaded or will be set by the database.
	Remove Quotes	Select this check box to remove quotation marks from the file to be loaded.
	Max. Errors	Type in the maximum number of errors before your Job stops.
	Date Format	Type in the date format to be used.
	Time/Timestamp Format	Enter the date and hour format to be used.
	Additional COPY Options	Enter the specific, customized ParAccel option that you want to use.
	Log file	Browse to or enter the access path to the log file in your directory.
	Logging level	Select the information type you want to record in your log file.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL database queries. It allows you to carry out actions on a table or on the data of a table in a ParAccel database. It enables you to create a reject flow, with a **Row** > **Reject** link filtering the data in error. For a usage example, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related scenarios

For a related scenario, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tParAccelRollback

tParAccelRollback properties

This component is closely related to **tParAccelCommit** and **tParAccelConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tParAccelConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with ParAccel components, especially with **tParAccelConnection** and **tParAccelCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tParAccelConnection** and **tParAccelCommit**. It usually doesn't make much sense to use one of them without using a **tParAccelConnection** component to open a connection for the current transaction.

For **tParAccelRollback** related scenario, see section *tMysqlRollback*.

tParAccelRow

tParAccelRow Properties

Component Family	Databases/ParAccel	
Function	**tParAccelRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tParAccelRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tFirebirdConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username et *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder.

	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a related scenario, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tParAccelSCD

tParAccelSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tParAccelSCD*.

tTeradataClose

tTeradataClose properties

Component family	Databases/Teradata	
Function	**tTeradataClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tTeradataConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Teradata components, especially with **tTeradataConnection** and **tTeradataCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tTeradataCommit

tTeradataCommit Properties

This component is closely related to **tTeradataConnection** and **tTeradataRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Teradata	
Function	**tTeradataCommit** validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tTeradataConnection** component in the list if more than one connection are planned for the current job.
	Close connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tTeradataCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Teradata components, especially with **tTeradataConnection** and **tTeradataRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tTeradataConnection** and **tTeradataRollback**. It usually does not make much sense to use one of these without using a **tTeradataConnection** component to open a connection for the current transaction.

For **tTeradataCommit** related scenario, see section *tVerticaConnection*

tTeradataConnection

tTeradataConnection Properties

This component is closely related to **tTeradataCommit** and **tTeradataRollback**. It usually doesn't make much sense to use one of these without using a **tTeradataConnection** component to open a connection for the current transaction.

Component family	Databases/Teradata	
Function	**tTeradataConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Additional JDBC parameters	Specify additional connection properties in the existing DB connection, to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters. You can set the encoding parameters through this field.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Utilisation	This component is to be used along with Teradata components, especially with **tTeradataCommit** and **tTeradataRollback** components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

This component is closely related to **tTeradataCommit** and **tTeradataRollback**. It usually doesn’t make much sense to use one of these without using a **tTeradataConnection** component to open a connection for the current transaction.

For **tTeradataConnection** related scenario, see section *tMysqlConnection*.

tTeradataFastExport

tTeradataFastExport Properties

Component Family	Databases/Teradata	
Function	**tTeradataFastExport** exports rapidly voluminous data batches from a Teradata table or view.	
Purpose	**tTeradataFastExport** exports data batches from a Teradata table to a cutsomer system or to a smaller database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Execution platform	Select the Operating System type you use.
	Host	Server name or IP.
	Database name	Database name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use query	Select this check box to enter the SQL statement in the **Query** box.
	Log database	Log database name.
	Log table	Log table name.
	Script generated folder	Browse your directory and select the destination of the file which will be created.
	Exported file	Name and path to the file which will be created.
	Field separator	Character, string or regular expression to separate fields.
	Error file	Browse your directory and select the destination of the file where the error messages will be recorded.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	

Related scenario

No scenario is available for this component yet.

tTeradataFastLoad

tTeradataFastLoad Properties

Component Family	Databases/Teradata	
Function	**tTeradataFastLoad** reads a database and extracts fields using queries.	
Purpose	**tTeradataFastLoad** executes a database query according to a strict order which must be the same as the one in the schema. The retrieve list of fields is then transfered to the next component, using a connexion flow (**Main row**).	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Database	Database name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Execute Batch every	Number of rows per batch to be loaded.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

No scenario is available for this component yet.

tTeradataFastLoadUtility

tTeradataFastLoadUtility Properties

Component Family	Databases/Teradata	
Function	**tTeradataFastLoadUtility** reads a database and extracts fields using queries.	
Purpose	**tTeradataFastLoadUtility** executes a database query according to a strict order which must be the same as the one in the schema. The retrieve list of fields is then transfered to the next component, using a connexion flow (**Main row**).	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Execution platform	Select the Operating System type you use.
	Host	Host name or IP address of the database server.
	Database name	Database name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Script generated folder	Browse your directory and select the destination of the file which will be created.
	Load file	Browse your directory and select the file from which you want to load data.
	Field separator	Character, string or regular expression to separate fields.
	Error file	Browse your directory and select the destination of the file where the error messages will be recorded.
Advanced settings	*Define character set*	Specify the character encoding you need use for your system.
	Check point	Enter the check point value.
	Error files	Enter the file name where the error messages are stored. By default, the code `ERRORFILES table_ERR1, table_ERR2` is entered, meaning that the two tables *table_ERR1* and *table_ERR2* are used to record the error messages.
	Return fastload error	Select this check box to specify the exit code number to indicate the point at which an error message should display in the console.
	ERRLIMIT	Enter the limit number of errors detected during the loading phase. Processing stops when the limit is reached. The default error limit value is 1000000. For more information, see *Teradata FastLoad Reference* documentation.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenario

For related topic, see section *Scenario: Inserting data into a Teradata database table*.

tTeradataInput

tTeradataInput Properties

Component family	Databases/Teradata	
Function	**tTeradataInput** reads a database and extracts fields based on a query.	
Purpose	**tTeradataInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection, to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Teradata databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tTeradataMultiLoad

tTeradataMultiLoad Properties

Component Family	Databases/Teradata	
Function	**tTeradataMultiLoad** reads a database and extracts fields using queries.	
Purpose	**tTeradataMultiLoad** executes a database query according to a strict order which must be the same as the one in the schema. The retrieve list of fields is then transfered to the next component, using a connexion flow (**Main row**).	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Execution platform	Select the Operating System type you use.
	Host	Host name or IP address of the database server.
	Database name	Database name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Script generated folder	Browse your directory and select the destination of the file which will be created.
	Action to data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s).*
	Where condition in case Delete	Type in a condition, which, once verified, will delete the row.
	Load file	Browse your directory and select the file from which you want to load data.
	Field separator	Character, string or regular expression to separate fields.
	Error file	Browse your directory and select the destination of the file where the error messages will be recorded.
Advanced settings	*Define Log table*	This check box is selected to define a log table you want to use in place of the default one that is the database table you defined

		in **Basic settings**. The syntax required to define the log table is *databasename.logtablename*.
	BEGIN LOAD	This field allows you to define your BEGIN LOAD command to initiate or restart a load task. You can specify the number of sessions to use, the error limit, any other parameters needed to execute the task. For more information, see *Teradata MultiLoad Reference* documentation.
	Return mload error	Select this check box to specify the exit code number to indicate the point at which an error message should display in the console.
	Define character set	Specify the character encoding you need use for your system
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenario

For related topic, see section *Scenario: Inserting data into a Teradata database table*.

tTeradataOutput

tTeradataOutput Properties

Component family	Databases/Teradata	
Function	**tTeradataOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tTeradataOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and ceate**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Create	This is not visible by default, until you choose to create a table from the **Action on table** drop-down list. The table to be created may be: - **SET TABLE**: tables which do not allow to duplicate - **MULTI SET TABLE**: tables allowing duplicate rows.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. This is intended to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters. You can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.

		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Teradata database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tTeradataRollback

tTeradataRollback Properties

This component is closely related to **tTeradataCommit** and **tTeradataConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Teradata	
Function	**tTeradataRollback** cancels the transaction commit in the connected DB.	
Purpose	**tTeradataRollback** avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **TeradataConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Teradata components, especially with **tTeradataConnection** and **tTeradataCommit** components.	
Limitation	n/a	

Related scenario

For **tTeradataRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tTeradataRow

tTeradataRow Properties

Component family	Databases/Teradata	
Function	**tTeradataRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tTeradataRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of the DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. This is intended to allow specific character set support. E.G.: CHARSET=KANJISJIS_OS to get support of Japanese characters.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output* .
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tTeradataTPTUtility

tTeradataTPTUtility Properties

Component Family	Databases/Teradata	
Function	**tTeradataTPTUtility** combines the utilities of **tTeradataFastLoad**, **tTeradataMultiLoad**, **tTeradataTPump**, and **tTeradataFastExport** into one comprehensive utility.	
Purpose	**tTeradataTPTUtility** allows you to insert data to load data into and delete data from any accessible table in the Teradata Database or from any other data stores for which an access operator or an access module exists.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Filename	Browse your directory and select the file to save the output data.
	Append	Select this check box to append the work table to the path set in the **Filename** field.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s).*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **Built-in** or stored remotely in the **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Execution platform	Select the Operating System type you use.
	TDPID	Teradata director program identifier. It can be either the name or the IP address of the Teradata Database system being accessed. If you do not specify a TDPID, the system will use the name of Teradata database as the default TDPID. The customized TDPID can be up to 256 characters and can be a domain server name. For further information about TDPID, see Teradata Parallel Transporter Reference.
	Database name	Fill this field with the name of the Teradata database.

	Load Operator	A consumer operator that functions similarly to **tTeradataFastLoad** to load data from data streams and inserts data into individual rows of a target table in the Teradata database.
	Data Connector	Functions as either a file reader to read from flat files or access modules or a file writer to write to flat files or access modules. For further information about flat file, see Flat file database.
	Job Name	Name of a Teradata Parallel Transporter Job which is defined using Teradata *tbuild* command. If you do not specify a Job name, the default is the user name followed by a hyphen and a generated TPT Job sequence number as follows: `<user name>-<job sequence number>` For further information about Teradata commands, see Teradata Parallel Transporter Reference.
	Layout Name(schema)	A schema for the data to be interchanged.
	Username and *Password*	The Teradata database username and the Teradata database password associated with the username for Teradata database authentication.
	Table	Name of the table to be written into the Teradata database. Note that only one table can be written at a time.
	Script generated folder	Browse your directory and select the destination of the file which will be created.
	Where condition in case Delete	Type in script as a condition, which, once verified, will delete the row.
	Error file	Browse your directory and select the destination of the file where the error messages will be recorded.
Advanced settings	*Row seperator*	Character, string or regular expression to separate rows.
	Field seperator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header to the file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Define Log table	This check box is selected to define a log table you want to use in place of the default one that is the database table you defined in **Basic settings** followed by "`_log`". The syntax required to define the log table is *logtablename*.
	Return mload error	Select this check box to specify the exit code number to indicate the point at which an error message should display in the console. For further information about this error, see Teradata MultiLoad Reference.
	Define character set	Specify the character encoding to be used in your system.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries. For further information about the usage of this component, see Teradata Parallel Transporter Reference.	

Related scenario

For related topic, see section *Scenario: Inserting data into a Teradata database table*.

tTeradataTPump

tTeradataTPump Properties

Component Family	Databases/Teradata	
Function	**tTeradataTPump** reads a database and extracts fields using queries.	
Purpose	**tTeradataTPump** executes a database query according to a strict order which must be the same as the one in the schema. The retrieve list of fields is then transfered to the next component, using a connexion flow (**Main row**).	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Execution platform	Select the Operating System type you use.
	Host	Host name or IP address of the database server.
	Database name	Database name.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Script generated folder	Browse your directory and select the destination of the file which will be created.
	Action to data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s).*
	Where condition in case Delete	Type in a condition, which, once verified, will delete the row.
	Load file	Browse your directory and select the file from which you want to load data.
	Field separator	Character, string or regular expression to separate fields.
	Error file	Browse your directory and select the destination of the file where the error messages will be recorded.
Advanced settings	*Define Log table*	This check box is selected to define a log table you want to use in place of the default one that is the database table you defined

		in **Basic settings**. The syntax required to define the log table is *databasename.logtablename*.
	BEGIN LOAD	This field allows you to define your BEGIN LOAD command to initiate or restart a TPump task. You can specify the number of sessions to use, the error limit and any other parameters needed to execute the task. The default value is: `SESSIONS 8 PACK 600 ARRAYSUPPORT ON CHECKPOINT 60 TENACITY 2 ERRLIMIT 1000.` For more information, see *Teradata Parallel Data Pump Reference* documentation.
	Return tpump error	Select this check box to specify the exit code number to indicate the point at which an error message should display in the console.
	Define character set	Specify the character encoding you need use for your system
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Scenario: Inserting data into a Teradata database table

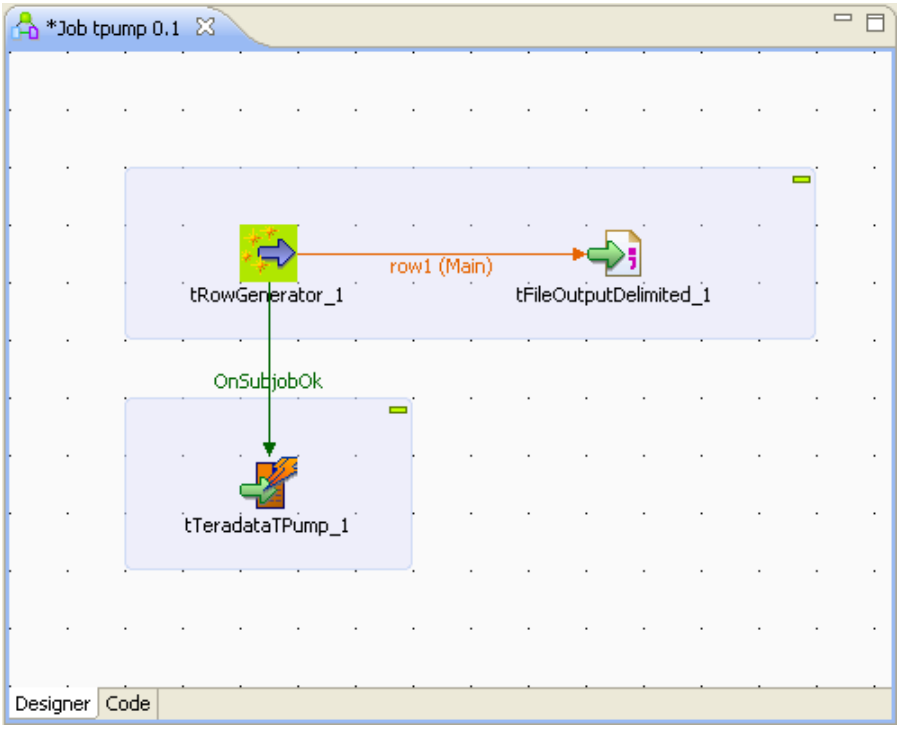
In this scenario, you create a Job using **tTeradataTPump** to insert customer data into a Teradata database table and specify the exit code to be displayed in the event of an exception error.

Three components are used in this Job:

- **tRowGenerator**: generates rows as required using random customer data taken from a list.
- **tFileOutputDelimited**: outputs the customer data into a delimited file.
- **tTeradataTPump**: inserts the customer data into the Teradata database table in the Tpump mode.

Dropping components

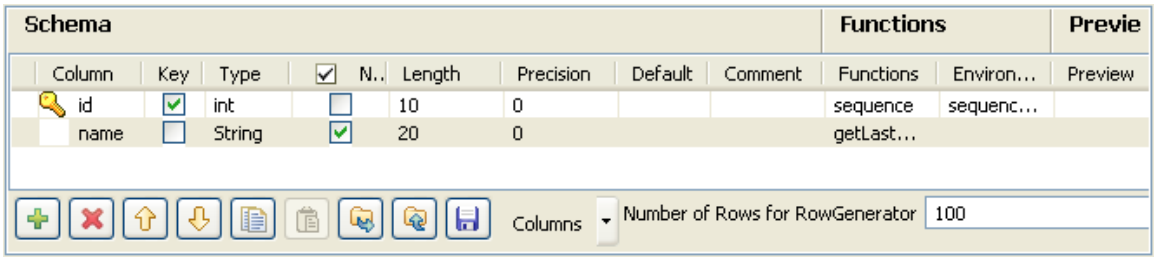
1. Drop the required components: **tRowGenerator**, **tFileOutputDelimited** and **tTeradataTPump** from the **Palette** onto the design workspace.
2. Link **tRowGenerator** to **tFileOutputDelimited** using a **Row** > **Main** connection.
3. Link **tRowGenerator** to **tTeradataTPump** using a **Trigger** > **On SubjobOk** connection.



Configuring the components

1. Double click **tRowGenerator** to open the **tRowGenerator Editor** window.

   In the **tRowGenerator Editor** window, define the data to be generated. For this Job, the schema is composed of two columns: *ID* and *Name*.

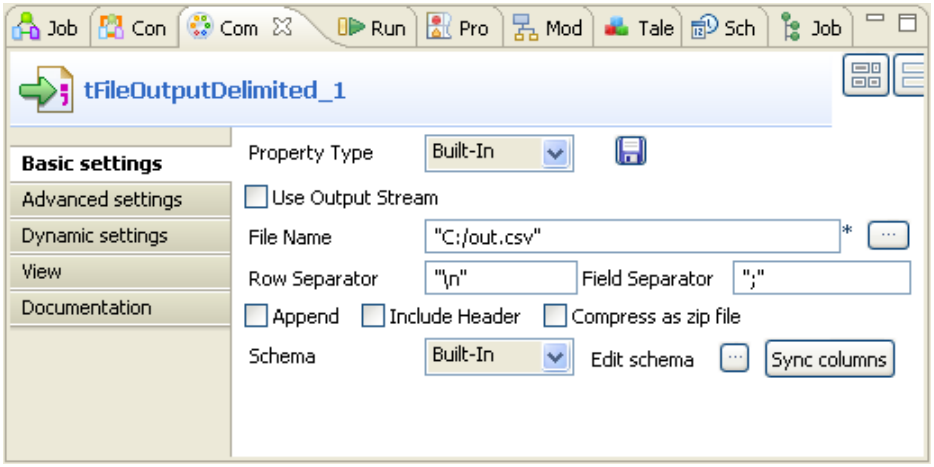


   Enter the **Number of Rows for RowGenerator** to generate.

2. Double click t**FileOutputDelimited** to define its properties in the **Component** view.

3. Next to **File Name**, browse to the output file or enter a name for the output file to be created.

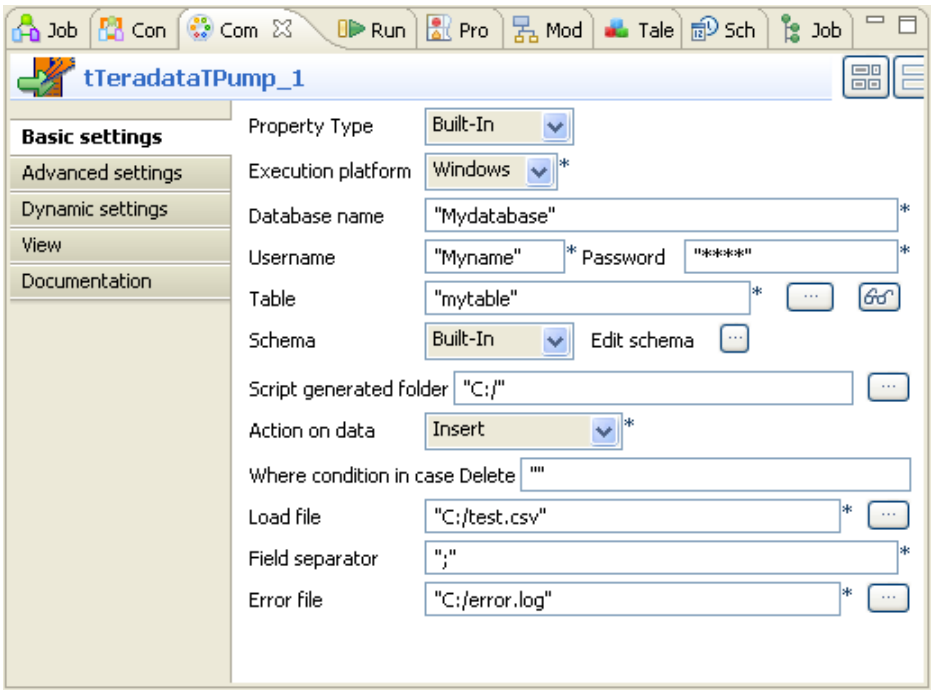
4. Between double quotation marks, enter the delimiters to be used next to **Row Separator** and **Field Separator**.



Click **Edit schema** and check that the schema matches the input schema. If need be, click **Sync Columns**.

5. Double click **tTeradataTPump** to open its **Component** view.

   In the **Basic settings** tab of the **Component** view, define the **tTeradataTPump** parameters. I



6. Enter the **Database name**, **User name** and **Password** in accordance with your database authentication information.

7. Specify the **Table** into which you want to insert the customer data. In this scenario, it is called *mytable*.

8. In the **Script generated folder** field, browse to the folder in which you want to store the script files generated.

9. In the **Load file** field, browse to the file which contains the customer data.

10. In the **Error file** field, browse to the file in which you want to log the error information.

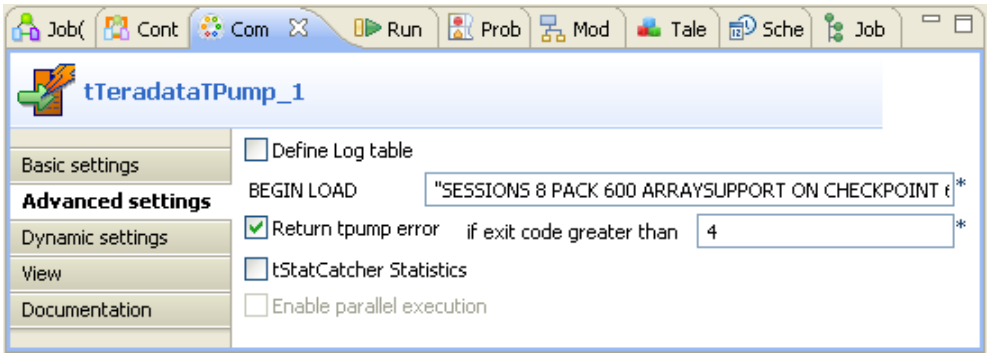
11. In the **Action on data** field, select **Insert**.

Executing the Job

1. Press **F6** to execute the Job.
2. The **Run** view console reads as follows:

```
Running job 'tpump'...
Starting job tpump at 17:01 19/07/2010.

[statistics] connecting to socket on port 3740
[statistics] connected
[statistics] disconnected
Job tpump ended at 17:02 19/07/2010. [exit code=0]
```

3. Double-click the **tTeradataTPump** component to go back to its **Component** view.
4. On the **Advanced settings** tab, select the **Return tpump error** check box and type in the exit code number to indicate the point at which an error message should be displayed in the console. In this example, enter the number *4* and use the default values for the other parameters.



5. Press **F6** to run the Job.
6. The **Run** view console reads as follows:

```
Running job 'tpump'...
Starting job tpump at 17:56 19/07/2010.

[statistics] connecting to socket on port 3549
[statistics] connected
[statistics] disconnected
Exception in component tTeradataTPump_1
java.lang.RuntimeException: TPump returned exit code 12
    at
bug.tpump_0_1.tpump.tTeradataTPump_1Process(tpump.java:307)
    at bug.tpump_0_1.tpump.runJobInTOS(tpump.java:504)
    at bug.tpump_0_1.tpump.main(tpump.java:378)
Job tpump ended at 17:56 19/07/2010. [exit code=0]
```

An exception error occurs and `TPump returned exit code 12` is displayed. If you need to view detailed information about the exception error, you can open the log file stored in the directory you specified in the **Error file** field in the **Basic settings** tab of the **Component** view.

tVectorWiseCommit

tVectorWiseCommit Properties

This component is closely related to **tVectorWiseConnection** and **tVectorWiseRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/VectorWise	
Function	**tVectorWiseCommit** validates the data processed in a Job into the connected DB.	
Purpose	Using a single connection, this component commits a global transaction in one go instead of doing so on every row or every batch. This provides a gain in performance	
Basic settings	*Component list*	Select the **tVectorWiseConnection** component from the list if more than one connection is planned for the current job.
	Close connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tVectorWiseCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is generally used with other VectorWise components, notably **tVectorWiseConnection** and **tVectorWiseRollback.**	
Limitation	n/a	

Related scenario

This component is closely related to **tVectorWiseConnection** and **tVectorWiseRollback**. It usually doesn't make much sense to use one of these without using a **tVectorWiseConnection** component to open a connection for the current transaction.

For a **tVectorWiseCommit** related scenario, see section *tVerticaConnection*.

tVectorWiseConnection

tVectorWiseConnection Properties

This component is closely related to **tVectorWiseCommit** and **tVectorWiseRollback**. It usually doesn't make much sense to use one of these without using a **tVectorWiseConnection** component to open a connection for the current transaction.

Component family	Databases/VectorWise	
Function	**tVectorWiseConnection** opens a connection to a database for a transaction to be carried out.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username et *Password*	Authentication information of the database user.
	Use or register a shared DB Connection	Select this check box to share your connection or retrieve a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name**: set or type in the shared connection name.
Advanced settings	*Auto Commit*	Select this check box to commit a transaction automatically upon completion.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with VectorWise components, particularly **tVectorWiseCommit** and **tVectorWiseRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

This component is closely related to **tVectorWiseCommit** and **tVectorWiseRollback**. It usually doesn't make much sense to use one of these without using a **tVectorWiseConnection** component to open a connection for the current transaction.

For a **tVectorWiseConnection** related scenario, see section *tMysqlConnection*.

tVectorWiseInput

tVectorWiseInput Properties

Component family	Databases/VectorWise	
Function	**tVectorWiseInput** reads a database and extracts fields based on a query.	
Purpose	**tVectorWiseInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tVectorWiseConnection** component. When a Job contains the parent Job and the child Job, the **Component list** only presents the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can deactivate the connection components and use the **Dynamic settings** of the component to specify the connection manually. In this case, enssure that the connection name is unique and distinctive throughout the two Job levels. For further information about **Dynamic settings**, see your studio user guide.
	Server	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the database.
	Username a *Password*	Authentication information of the database user.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be read.
	Query type and *Query*	Enter your DB query, ensuring that the field order matches the order in the schema.
	Guess Query	Click this button to generate a query that corresponds to your table schema in the **Query** field.
	Guess schema	Click this button to retrieve the schema from the table.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Define columns from which to remove leading and trailing whitespace.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibile SQL queries forVertica databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For **tVectorWiseInput** related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tVectorWiseOutput

tVectorWiseOutput Properties

Component family	Databases/VectorWise	
Function	**tVectorWiseOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tVectorWiseOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tVerticaConnection** component. When a Job contains the parent Job and the child Job, the **Component list** only presents the connection components of the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive throughout the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Ensure that the Nullable check box is selected for the corresponding columns in the schema.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Vertica database. It also allows you to create a reject flow using a	

	Row > Rejects link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

For **tVectorWiseOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tVectorWiseRollback

tVectorWiseRollback Properties

This component is closely related to **tVectorWiseCommit** and **tVectorWiseConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/VectorWise	
Function	**tVectorWiseRollback** cancels transactions commited to the DB connected.	
Purpose	This component prevents involuntary commits.	
Basic settings	*Component list*	Select the **tVectorWiseConnection** component from the list if more than one connection is planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Teradata components, especially with **tVectorWiseConnection** and **tVectorWiseCommit** components.	
Limitation	n/a	

Related scenario

For a **tVectorWiseRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tVectorWiseRow

tVectorWiseRow Properties

Component family	Databases/VectorWise	
Function	**tVectorWiseRow** is the specific component for this database query. It executes the SQL query stated in the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tVectorWiseRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write your SQL statements easily.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tVectorWiseConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, the **Component list** only presents the connection components of the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For further information about how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in the query statement manually or build it graphically using the SQLBuilder.
	Guess Query	Click this button to generate a query that corresponds to your table schema in the **Query** field.

	Query	Enter your DB query taking care to sequence the fields properly in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tVerticaBulkExec

tVerticaBulkExec Properties

The **tVerticaOutputBulk** and **tVerticaBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tVerticaOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases/Vertica	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tVerticaBulkExec** offers gains in performance while carrying out the Insert operations to a Mysql database	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tVerticaConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. You have the possibility to rollback the operation.

	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Remote Filename	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Write to ROS (Read Optimized Store)*	Select this check box to store the data in a physical storage area, in order to optimize the reading, as the data is compressed and pre-sorted.
	Exit job if no row was loaded	The Job automatically stops if no row has been loaded.
	Fields terminated by	Character, string or regular expression to separate fields.
	Null string	String displayed to indicate that the value is null.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tVerticaOutputBulk** component. Used together, they can offer gains in performance while feeding a Vertica database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tVerticaClose

tVerticaClose properties

Component family	Databases/Vertica	
Function	**tVerticaClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tVerticaConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Vertica components, especially with **tVerticaConnection** and **tVerticaCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tVerticaCommit

tVerticaCommit Properties

This component is closely related to **tVerticaConnection** and **tVerticaRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Vertica	
Function	**tVerticaConnection** validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tVerticaConnection** component in the list if more than one connection are planned for the current job.
	Close connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tVerticaCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tVerticaConnection** and **tVerticaRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tVerticaConnection** and **tVerticaRollback**. It usually does not make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

For **tVerticaCommit** related scenario, see section *tVerticaConnection*

tVerticaConnection

tVerticaConnection Properties

This component is closely related to **tVerticaCommit** and **tVerticaRollback**. It usually does not make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

Component family	Databases/Vertica	
Function	**tVerticaConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of Vertica you are using from the list.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Utilisation	This component is to be used along with Vertica components, especially with **tVerticaCommit** and **tVerticaRollback** components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

This component is closely related to **tVerticaCommit** and **tVerticaRollback**. It usually does not make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

For **tVerticaConnection** related scenario, see section *tMysqlConnection*.

tVerticaInput

tVerticaInput Properties

Component family	Databases/Vertica	
Function	**tVerticaInput** reads a database and extracts fields based on a query.	
Purpose	**tVerticaInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of Vertica you are using from the list.
	Use an existing connection	Select this check box when using a configured **tVerticaConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query, ensuring that the field order matches the order in the schema.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.

Usage	This component covers all possible SQL queries for Vertica databases.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tVerticaOutput

tVerticaOutput Properties

Component family	Databases/Vertica	
Function	**tVerticaOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tVerticaOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of Vertica you are using from the list.
	Use an existing connection	Select this check box when using a configured **tVerticaConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Action on table	On the table defined, you can perform one of the following operations: **Default**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform:

		Insert: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. **Copy**: Read data from a text file and insert tuples of entries into the WOS (Write Optimized Store) or directly into the ROS (Read Optimized Store). This option is ideal for bulk loading. For further information, see your Vertica SQL Reference Manual. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
Copy parameters This area is available only when the **Action on data** is **Copy**. For further details about the **Copy** parameters, see your Vertica SQL Reference Manual.	*Abort on error*	Select this check box to stop the **Copy** operation on data if a row is rejected and rolls back this operation. Thus no data is loaded.
	Maximum rejects	Type in a number to set the REJECTMAX command used by Vertica, which indicates the upper limit on the number of logical records to be rejected before a load fails. If not specified or if value is 0, an unlimited number of rejections are allowed.
	No commit	Select this check box to prevent the current transaction from committing automatically.
	Exception file	Type in the path to, or browse to the file in which messages are written indicating the input line number and the reason for each rejected data record.
	Exception file node	Type in the node of the exception file. If not specified, operations default to the query's initiator node.
	Rejected data file	Type in the path to, or browse to the file in which to write rejected rows. This file can then be edited to resolve problems and reloaded.
	Rejected data file node	Type in the node of the rejected data file. If not specified, operations default to the query's initiator node.

	Use batch mode	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected, you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected the **Insert**, the **Update**, the **Delete** or the **Copy** option in the **Action on data** field.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box to allow for the Null value in the "SQL WHERE" statement.
	Create projection when create table	Select this check box to create a projection for a table to be created. This check box is available only when you have selected the table creation related option in the **Action on table** field.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Vertica database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tVerticaOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tVerticaOutputBulk

tVerticaOutputBulk Properties

The **tVerticaOutputBulk** and **tVerticaBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tVerticaOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases/Vertica	
Function	**tVerticaBulkOutputExec** writes a file with columns based on the defined delimiter and the Vertica standards.	
Purpose	**tVerticaBulkOutputExec** prepares the file to be used as parameter in the INSERT query to feed the Vertica database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header to the file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	*tStat*Catcher Statistics	Select this check box to collect log data at the component level.
Utilisation	This component is to be used along with **tVerticaBulkExec**. Used together, they offer gains in performance while feeding a Vertica database.	

Related scenarios

For use cases in relation with **tVerticaOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tVerticaOutputBulkExec

tVerticaOutputBulkExec Properties

The **tVerticaOutputBulk** and **tVerticaBulkExec** components are generally used together as parts of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tVerticaOutputBulkExec** component.

Component family	Databases/Vertica	
Function	**tVerticaOutputBulkExec** executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Vertica database.	
Basic settings	*Property Type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of Vertica you are using from the list.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	DB Name	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Append	Select this check box to add the new rows at the end of the file
Advanced settings	*Write to ROS (Read Optimized Store)*	Select this check box to store the data in a physical storage area, in order to optimize the reading, as the data is compressed and pre-sorted.
	Exit job if no row was loaded	The Job automatically stops if no row has been loaded.
	Field Separator	Character, string or regular expression to separate fields.
	Null string	String displayed to indicate that the value is null.

	Include header	Select this check box to include the column header to the file.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tVerticaOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.

tVerticaRollback

tVerticaRollback Properties

This component is closely related to **tVerticaCommit** and **tVerticaConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Vertica	
Function	**tVerticaRollback** cancels the transaction commit in the connected DB.	
Purpose	**tVerticaRollback** avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **VerticaConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tVerticaConnection** and **tVerticaCommit** components.	
Limitation	n/a	

Related scenario

For **tVerticaRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tVerticaRow

tVerticaRow Properties

Component family	Databases/Vertica	
Function	**tVerticaRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tVerticaRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	DB Version	Select the version of Vertica you are using from the list.
	Use an existing connection	Select this check box and click the relevant **tVerticaConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in the query statement manually or build it graphically using the SQLBuilder.
	Query	Enter your DB query taking care to sequence the fields properly in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this check box if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

Databases - other components

This chapter describes connectors that give access to a variety of databases and provide tools for database management. These connectors cover various needs, including: opening connections, reading and writing tables, committing transactions as a whole, as well as performing rollback for error handling. These components can be found in the **Databases** family in the **Palette** in the **Integration** perspective of the **Talend** Studio.

Other types of database connectors, such as connectors for traditional and appliance/DW databases, are documented in *Databases - traditional components* and *Databases - appliance/datawarehouse components*.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tCreateTable

tCreateTable Properties

Component family	Databases	
Function	**tCreateTable** creates, drops and creates and clears the specified table.	
Purpose	This specific component helps create or drop any database table	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	Database Type	Select the DBMS type from the list. The component properties may differ slightly according to the database type selected from the list.
	DB Version	Select the database version in use.
	Table Action	Select the action to be carried out on the database among: **Create table**: when you know already that the table doesn't exist. **Create table if not exists**: when you don't know whether the table is already created or not **Drop table if exits and create**: when you want to drop an existing table and create it again.
	Use an existing connection	Select this check box in case you use a database connection component, for example: **tMysqlConnection** or **tOracleConnection**, etc. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Database name	Name of the database.
	Schema	Name of the schema. (DB2, Greenplum, Informix, Oracle, PostgresPlus, Postgresql, Vertica)
	Username and *Password*	DB user authentication data.
	Table name	Type in between quotes a name for the newly created table.

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
	Additional JDBC Parameters	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. (AS400, MSSQL Server)
	Create projection	Select this check box to create a projection. (Vertica)
Usage	This component offers the flexibility of the database query and covers all possible SQL queries. More scenarios are available for specific database Input components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Database-specific fields:

Access	*Access File*	Name and path of the file to be processed.
Firebird	*Firebird File*	Name and path of the file to be processed.
HSQLDb	*Running Mode*	Select from the list the Server Mode that corresponds to your database setup.
	Use TLS/SSL Sockets	Select this check box to enable the secured mode, if required.
	DB Alias	Name of the database.
Interbase	*Interbase File*	Name and path of the file to be processed.
JavaDb	*Framework Type*	Select from the list a framework for your database.
	Structure type	Select in the list the structure type.
	DB Root Path	Browse to your database root.
Mysql	*Temporary table*	Select this check box if you want to save the created table temporarily.
ODBC	*ODBC Name*	Name of the database.
Oracle	*Connection Type*	Drop-down list of available drivers: **Oracle SID**: Select this connection type to uniquely identify a particular database on a system. **Oracle Service Name**: Select this connection type to use the TNS alias that you give when you connect to the remote database. **Oracle OCI**: Select this connection type to use Oracle Call Interface with a set of C-language software APIs that provide an interface to the Oracle database. **WALLET**: Select this connection type to store credentials in an Oracle wallet.
SQLite	*SQLite File*	Name and path of the file to be processed.

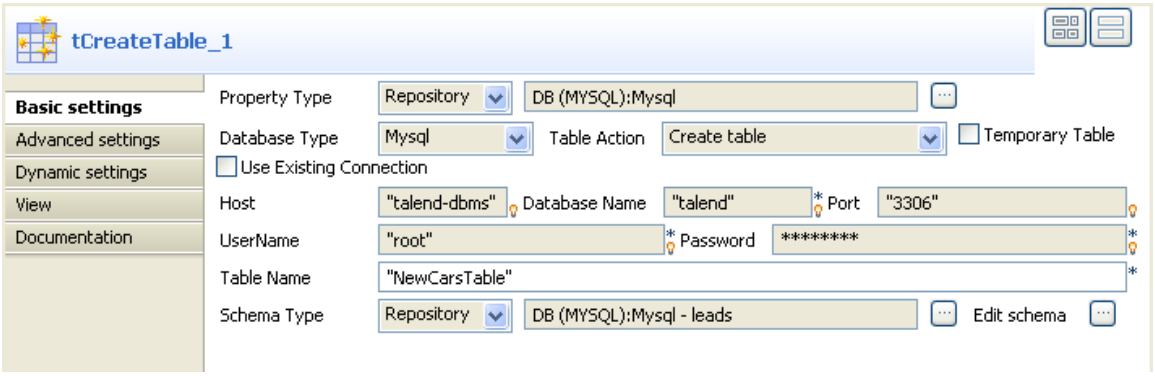
		Related topic: see *Talend Open Studio for Big Data User Guide*.
Teradata	*Create*	Select the table type from the drop-down list. The type may be: - **SET TABLE**: tables which do not allow to duplicate. - **MULTI SET TABLE**: tables allowing duplicate rows

Scenario: Creating new table in a Mysql Database

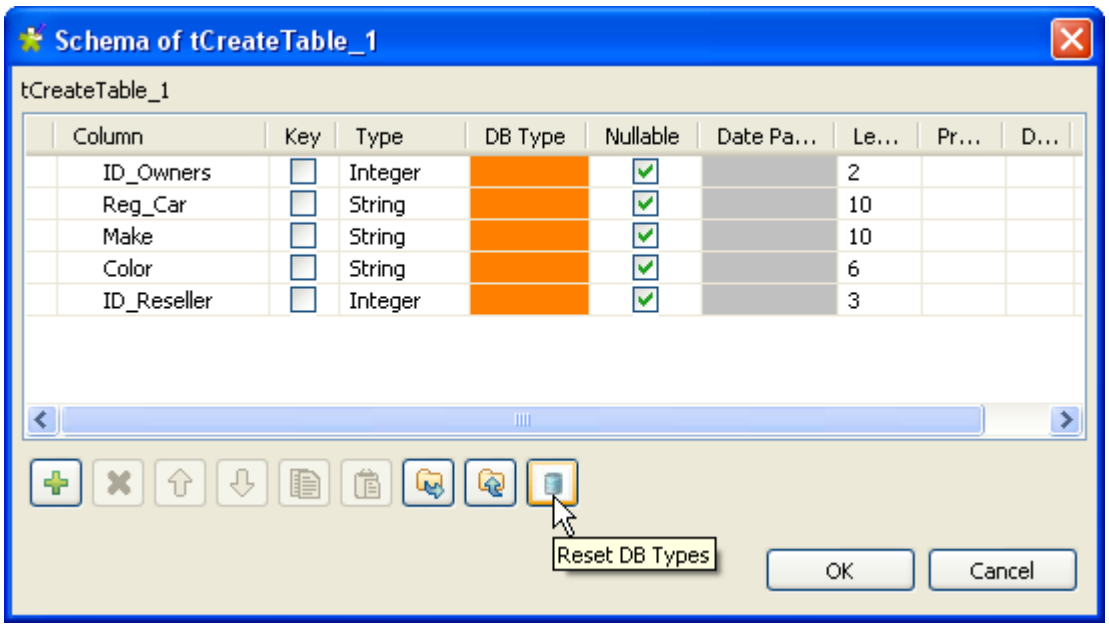
The Job described below aims at creating a table in a database. This Job is composed of a single component.

*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

1. Drop a **tCreateTable** component from the **Databases** family in the **Palette** to the design workspace.
2. In the **Basic settings** view, and from the **Database Type** list, select Mysql for this scenario.



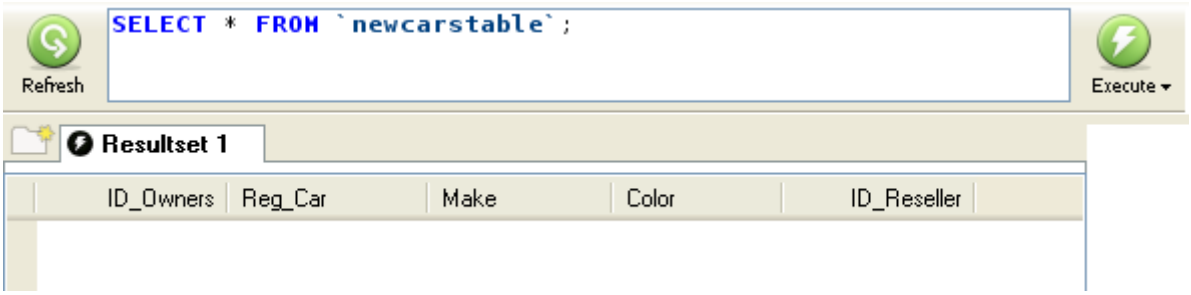
3. From the **Table Action** list, select **Create table**.
4. Select the **Use Existing Connection** check box only if you are using a dedicated DB connection component section *tMysqlConnection*. In this example, we won't use this option.
5. Fill in the DB connection details maually..
6. In the **Table Name** field, fill in a name for the table to be created.
7. Click **Edit Schema** to define the data structure.



8. Click the **Reset DB Types** button in case the DB type column is empty or shows discrepancies (marked in orange). This allows you to map any data type to the relevant DB data type. Then, click **OK** to validate your changes and close the dialog box.

9. Save your Job and press **F6** to execute it.

The table is created empty but with all columns defined in the Schema.



tDBInput

tDBInput properties

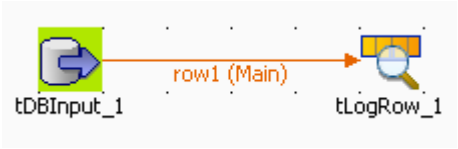
Component family	Databases/DB Generic	
Function	**tDBInput** reads a database and extracts fields based on a query. To use this component, relevant DBMSs' ODBC drivers should be installed and the corresponding ODBC connections should be configured via the database connection configuration wizard.	
Purpose	**tDBInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link. For performance reasons, a specific Input component (e.g.: **tMySQLInput** for MySQL database) should always be preferred to the generic component.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Database	Name of the data source defined via the database connection configuration wizard.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the source table where changes made to data should be captured.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries using a generic ODBC connection.	

Scenario 1: Displaying selected data from DB table

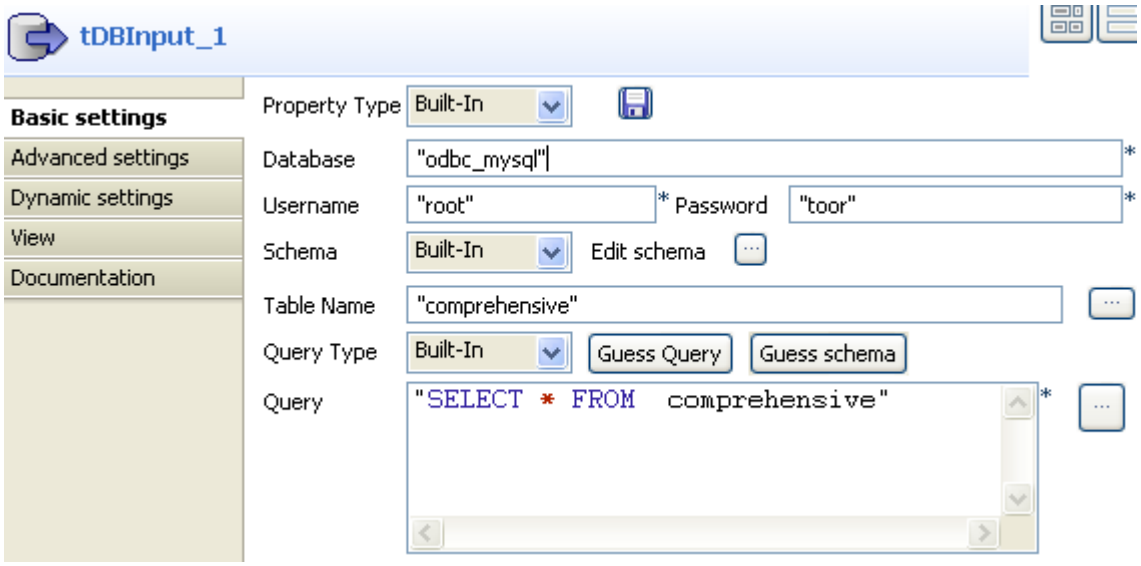
The following scenario creates a two-component Job, reading data from a database using a DB query and outputting delimited data into the standard output (console).

As a prerequisite of this Job, the MySQL ODBC driver must have been installed and the corresponding ODBC connection must have been configured.

1. Drop a **tDBInput** and **tLogRow** component from the **Palette** to the design workspace.
2. Connect the components using **Row** > **Main** link.

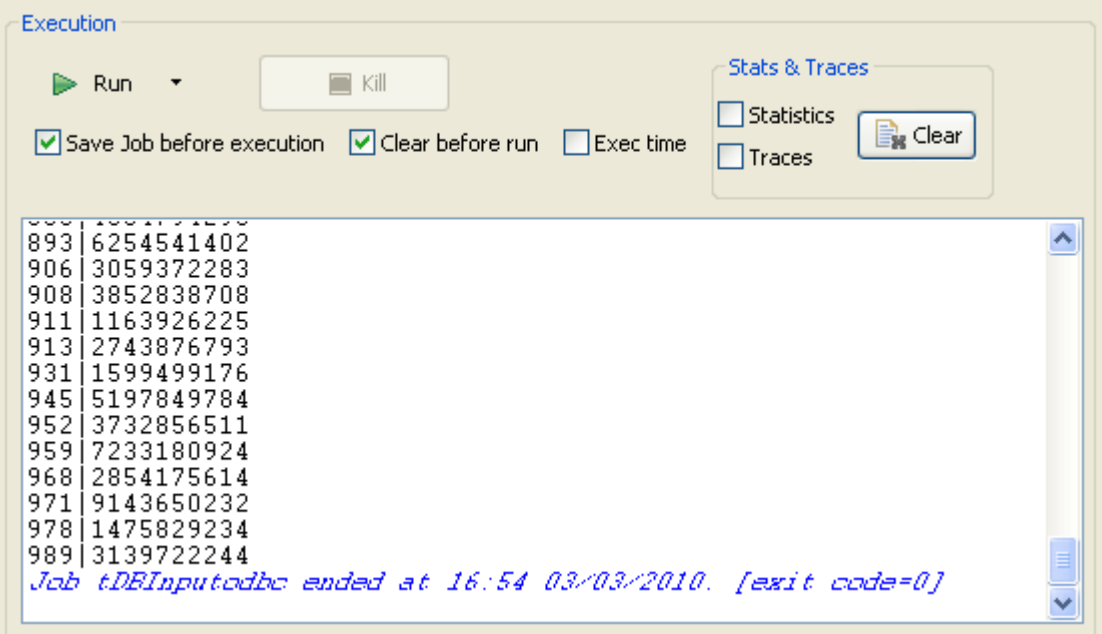


3. Double-click **tDBInput** to open its **Basic settings** view in the **Component** tab.



4. Fill in the database name, the username and password in the corresponding fields.
5. Click **Edit Schema** and create a 2-column description including shop code and sales.
6. Enter the table name in the corresponding field.
7. Type in the query making sure it includes all columns in the same order as defined in the Schema. In this case, as we'll select all columns of the schema, the asterisk symbol makes sense.
8. Click on the second component to define it.
9. Enter the fields separator. In this case, a pipe separator.
10. Now go to the **Run** tab, and click on **Run** to execute the Job.

    The DB is parsed and queried data is extracted from the specified table and passed on to the job log console. You can view the output file straight on the console.

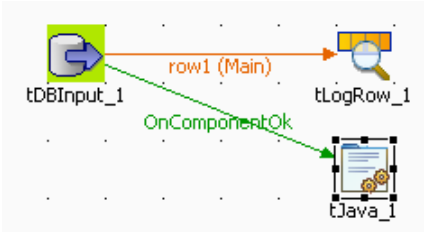


Scenario 2: Using StoreSQLQuery variable

StoreSQLQuery is a variable that can be used to debug a **tDBInput** scenario which does not operate correctly. It allows you to dynamically feed the SQL query set in your **tDBInput** component.

1. Use the same scenario as scenario 1 above and add a third component, **tJava**.

2. Connect **tDBInput** to **tJava** using a trigger connection of the **OnComponentOk** type. In this case, we want the **tDBInput** to run before the **tJava** component.

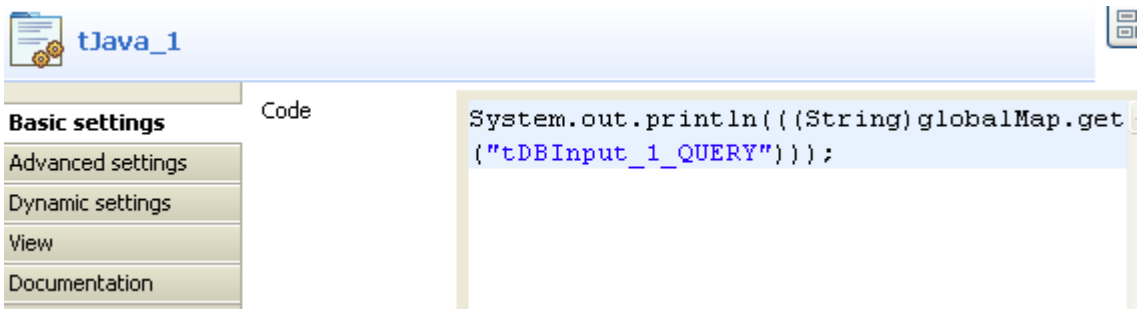


3. Set both **tDBInput** and **tLogRow** component as in **tDBInput** scenario 1.

4. Click anywhere on the design workspace to display the **Contexts** property panel.

5. Create a new parameter called explicitly **StoreSQLQuery**. Enter a default value of 1. This value of 1 means the **StoreSQLQuery** is “true” for a use in the QUERY global variable.

6. Click on the **tJava** component to display the **Component** view. Enter the `System.Out.println("")`command to display the query content, press **Ctrl+Space bar** to access the variable list and select the global variable QUERY.



7. Go to your **Run tab** and execute the Job.

8. The query entered in the **tDBInput** component shows at the end of the job results, on the log:

```
971|9143650232
978|1475829234
989|3139722244
SELECT * FROM   comprehensive
Job tDBInputodbc ended at 10:15 04/03/2010. [exit code=0]
```

tDBOutput

tDBOutput properties

Component family	Databases/DB Generic	
Function	**tDBOutput** writes, updates, makes changes or suppresses entries in a database. To use this component, relevant DBMSs' ODBC drivers should be installed and the corresponding ODBC connections should be configured via the database connection configuration wizard.	
Purpose	**tDBOutput** executes the action defined on the data in a table, based on the flow incoming from the preceding component in the Job. Specific Output component should always be preferred to generic component.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Database	Name of the data source defined via the database connection configuration wizard.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the Use field options check box and then in the **Key in update** column, select the check boxes next to the column names you want to use as a base for the **Update** operation. Do the same in the **Key in delete** column for the **Delete** operation.*
	Clear data in table	Select this check box to delete data in the selected table before any operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.

		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** depending on the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use java.sql.Statement	Select this check box to use the `Statement` object in case the `PreparedStatement` object is not supported by certain DBMSs.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on the data of a table in a database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For a related scenario, see section *Scenario 3: Retrieve data in error with a Reject link*.	

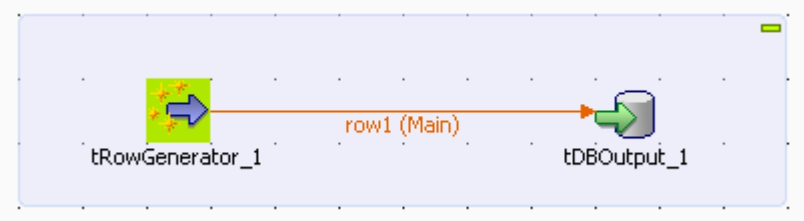
Scenario: Writing a row to a table in the MySql database via an ODBC connection

This scenario clears the data in a table of a MySql database first and then adds a row to it.

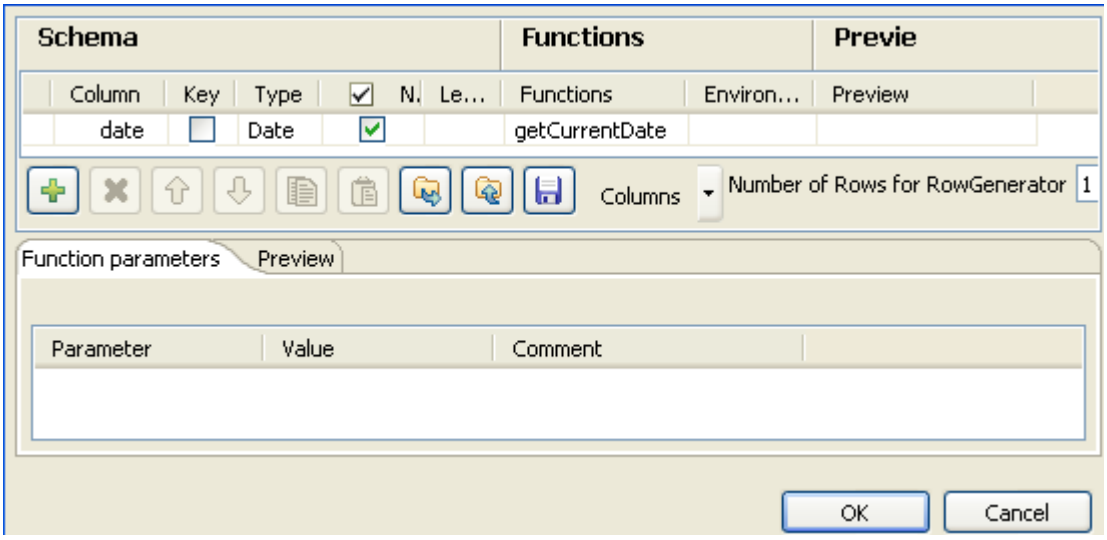
The table, named *Date*, contains one column called *date* with the type being date.

As a prerequisite of this Job, the MySQL ODBC driver must have been installed and the corresponding ODBC connection must have been configured.

1. Drop **tDBOutput** and **tRowGenerator** from the **Palette** to the design workspace.
2. Connect the components using a **Row** > **Main** link.



3. Double-click **tRowGenerator** to open its **Schema editor**.



4. Click the **[+]** button to add a line.

   Enter *date* as the column name.

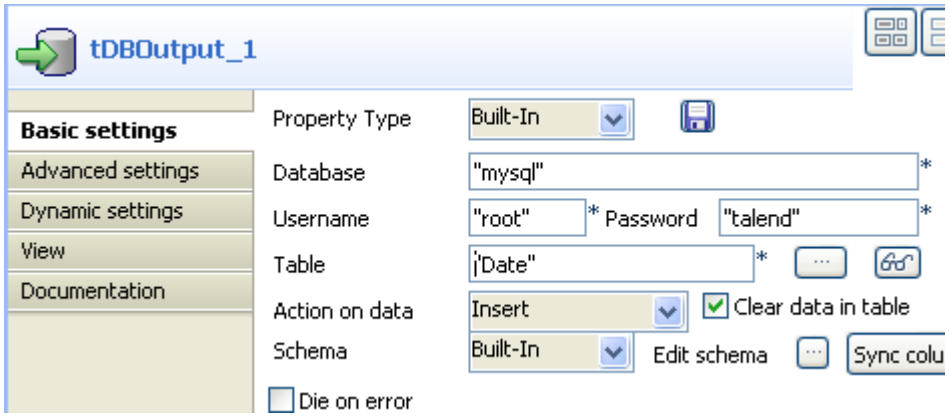
   Select *Date* from the data type list.

   Select *getCurrentDate* from the **Functions** list.

   Enter *1* in the **Number of Rows for RowGenerator** field as only one row will be added to the table.

   Click **OK** to close the editor and propagate the schema changes to **tDBOutput** subsequently.

5. Double-click **tDBOutput** to open its **Basic settings** view in the **Component** tab.



6. In the **Database** field, enter the name of the data source defined during the configuration of the MySql ODBC connection.

   To configure an ODBC connection, click to open the database connection configuration wizard.

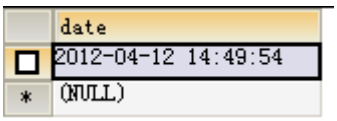
7. In the **Username** and **Password** fields, enter the database authentication credentials.

8. In the **Table** field, enter the table name, *Date* in this example.

9. In the **Action on data** field, select *Insert* to insert a line to the table.

10. Select the check box **Clear data in table** to clear the table before the insertion.

11. Save the Job and press **F6** to run.



As shown above, the table now has only one line about the current date and time.

tDBSQLRow

tDBSQLRow properties

Component family	Databases/DB Generic	
Function	**tDBSQLRow** is the generic component for database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it does not provide output. For performance reasons, specific DB component should always be preferred to the generic component.	
Purpose	Depending on the nature of the query and the database, **tDBSQLRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements. To use this component, relevant DBMSs' ODBC drivers should be installed and the corresponding ODBC connections should be configured via the database connection configuration wizard.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Datasource	Name of the data source defined via the database connection configuration wizard.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the source table where changes made to data should be captured.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Use PreparedStatement	Select this check box if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value.

		This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries. Note that the relevant DBRow component should be preferred according to your DBMSs. Most of the DBMSs have their specific DBRow components.	

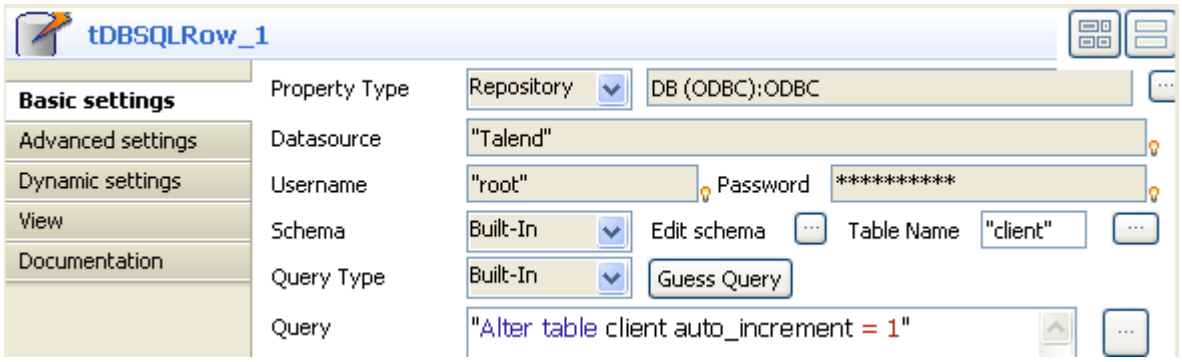
Scenario: Resetting a DB auto-increment

This scenario describes a single component Job which aims at re-initializing the DB auto-increment to 1. This job has no output and is generally to be used before running a script.

As a prerequisite of this Job, the relevant DBMS's ODBC driver must have been installed and the corresponding ODBC connection must have been configured.

1. Drag and drop a **tDBSQLRow** component from the **Palette** to the design workspace.

2. Double-click **tDBSQLRow** to open its **Basic settings** view.



*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

3. Fill in the DB connection details in the corresponding fields.

4. The **Schema** is built-in for this Job and it does not really matter in this example as the action is made on the table auto-increment and not on data.

5. The **Query type** is also built-in. Click on the **[...]** button next to the **Query** statement box to launch the SQLbuilder editor, or else type in directly in the statement box:

   Alter table <TableName> auto_increment = 1

6. Press **Ctrl+S** to save the Job and **F6** to run.

   The database autoincrement is reset to 1.

tEXAInput

tEXAInput properties

Component family	Databases/EXA	
Function	**tEXAInput** reads databases and extracts fields using queries.	
Purpose	**tEXAInput** executes queries in databases according to a strict order which must correspond exactly to that defined in the schema. The list of fields retrieved is then transmitted to the following component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No properties stored centrally
	Host name	Database server IP address.
	Port	Listening port number of the DB server
	Schema name	Enter the schema name.
	Username et *Password*	User authentication information.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*
	Table Name	Enter the table name.
	Query type and *Query*	Enter your database query, taking care to ensure that the order of the fields corressponds exactly to that defined in the schema.
	Guess Query	Click this button to generate a query that corresponds to your table schema in the **Query** field.
	Guess schema	Click this button to retrieve the schema from the table.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Trim all the String/Char columns	Select this check box to delete the spaces at the start and end of fields in all of the columns containing strings.
	Trim column	Deletes the spaces from the start and end of fields in the selected columns.
	*tStat*Catcher Statistics	Select this check box to collect the log data and a component level.
Usage	This component covers all possible SQL queries for EXA databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For scenarios in which **tEXAInput** might be used, see the following **tBIInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*

- section *Scenario 2: Using StoreSQLQuery variable*

tEXAOutput

tEXAOutput properties

Component family	Databases/EXA	
Function	**tEXAOutput** writes, updates, modifies or deletes data from databases.	
Purpose	**tEXAOutput** executes the action defined on the table and/or on the table data, depending on the function of the input flow, from the preceding component.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No properties stored centrally.
	Host	Database server IP address.
	Port	Listening port number of the DB serve.
	Schema name	Enter the schema name.
	Username and *Password*	User authentication data.
	Table	Name of the table to be created. You can only create one table at a time.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation.*

	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Use commit control*	Select this box to display the **Commit every** field in which you can define the number of rows to be processed brefore comitting.
	Additional JDBC parameters	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**. You can press **Ctrl+Space** to access a list of predefined global variables.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing
		Name: Enter the name of the column to be modified or inserted.
		SQL expression: Enter the SQL expression to be executed to modify or insert data in the corresponding columns.
		Position : Select **Before**, **Replace** or **After**, depending on the action to be carried out on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, particularly when there are several actions to be carried out on the data.
	Enable debug mode	Select this check box to display each step of the process by which the data is written in the database.
	*tStat*Catcher Statistics	Select this check box to collect the log data at a component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in an EXA database. It also allows you to create a reject flow using a **Row** > **Reject**s link to filter data in error. For a user scenario, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a scenario in which tEXAOutput might be used, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tEXARow

tEXARow properties

Component family	Databases/EXA	
Function	The **tEXARow** component is specific to this type of database. It executes SQL queries on specified databases. The Row suffix indicates that it is used to channel a flow in a Job although it does not produce any output data.	
Purpose	Depending on the nature of the query and the database, **tEXARow** acts on the actual structure of the database, or indeed the data, although without modifying them.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No properties stored centrally.
	Host	Database server IP address.
	Port	Listening port number of the DB server.
	Schema name	Enter the schema name.
	Username and *Password*	User authentication information.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Enter the query manually or with the help of the SQLBuilder.
	Guess Query	Click the **Guess Query** button to generate the query that corresponds to the table schema in the **Query** field.
	Query	Enter your query, taking care to ensure that the field order matches that defined in the schema.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Propagate QUERY's recordset	Select this check box to insert the query results in one of the flow columns. Select the particular column from the **use column** list.
	Commit every	Number of rows to be included in the batch before the data is written. This option guarantees the quality of the transaction (although there is no rollback option) and improves performance.
	tStatCatcher Statistics	Select this check box to collect the log data at a component level.
Usage	This component offers query flexibility as it covers all possible SQL query requirements.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a scenario in which **tEXARow** might be used, see:

- section *Scenario: Resetting a DB auto-increment*
- section *Scenario 1: Removing and regenerating a MySQL table index*

tEXistConnection

tEXistConnection properties

This component is closely related to **tEXistGet** and **tEXistPut**. Once you have set the connection properties in this component, you have the option of reusing the connection without having to set the properties again for each **tEXist** component used in the Job.

Component family	Databases/eXist	
Function	**tEXistConnection** opens a connection to an **eXist** database in order that a transaction may be carried out.	
Purpose	Opens a connection to an **eXist** database in order that a transaction may be carried out.	
Basic settings	*URI*	URI of the database you want to connect to.
	Collection	Enter the path to the collection of interest on the database server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and Password	User authentication information.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with the other **tEXist** components such as **tEXistGet** and **tEXistPut**. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

This component is closely related to **tEXistGet** and **tEXistPut**. It usually does not make much sense to use one of these without using a **tEXistConnection** component to open a connection for the current transaction.

For **tEXistConnection** related scenario, see section *tMysqlConnection*

tEXistDelete

tEXistDelete properties

Component family	Databases/eXist	
Function	This component deletes resources from an eXist database.	
Purpose	**tEXistDelete** deletes specified resources from remote eXist databases.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter the path to the collection of interest on the database server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	User authentication information.
	Target Type	Either **Resource**, **Collection**, or **All**.
	Files	Click the plus button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as a single component sub-job but can also be used as an output or end object. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your	

	studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tEXistGet

tEXistGet properties

Component family	Databases/eXist	
Function	This component retrieves resources from a remote eXist DB server.	
Purpose	**tEXistGet** downloads selected resources from a remote DB server to a defined local directory.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter the path to the collection of interest on the database server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	User authentication information.
	Local directory	Path to the file's destination location.
	Files	Click the plus button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as a single component sub-job but can also be used as an output or end object. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your	

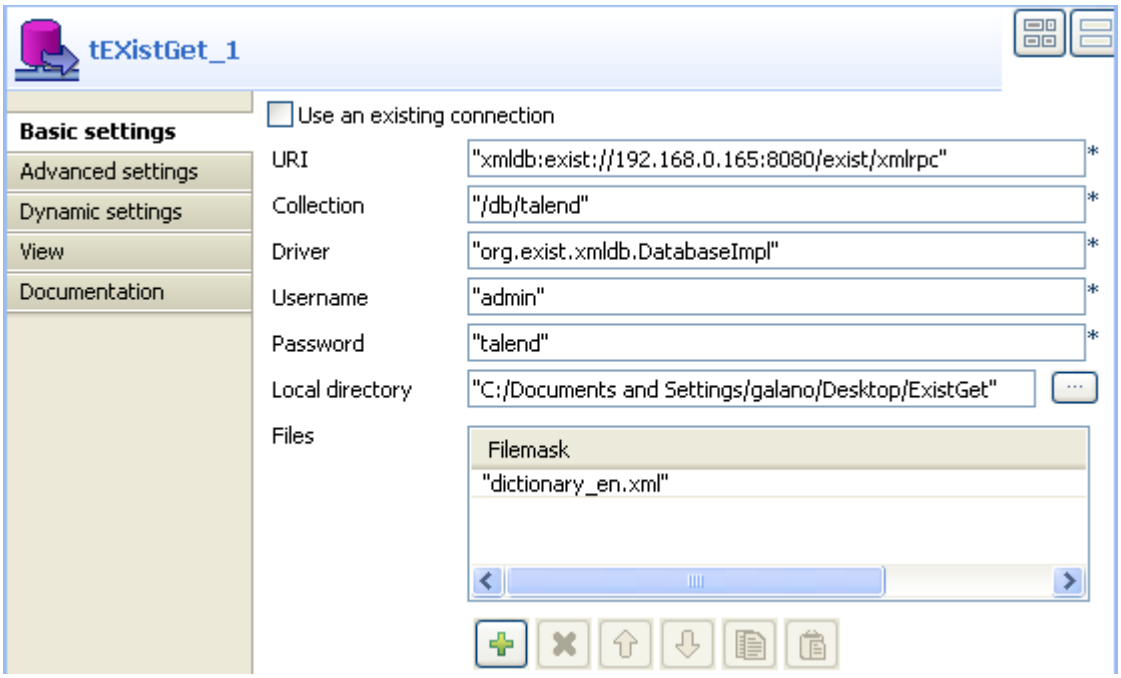
	studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Scenario: Retrieve resources from a remote eXist DB server

This is a single-component Job that retrieves data from a remote eXist DB server and download the data to a defined local directory.

This simple Job requires one component: **tEXistGet**.

1. Drop the **tEXistGet** component from the **Palette** into the design workspace.
2. Double-click the **tEXistGet** component to open the Component view and define the properties in its **Basic settings** view.



3. Fill in the **URI** field with the URI of the eXist database you want to connect to.

   In this scenario, the URI is *xmldb:exist://192.168.0.165:8080/exist/xmlrpc*. Note that the URI used in this use case is for demonstration purpose only and is not an active address.

4. Fill in the **Collection** field with the path to the collection of interest on the database server, */db/talend* in this scenario.
5. Fill in the **Driver** field with the driver for the XML database, *org.exist.xmldb.DatabaseImpl* in this scenario.
6. Fill in the **Username** and **Password** fields by typing in *admin* and *talend* respectively in this scenario.

7. Click the three-dot button next to the **Local directory** field to set a path for saving the XML file downloaded from the remote database server.

   In this scenario, set the path to your desktop, for example *C:/Documents and Settings/galano/Desktop/ExistGet*.

8. In the **Files** field, click the plus button to add a new line in the **Filemask** area, and fill it with a complete file name to retrieve data from a particular file on the server, or a filemask to retrieve data from a set of files. In this scenario, fill in *dictionary_en.xml*.

9. Save your Job and press **F6** to execute it.

```
<!-- generated by ToXgene Version 1.1a in Wed Jun 23 12:16:00 EDT 2004 -->
<dictionary>
    <e id="E1">
        <hwg>
            <hw>planks</hw>
            <pr>!gzkn+J|i@u</pr>
            <hw>gaul</hw>
            <pr>Zft^OeKY</pr>
            <pos>adv.</pos>
        </hwg>
        <et>
            <cr>E709</cr>
            <cr>E3</cr>
            <cr>E439</cr>
            <cr>E414</cr>
        </et>
        <ss>
            <s>
                <def>forges doubt ironic, sly dugouts:sly, regular patt

</def>
                <qp>
                    <q>
                        <qd>259</qd>
                        <w>closely brave</w>
                        <loc>VbGf~UiSc[</loc>
```

The XML file *dictionary_en.xml* is retrieved and downloaded to the defined local directory.

tEXistList

tEXistList properties

Component family	Databases/eXist	
Function	This component lists the resources stored on a remote DB server.	
Purpose	**tEXistList** lists the resources stored on a remote database server.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter the path to the collection of interest on the database server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	Server authentication information.
	Files	Click the plus button to add the lines you want to use as filters:. **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions.
	Target Type	Either **Resource**, **Collection** or **All contents**:
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used along with a **tEXistGet**component to retrieve the files listed, for example. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	

Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tEXistPut

tEXistPut properties

Component family	Databases/eXist	
Function	This component uploads resources to a DB server.	
Purpose	**tEXistPut** uploads specified files from a defined local directory to a remote DB server.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter a path to indicate where the resource is to be saved on the server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	User authentication information.
	Local directory	Path to the source location of the file(s).
	Files	Click the plus button to add the lines you want to use as filters:. **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as a single component sub-job but can also be used as an output or end object. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	

Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tEXistXQuery

tEXistXQuery properties

Component family	Databases/eXist	
Function	This component uses local files containing XPath queries to query XML files stored on remote databases.	
Purpose	**tEXistXQuery** queries XML files located on remote databases and outputs the results to an XML file stored locally.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter the path to the XML file location on the database.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	DB server authentication information.
	XQuery Input File	Browse to the local file containing the query to be executed.
	Local Output	Browse to the directory in which the query results should be saved.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as a single component Job but can also be used as part of a more complex Job. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	
Limitation		

	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tEXistXUpdate

tEXistXUpdate properties

Component family	Databases/eXist	
Function	This component processes XML file records and updates the records on the DB server.	
Purpose	**tEXistXUpdate** processes XML file records and updates the existing records on the DB server.	
Basic settings	*Use an existing connection/ Component List*	Select this check box and click the relevant **tEXistConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	URI	URI of the database you want to connect to.
	Collection	Enter the path to the collection and file of interest on the database server.
	Driver	This field is automatically populated with the standard driver. Users can enter a different driver, depending on their needs.
	Username and *Password*	DB server authentication information.
	Update File	Browse to the local file in the local directory to be used to update the records on the database.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as a single component Job but can also be used as part of a more complex Job. eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing. For further information about XQuery, see XQuery. For further information about the XQuery update extension, see XQuery update extension.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your	

	studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tFirebirdClose

tFirebirdClose properties

Component family	Databases/Firebird	
Function	**tFirebirdClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tFirebirdConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Firebird components, especially with **tFirebirdConnection** and **tFirebirdCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tFirebirdCommit

tFirebirdCommit Properties

This component is closely related to **tFirebirdConnection** and **tFirebirdRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Firebird	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tFirebirdConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tFirebirdCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Firebird components, especially with **tFirebirdConnection** and **tFirebirdRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tFirebirdConnection** and **tFirebirdRollback**. It usually doesn't make much sense to use one of these without using a **tFirebirdConnection** component to open a connection for the current transaction.

For **tFirebirdCommit** related scenario, see section *tMysqlConnection*

tFirebirdConnection

tFirebirdConnection properties

This component is closely related to **tFirebirdCommit** and **tFirebirdRollback**. It usually does not make much sense to use one of these without using a **tFirebirdConnection** to open a connection for the current transaction.

Component family	Databases/Firebird	
Function	**tFirebirdConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host name	Database server IP address.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with Firebird components, especially with **tFirebirdCommit** and **tFirebirdRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

This component is closely related to **tFirebirdCommit** and **tFirebirdRollback**. It usually does not make much sense to use one of these without using a **tFirebirdConnection** component to open a connection for the current transaction.

For **tFirebirdConnection** related scenario, see section *tMysqlConnection*

tFirebirdInput

tFirebirdInput properties

Component family	Databases/FireBird	
Function	**tFirebirdInput** reads a database and extracts fields based on a query.	
Purpose	**tFirebirdInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of the DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced Settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for FireBird databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also related topic: section *Scenario: Dynamic context use in MySQL DB insert*.

tFirebirdOutput

tFirebirdOutput properties

Component family	Databases/FireBird	
Function	**tFirebirdOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tFirebirdOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *You must specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the update and delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column name on which you want to base the update operation. Do the same in the **Key in delete** column for the deletion operation.*

	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Make sure the Nullable check box is selected for the corresponding columns in the schema.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Firebird database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tFirebirdRollback

tFirebirdRollback properties

This component is closely related to **tFirebirdCommit** and **tFirebirdConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Firebird	
Function	**tFirebirdRollback** cancels the transaction committed in the connected database.	
Purpose	This component avoids to commit part of a transaction involuntarily..	
Basic settings	*Component list*	Select the **tFirebirdConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Firebird components, especially with **tFirebirdConnection** and **tFirebirdCommit**.	
Limitation	n/a	

Related scenario

For **tFirebirdRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tFirebirdRow

tFirebirdRow properties

Component family	Databases/FireBird	
Function	**tFirebirdRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tFirebirdRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tFirebirdConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tHBaseClose

tHBaseClose component belongs to two component families: Big Data and Databases. For more information about **tHBaseClose**, see section *tHBaseClose*.

tHBaseConnection

tHBaseConnection component belongs to two component families: Big Data and Databases. For more information about **tHBaseConnection**, see section *tHBaseConnection* .

tHBaseInput

tHBaseInput component belongs to two component families: Big Data and Databases. For more information about **tHBaseInput**, see section *tHBaseInput*.

tHBaseOutput

tHBaseOutput component belongs to two component families: Big Data and Databases. For more information about **tHBaseOutput**, see section *tHBaseOutput*.

tHiveClose

tHiveClose component belongs to two component families: Big Data and Databases. For more information about **tHiveClose**, see section *tHiveClose*.

tHiveConnection

tHiveConnection component belongs to two component families: Big Data and Databases. For more information about **tHiveConnection**, see section *tHiveConnection*.

tHiveInput

tHiveInput component belongs to two component families: Big Data and Databases. For more information about **tHiveInput**, see section *tHiveInput* .

tHiveRow

tHiveRow component belongs to two component families: Big Data and Databases. For more information about **tHiveRow**, see section *tHiveRow*.

tHSQLDbInput

tHSQLDbInput properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbInput** reads a database and extracts fields based on a query.	
Purpose	**tHSQLDbInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Running Mode	Select on the list the Server Mode corresponding to your DB setup among the four propositions : **HSQLDb Server**, **HSQLDb WebServer**, **HSQLDb In Process Persistent**, **HSQLDb In Memory**.
	Use TLS/SSL sockets	select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database Alias	Alias name of the database
	Username and *Password*	DB user authentication data.
	DB path	Specify the directory to the database you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode. By default, if the database you specify in this field does not exist, it will be created automatically. If you want to change this default setting, modify the connection parameter set in the **Additional JDBC parameter** field in the **Advanced settings** view
	Db name	Enter the database name that you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode and the **HSQLDb In Memory** running mode.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. When the running mode is **HSQLDb In Process Persistent** , this additional property is set as `ifexists=true` by default, meaning that the database will be automatically created when needed.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for HSQLDb databases.	
Global Variables		**Number of Lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. Query: Indicates the query to be processed. This is available as a **Flow** variable. Returns a string For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*

tHSQLDbOutput

tHSQLDbOutput properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tHSQLDbOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Running Mode	Select on the list the Server Mode corresponding to your DB setupamong the four propositions : **HSQLDb Server**, **HSQLDb WebServer**, **HSQLDb In Process Persistent**, **HSQLDb In Memory**.
	Use TLS/SSL sockets	Select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	DB path	Specify the directory to the database you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode. By default, if the database you specify in this field does not exist, it will be created automatically. If you want to change this default setting, modify the connection parameter set in the **Additional JDBC parameter** field in the **Advanced settings** view
	Db name	Enter the database name that you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode and the **HSQLDb In Memory** running mode.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops.

		Update: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. When the running mode is **HSQLDb In Process Persistent** , this additional property is set as `ifexists=true` by default, meaning that the database will be automatically created when needed. You can press **Ctrl+Space** to access a list of predefined global variables.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a MySQL database. It also allows you to create a reject flow using a	

	Row > Rejects link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Global Variables		**Number of Lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. **NB line Updated**: Indicates the number of lines updated. This is available as an **After** variable. Returns an integer. **NB line Inserted**: Indicates the number of lines inserted. This is available as an **After** variable. Returns an integer. **NB line Deleted**: Indicates the number of lines deleted. This is available as an **After** variable. Returns an integer. **NB line Rejected**: Indicates the number of lines rejected. This is available as an **After** variable. Returns an integer **Query**: Indicates the query to be processed. This is available as a **After** variable. Returns a string For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Reject **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. Incoming links (from one component to another): **Row**: Main; **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tHSQLDbRow

tHSQLDbRow properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tHSQLDbRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Running Mode	Select on the list the Server Mode corresponding to your DB setup among the four propositions : **HSQLDb Server**, **HSQLDb WebServer**, **HSQLDb In Process Persistent**, **HSQLDb In Memory**.
	Use TLS/SSL sockets	Select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database Alias	Name of the database
	Username and *Password*	DB user authentication data.
	DB path	Specify the directory to the database you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode. By default, if the database you specify in this field does not exist, it will be created automatically. If you want to change this default setting, modify the connection parameter set in the **Additional JDBC parameter** field in the **Advanced settings** view
	Database	Enter the database name that you want to connect to. This field is available only to the **HSQLDb In Process Persistent** running mode and the **HSQLDb In Memory** running mode.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.

Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. When the running mode is **HSQLDb In Process Persistent** , this additional property is set as `ifexists=true` by default, meaning that the database will be automatically created when needed.
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Global Variables		**Query**: Indicates the query to be processed. This is available as a **Flow** variable. Returns a string For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Reject; Iterate **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. Incoming links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tInterbaseClose

tInterbaseClose properties

Component family	Databases/Interbase	
Function	**tInterbaseClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tInterbaseConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Interbase components, especially with **tInterbaseConnection** and **tInterbaseCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tInterbaseCommit

tInterbaseCommit Properties

This component is closely related to **tInterbaseConnection** and **tInterbaseRollback**. It usually doesn't make much sense to use JDBC components independently in a transaction.

Component family	Databases/Interbase	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tInterbaseConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tInterbaseCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Interbase components, especially with the **tInterbaseConnection** and **tInterbaseRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tInterbaseConnection** and **tInterbaseRollback**. It usually doesn't make much sense to use JDBC components without using the **tInterbaseConnection** component to open a connection for the current transaction.

For **tInterbaseCommit** related scenario, see section *tMysqlConnection*

tInterbaseConnection

tInterbaseConnection properties

This component is closely related to **tInterbaseCommit** and **tInterbaseRollback**. It usually does not make much sense to use one of these without using a **tInterbaseConnection** to open a connection for the current transaction.

Component family	Databases/Interbase	
Function	**tInterbaseConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host name	Database server IP address.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with Interbase components, especially with **tInterbaseCommit** and **tInterbaseRollback**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

This component is closely related to **tInterbaseCommit** and **tInterbaseRollback**. It usually does not make much sense to use one of these without using a **tInterbaseConnection** component to open a connection for the current transaction.

For **tInterbaseConnection** related scenario, see section *tMysqlConnection*

tInterbaseInput

tInterbaseInput properties

Component family	Databases/Interbase	
Function	**tInterbaseInput** reads a database and extracts fields based on a query.	
Purpose	**tInterbaseInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Interbase databases.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also the related topic in **tContextLoad**: section *Scenario: Dynamic context use in MySQL DB insert*.

tInterbaseOutput

tInterbaseOutput properties

Component family	Databases/Interbase	
Function	**tInterbaseOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tInterbaseOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*

	Clear data in table	Wipes out data from the selected table before action.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Interbase database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tInterbaseRollback

tInterbaseRollback properties

This component is closely related to **tInterbaseCommit** and **tInterbaseConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Interbase	
Function	**tInterbaseRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tInterbaseConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Interbase components, especially with **tInterbaseConnection** and **tInterbaseCommit**.	
Limitation	n/a	

Related scenarios

For **tInterbaseRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tInterbaseRow

tInterbaseRow properties

Component family	Databases/Interbase	
Function	**tInterbaseRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tInterbaseRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tInterbaseConnection** component on the **Component** list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related scenarios, see:

- section *Scenario 3: Combining two flows for selective output*
- For **tDBSQLRow** related scenario: see section *Scenario: Resetting a DB auto-increment*
- For **tMySQLRow** related scenario: see section *Scenario 1: Removing and regenerating a MySQL table index*.

tJavaDBInput

tJavaDBInput properties

Component family	Databases/JavaDB	
Function	**tJavaDBInput** reads a database and extracts fields based on a query.	
Purpose	**tJavaDBInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL database queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see the **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

See also the related topic in **tContextLoad**: section *Scenario: Dynamic context use in MySQL DB insert*.

tJavaDBOutput

tJavaDBOutput properties

Component family	Databases/JavaDB	
Function	**tJavaDBOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tJavaDBOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*

	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a Java database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For an example of **tMysqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tJavaDBRow

tJavaDBRow properties

Component family	Databases/JavaDB	
Function	**tJavaDBRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tJavaDBRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.

	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tJDBCColumnList

tJDBCColumnList Properties

Component family	Databases/JDBC	
Function	Iterates on all columns of a given table through a defined JDBC connection.	
Purpose	Lists all column names of a given JDBC table.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Table name	Enter the name of the tabe.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection**.	
Limitation	n/a	

Related scenario

For **tJDBCColumnList** related scenario, see section *Scenario: Iterating on a DB table and listing its column names*.

tJDBCClose

tJDBCClose properties

Component family	Databases/JDBC	
Function	**tJDBCClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection** and **tJDBCCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tJDBCCommit

tJDBCCommit Properties

This component is closely related to **tJDBCConnection** and **tJDBCRollback**. It usually doesn't make much sense to use JDBC components independently in a transaction.

Component family	Databases/JDBC	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tJDBCCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with JDBC components, especially with the **tJDBCConnection** and **tJDBCRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tJDBCConnection** and **tJDBCRollback**. It usually doesn't make much sense to use JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCCommit** related scenario, see section *tMysqlConnection*

tJDBCConnection

tJDBCConnection Properties

This component is closely related to **tJDBCCommit** and **tJDBCRollback**. It usually doesn't make much sense to use one of JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

Component family	Databases/JDBC	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings		
	JDBC URL	Enter the JDBC URL to connect to the desired DB. For example, enter: *jdbc:mysql://IP address/database name* to connect to a mysql database.
	Driver JAR	Click the plus button under the table to add lines of the count of your need for the purpose of loading several JARs. Then on each line, click the three dot button to open the **Select Module** wizard from which you can select a driver JAR of your interest for each line.
	Driver Class	Enter the driver class related o your connection. For example, enter *com.mysql.jdbc.Driver* as a driver class to connect to a mysql database.
	Username and *Password*	Enter your DB authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime.
Advanced settings	*Use Auto-Commit*	Select this check box to display the **Auto Commit** check box. Select it to activate auto commit mode. Once you clear the **Use Auto-Commit** check box, the auto-commit statement will be removed from the codes.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with JDBC components, especially with the **tJDBCCommit** and **tJDBCRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tJDBCCommit** and **tJDBCRollback**. It usually doesn't make much sense to use one of JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCConnection** related scenario, see section *tMysqlConnection*

tJDBCInput

tJDBCInput properties

Component family	Databases/JDBC	
Function	**tJDBCInput** reads any database using a JDBC API connection and extracts fields based on a query.	
Purpose	**tJDBCInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	JDBC URL	Type in the database location path
	Driver JAR	Click the plus button under the table to add lines of the count of your need for the purpose of loading several JARs. Then on each line, click the three dot button to open the **Select Module** wizard from which you can select a driver JAR of your interest for each line.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Type in the name of the table.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined

		in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for any database using a JDBC connection.	

Related scenarios

Related topics in **tDBInput** scenarios:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

Related topic in **tContextLoad**: see section *Scenario: Dynamic context use in MySQL DB insert*.

tJDBCOutput

tJDBCOutput properties

Component family	Databases/JDBC	
Function	**tJDBCOutput** writes, updates, makes changes or suppresses entries in any type of database connected to a JDBC API.	
Purpose	**tJDBCOutput** executes the action defined on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	JDBC URL	Type in the database location path
	Driver JAR	Click the plus button under the table to add lines of the count of your need for the purpose of loading several JARs. Then on each line, click the three dot button to open the **Select Module** wizard from which you can select a driver JAR of your interest for each line.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow.

		It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a JDBC database. It also allows you to create a reject flow using a	

	Row > Rejects link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.

Related scenarios

For **tJDBCOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tJDBCRollback

tJDBCRollback properties

This component is closely related to **tJDBCCommit** and **tJDBCConnection**. It usually does not make much sense to use JDBC components independently in a transaction.

Component family	Databases/JDBC	
Function	Cancels the transaction committed in the connected DB.	
Purpose	Avoid commiting part of a transaction accidentally.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection** and **tJDBCCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tJDBCConnection** and **tJDBCCommit**. It usually does not make much sense to use JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCRollback** related scenario, see section *tMysqlRollback*

tJDBCRow

tJDBCRow properties

Component family	Databases/JDBC	
Function	**tJDBCRow** is the component for any type database using a JDBC API. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tJDBCRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	JDBC URL	Type in the database location path.
	Driver JAR	Click the plus button under the table to add lines of the count of your need for the purpose of loading several JARs. Then on each line, click the three dot button to open the **Select Module** wizard from which you can select a driver JAR of your interest for each line.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Name of the table to be processed.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the component. To prevent this from happening, use a shared DB connection with the data source alias specified.* This check box is not available when the **Use an existing connection** check box is selected.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query for any database using a JDBC connection and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output* .
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tJDBCSP

tJDBCSP Properties

Component family	Databases/JDBC	
Function	**tJDBCSP** calls the specified database stored procedure.	
Purpose	**tJDBCSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*JDBC URL*	Type in the database location path
	Driver JAR	Click the plus button under the table to add lines of the count of your need for the purpose of loading several JARs. Then on each line, click the three dot button to open the **Select Module** wizard from which you can select a driver JAR of your interest for each line.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SP Name	Type in the exact name of the Stored Procedure.
	Is Function / *Return result in*	Select this check box , if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check section *tPostgresqlCommit*, if you want to analyze a set of records from a database table or DB query and return single records.
	Specify a data source alias	Select this check box and specify the alias of a data source created on the **Talend** Runtime side to use the shared connection pool defined in the data source configuration. This option works only when you deploy and run your Job in **Talend** Runtime. *If you use the component's own DB configuration, your data source connection will be closed at the end of the*

		component. To prevent this from happening, use a shared DB connection with the data source alias specified. This check box is not available when the **Use an existing connection** check box is selected.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Related scenario

For related scenarios, see:

- section *Scenario: Executing a stored procedure in the MDM Hub*.
- section *Scenario: Checking number format using a stored procedure*

Check as well section *tPostgresqlCommit* if you want to analyze a set of records from a database table or DB query and return single records.

tJDBCTableList

tJDBCTableList Properties

Component family	Databases/JDBC	
Function	Iterates on a set of table names through a defined JDBC connection.	
Purpose	Lists the names of a given set of JDBC tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection**.	
Limitation	n/a	

Related scenario

For **tJDBCTableList** related scenario, see section *Scenario: Iterating on a DB table and listing its column names*.

tLDAPAttributesInput

tLDAPAttributesInput Properties

Component family	Databases/LDAP	
Function	**tLDAPAttributesInput** analyses each object found via the LDAP query and lists a collection of attributes associated with the object.	
Purpose	**tLDAPAttributesInput** executes an LDAP query based on the given filter and corresponding to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tLDAPConnection** component on the **Component list** to reuse the connection details you already defined.
	Host	LDAP Directory server IP address.
	Port	Listening port number of server.
	Base DN	Path to user's authorised tree leaf.
	Protocol	Select the protocol type on the list. **LDAP** : no encryption is used **LDAPS**: secured LDAP. When this option is chosen, the **Advanced CA** check box appears. Once selected, the advanced mode allows you to specify the directory and the keystore password of the certificate file for storing a specific CA. However, you can still deactivate this certificate validation by selecting the **Trust all certs** check box. **TLS**: certificate is used. When this option is chosen, the **Advanced CA** check box appears and is used the same way as that of the **LDAPS** type.
	Authentication User and *Password*	Select the **Authentication** check box if LDAP login is required. Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Filter	Type in the filter as expected by the LDAP directory db.
	Multi valued field separator	Type in the value separator in multi-value fields.
	Alias dereferencing	Select the option on the list. **Never** improves search performance if you are sure that no alias is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution
	Referral handling	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections
	Limit	Fill in a limit number of records to be read If needed.

	Time Limit	Fill in a timeout period for the directory. access
	Paging	Specify the number of entries returned at a time by the LDAP server.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		The pre-defined schema lists: - *objectclass*: list of object classes - *mandatoryattributes*: list of mandatory attributes to these classes - *optionalattributes*: list of optional attributes to these classes - *objectattributes*: list of attributes that are essential for the analysed object.
Advanced settings	*Class Definition Root*	Specify the root of the object class definition namespace.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component covers all possible LDAP queries. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve automatically the relevant Base	

Related scenario

The **tLDAPAttributesInput** component follows the usage similar to that of **tLDAPInput**. Hence for **tLDAPInput** related scenario, see section *Scenario: Displaying LDAP directory's filtered content*.

tLDAPConnection

tLDAPConnection Properties

Component family	Databases/LDAP	
Function	Opens a connection to an LDAP Directory server for data transaction.	
Purpose	This component creates a connection to an LDAP Directory server. Then it can be invoked by other components that need to access the LDAP Directory server, e.g., **tLDAPInput, tLDAPOutput,** etc.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	LDAP Directory server IP address.
	Port	Listening port number of server.
	Protocol	Select the protocol type on the list. **LDAP**: no encryption is used **LDAPS**: secured LDAP. When this option is chosen, the **Advanced CA** check box appears. Once selected, the advanced mode allows you to specify the directory and the keystore password of the certificate file for storing a specific CA. However, you can still deactivate this certificate validation by selecting the **Trust all certs** check box. **TLS**: certificate is used. When this option is chosen, the **Advanced CA** check box appears and is used the same way as that of the **LDAPS** type.
	Base DN	Path to user's authorized tree leaf.
	User and *Password*	Fill in the **User** and **Password** as required by the directory Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Alias dereferencing	Select the option on the list. **Never** improves search performance if you are sure that no aliases is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution
	Referral handling	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is to be used with other LDAP components, especially with **tLDAPInput** and **tLDAPOutput**.	

Related scenarios

This component is closely related to **tLDAPInput** and **tLDAPOutput** as it frees you from filling in the connection details repeatedly if multiple LDAP input/output components exist.

For **tLDAPConnection** related scenarios, see section *Scenario: Inserting data in mother/daughter tables*.

tLDAPInput

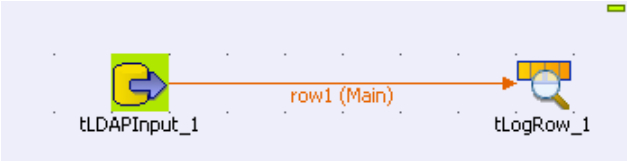
tLDAPInput Properties

Component family	Databases/LDAP	
Function	**tLDAPInput** reads a directory and extracts data based on the defined filter.	
Purpose	**tLDAPInput** executes an LDAP query based on the given filter and corresponding to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tLDAPConnection** component on the **Component list** to reuse the connection details you already defined.
	Host	LDAP Directory server IP address.
	Port	Listening port number of server.
	Base DN	Path to the user's authorised tree leaf. To retrieve the full DN information, enter a field named *DN* in the schema, in either upper case or lower case.
	Protocol	Select the protocol type on the list. **LDAP** : no encryption is used **LDAPS**: secured LDAP. When this option is chosen, the **Advanced CA** check box appears. Once selected, the advanced mode allows you to specify the directory and the keystore password of the certificate file for storing a specific CA. However, you can still deactivate this certificate validation by selecting the **Trust all certs** check box. **TLS**: certificate is used When this option is chosen, the **Advanced CA** check box appears and is used the same way as that of the **LDAPS** type.
	Authentication User and *Password*	Select the **Authentication** check box if LDAP login is required. Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Filter	Type in the filter as expected by the LDAP directory db.
	Multi valued field separator	Type in the value separator in multi-value fields.
	Alias dereferencing	Select the option on the list. **Never** improves search performance if you are sure that no alias is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution
	Referral handling	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections

	Limit	Fill in a limit number of records to be read If needed.
	Time Limit	Fill in a timeout period for the directory. access
	Paging	Specify the number of entries returned at a time by the LDAP server.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Usage	This component covers all possible LDAP queries. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve automatically the relevant Base.	

Scenario: Displaying LDAP directory's filtered content

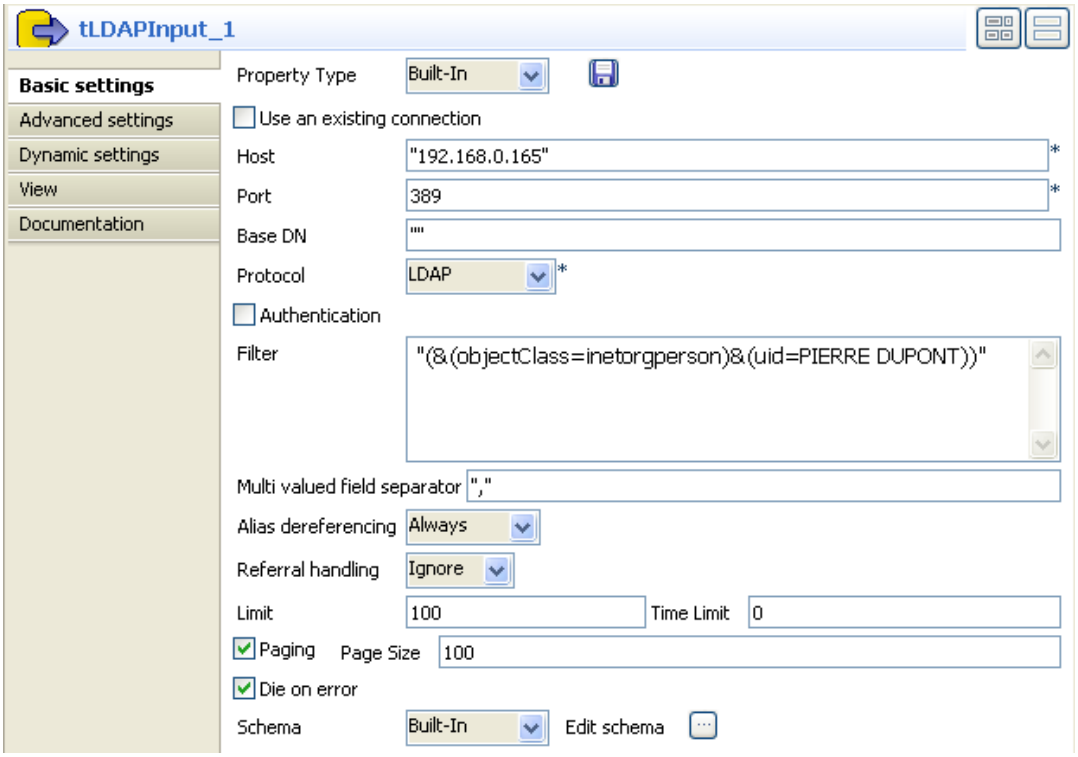
The Job described below simply filters the LDAP directory and displays the result on the console.



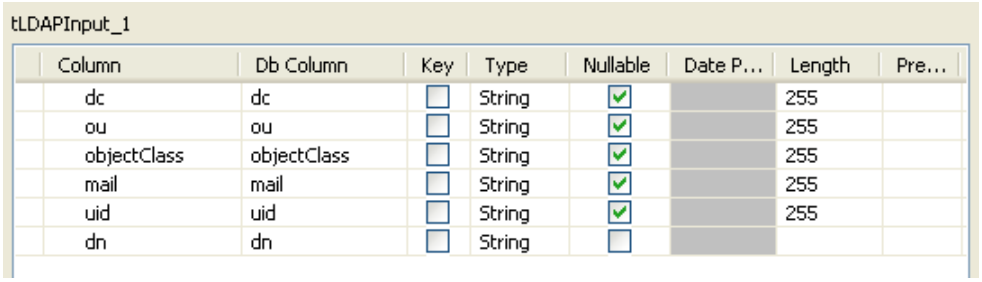
- Drop the **tLDAPInput** component along with a **tLogRow** from the **Palette** to the design workspace.
- Set the **tLDAPInput** properties.
- Set the **Property type** on **Repository** if you stored the LDAP connection details in the **Metadata Manager** in the **Repository**. Then select the relevant entry on the list.

  *For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

- In **Built-In** mode, fill in the **Host** and **Port** information manually. Host can be the IP address of the LDAP directory server or its DNS name.
- No particular **Base DN** is to be set.



- Then select the relevant **Protocol** on the list. In this example: a simple **LDAP** protocol is used.
- Select the **Authentication** check box and fill in the login information if required to read the directory. In this use case, no authentication is needed.
- In the **Filter** area, type in the command, the data selection is based on. In this example, the filter is: `(&(objectClass=inetorgperson)&(uid=PIERRE DUPONT))`.
- Fill in **Multi-valued field separator** with a comma as some fields may hold more than one value, separated by a comma.
- As we do not know if some aliases are used in the LDAP directory, select **Always** on the list.
- Set **Ignore** as **Referral handling**.
- Set the limit to **100** for this use case.



- Set the **Schema** as required by your LDAP directory. In this example, the schema is made of 6 columns including the `objectClass and uid columns which get filtered on.`
- In the **tLogRow** component, no particular setting is required.

```
Starting job testLDAPInput at 18:05 18/09/2007.
|DATA|top,person,organizationalPerson,inetorgperson,a4400user|mhirt78@talend.com|PIERRE DUPONT|
Job testLDAPInput ended at 18:05 18/09/2007. [exit code=0]
```

Only one entry of the directory corresponds to the filter criteria given in the **tLDAPInput** component.

tLDAPOutput

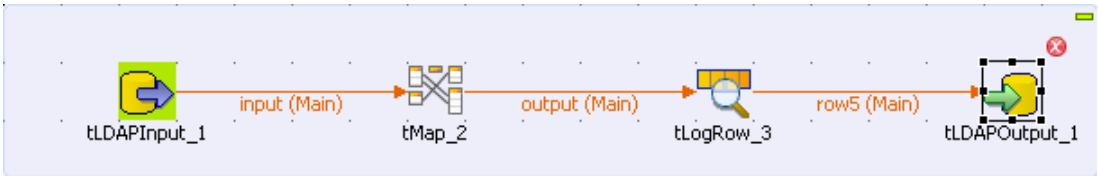
tLDAPOutput Properties

Component family	Databases/LDAP	
Function	**tLDAPOutput** writes into an LDAP directory.	
Purpose	**tLDAPOutput** executes an LDAP query based on the given filter and corresponding to the schema definition. Then it passes on the field list to the next component via a **Main row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tLDAPConnection** component on the **Component list** to reuse the connection details you already defined.
	Host	LDAP Directory server IP address.
	Port	Listening port number of server.
	Base DN	Path to user's authorized tree leaf.
	Protocol	Select the protocol type on the list. **LDAP** : no encryption is used **LDAPS**: secured LDAP. When this option is chosen, the **Advanced CA** check box appears. Once selected, the advanced mode allows you to specify the directory and the keystore password of the certificate file for storing a specific CA. However, you can still deactivate this certificate validation by selecting the **Trust all certs** check box. **TLS**: certificate is used When this option is chosen, the **Advanced CA** check box appears and is used the same way as that of the **LDAPS** type.
	User and *Password*	Fill in the **User** and **Password** as required by the directory Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Multi valued field separator	Character, string or regular expression to separate data in a multi-value field.
	Alias dereferencing	Select the option on the list. **Never** improves search performance if you are sure that no aliases is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution
	Referral handling	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections
	Insert mode	Select the editing mode on the list:

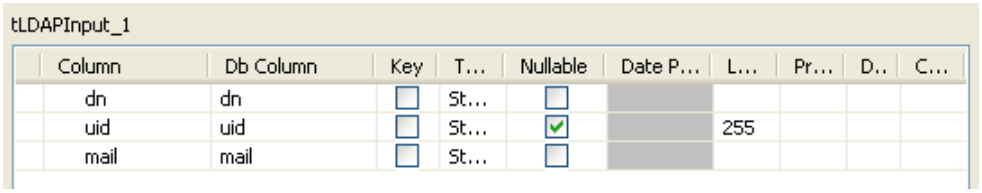
		Add: add a value in a multi-value attribute, **Insert**: insert new data, **Updata**: updates the existing data, **Delete**: remove the selected data from the directory, **Insert or Update**: insert new data or update existing ones.
	DN Column Name	Select in the list the type of the LDAP input entity used.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
Advanced settings	*Use Attribute Options (for update mode)*	Select this check box to choose the desired attribute (including dn, dc, ou, objectClass, mail and uid) and the corresponding operation (including Add, Replace, Remove Attribute and Remove Value).
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component covers all possible LDAP queries. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve the relevant DN Base automatically. This component allows you to carry out actions on a table or on the data of a table in an database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Scenario: Editing data in a LDAP directory

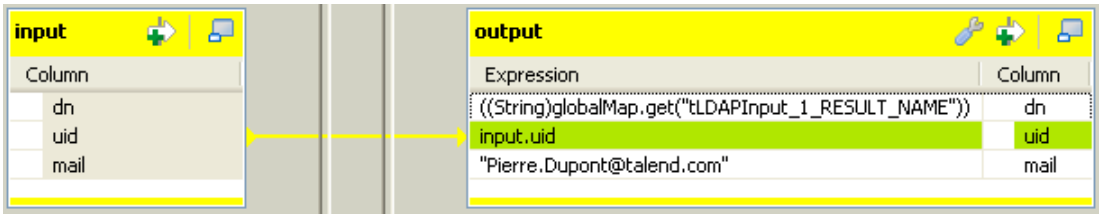
The following scenario describes a Job that reads an LDAP directory, updates the email of a selected entry and displays the output before writing the LDAP directory. To keep it simple, no alias dereferencing nor referral handling is performed. This scenario is based on section *Scenario: Displaying LDAP directory's filtered content*. The result returned was a single entry, related to an organisational person, whom email is to be updated.



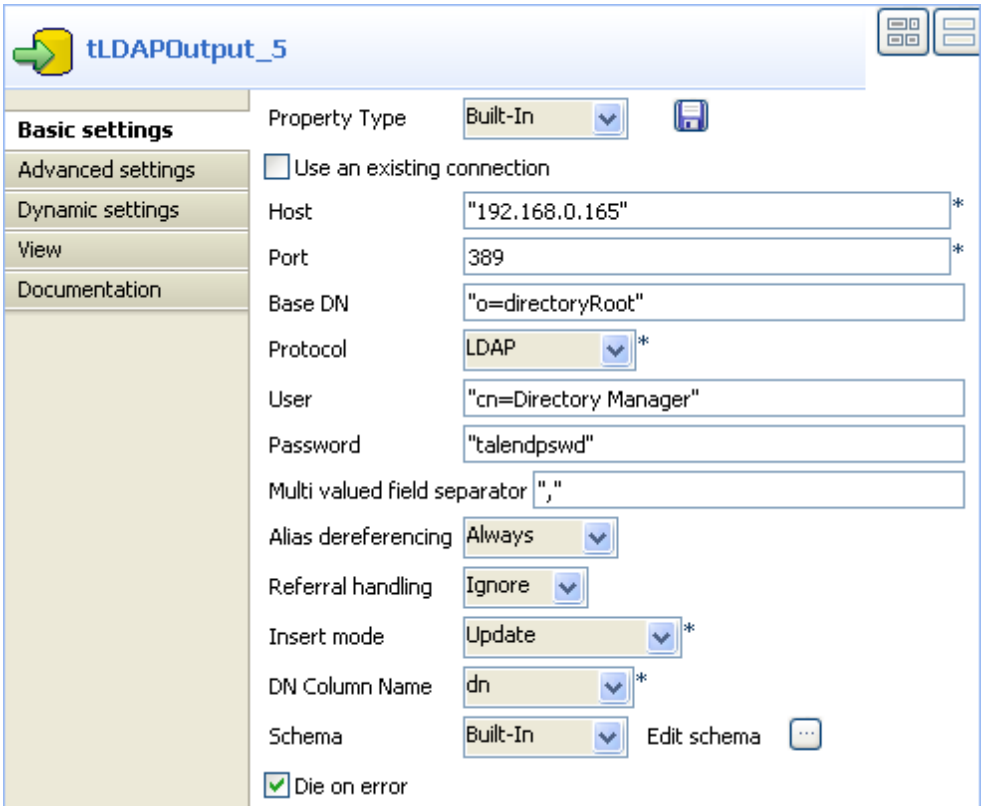
- Drop the **tLDAPInput**, **tLDAPOutput**, **tMap** and **tLogRow** components from the **Palette** to the design workspace.
- Connect the input component to the **tMap** then to the **tLogRow** and to the output component.
- In the **tLDAPInput** Component view, set the connection details to the LDAP directory server as well as the filter as described in section *Scenario: Displaying LDAP directory's filtered content*.
- Change the schema to make it simpler, by removing the unused fields: *dc*, *ou*, *objectclass*.



- Then open the mapper to set the edit to be carried out.
- Drag & drop the uid column from the input table to the output as no change is required on this column.



- In the **Expression** field of the *dn* column (output), fill in with the exact expression expected by the LDAP server to reach the target tree leaf and allow directory writing on the condition that you haven't set it already in the **Base DN** field of the **tLDAPOutput** component.
- In this use case, the **GetResultName** global variable is used to retrieve this path automatically. Press **Ctrl +Space bar** to access the variable list and select `tLDAPInput_1_RESULT_NAME`.
- In the *mail* column's expression field, type in the new email that will overwrite the current data in the LDAP directory. In this example, we change to *Pierre.Dupont@talend.com*.
- Click **OK** to validate the changes.
- The **tLogRow** component does not need any particular setting.
- Then select the **tLDAPOutput** component to set the directory writing properties.



- Set the **Port** and **Host** details manually if they aren't stored in the **Repository**.

  *For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query type** of components are always **Built-in**. You have to complete the relevant settings manually in the specific Job.*

- In **Base DN** field, set the highest tree leaf you have the rights to access. If you have not set previously the exact and full path of the target DN you want to access, then fill in it here. In this use case, the full DN is provided by the *dn* output from the **tMap** component, therefore only the highest accessible leaf is given: *o=directoryRoot*.
- Select the relevant protocol to be used: **LDAP** for this example.
- Fill in the **User** and **Password** as expected by the LDAP directory.
- Fill in **Multi-valued field separator** with a comma as some fields may hold more than one value, separated by a comma.
- Use the default setting of **Alias Dereferencing** and **Referral Handling** fields, respectively **Always** and **Ignore**.
- The **Insert mode** for this use case is **Update** (the email address).
- The schema was provided by the previous component through the propagation operation.
- Save the Job and execute.

```
Starting job LDAPInputnew at 14:17 20/09/2007.
uid=PIERRE DUPONT,ou=DATA,o=TALENDM,o=TALEND|PIERRE DUPONT|Pierre.Dupont@talend.com
Job LDAPInputnew ended at 14:17 20/09/2007. [exit code=0]
```

The output shows the following fields: *dn*, *uid* and *mail* as defined in the Job.

tLDAPRenameEntry

tLDAPRenameEntry properties

Component family	Databases/LDAP	
Function	**tLDAPRenameEntry** renames entries in an LDAP directory.	
Purpose	The **tLDAPRenameEntry** component rename ones or more entries in a specific LDAP directory.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tLDAPConnection** component on the **Component list** to reuse the connection details you already defined.
	Host	LDAP directory server IP address.
	Port	Number of the listening port of the server.
	Base DN	Path to user's authorized tree leaf.
	Protocol	Select the protocol type on the list. **LDAP**: no encryption is used, **LDAPS**: secured LDAP, **TLS**: certificate is used.
	User and *Password*	Fill in user authentication information. Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Alias dereferencing	Select the option on the list. **Never** improves search performance if you are sure that no alias is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases, **Never**: Never dereferences aliases, **Searching**: Dereferences aliases only after name resolution, **Finding**: Dereferences aliases only during name resolution.
	Referrals handling	Select the option on the list: **Ignore**: does not handle request redirections, **Follow**: does handle request redirections.
	Previous DN and *New DN*	Select from the list the schema column that holds the old DN (**Previous DN**) and the column that holds the new DN (**New DN**).
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.

Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component covers all possible LDAP queries. It is usually used as a one-component subjob but you can use it with other components as well. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve automatically the relevant DN Base.	

Related scenarios

For use cases in relation with **tLDAPRenameEntry**, see the following scenarios:

- section *Scenario: Displaying LDAP directory's filtered content*.
- section *Scenario: Editing data in a LDAP directory*.

tMaxDBInput

tMaxDBInput properties

Component family	Databases/MaxDB	
Function	**tMaxDBInput** reads a database and extracts fields based on a query.	
Purpose	**tMaxDBInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host name	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Type in the table name.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Guess schema	Click the **Guess schema** button to retrieve the table schema.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

tMaxDBOutput

tMaxDBOutput properties

Component family	Databases/MaxDB	
Function	**tMaxDBOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tMaxDBOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the **Use field options** check box and then in the **Key in update** column, select the check boxes next to the column names you want to use as a base for the Update*

		operation. Do the same in the ***Key in delete*** *column for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a database. It also allows you to create a reject flow using a **Row** > **Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tMaxDBRow

tMaxDBRow properties

Component family	Databases/MaxDB	
Function	**tMaxDBRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tMaxDBRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Type in the table name.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Guess Query	Click the **Guess Query** button to generate the query which corresponds to your table schema in the Query field.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.

	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possible SQL queries.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see:

- section *Scenario 1: Displaying selected data from DB table*
- section *Scenario 2: Using StoreSQLQuery variable*

tMongoDBClose

tMongoDBClose belongs to two component families: Big Data and Databases. For more information about **tMongoDBClose**, see section *tMongoDBClose*.

tMongoDBConnection

tMongoDBConnection belongs to two component families: Big Data and Databases. For more information about **tMongoDBConnection**, see section *tMongoDBConnection*.

tMongoDBInput

tMongoDBInput belongs to two component families: Big Data and Databases. For more information about **tMongoDBInput**, see section *tMongoDBInput*.

tMongoDBOutput

tMongoDBOutput belongs to two component families: Big Data and Databases. For more information about **tMongoDBOutput**, see section *tMongoDBOutput*.

tMongoDBRow

tMongoDBRow belongs to two component families: Big Data and Databases. For more information about **tMongoDBRow**, see section *tMongoDBRow*.

tParseRecordSet

You can find this component at the root of **Databases** group of the **Palette** of *Talend Open Studio for Big Data*. **tParseRecordSet** covers needs related indirectly to the use of any database.

tParseRecordSet properties

Component family	Databases	
Function	**tParseRecordSet** parses a set of records from a database table or DB query and possibly returns single records.	
Purpose	.Parses a recordset rather than individual records from a table.	
Basic settings	*Prev. Comp. Column list*	Set the column from the database that holds the recordset.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Attribute table	Set the position value of each column for single records from the recordset.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	This component is mainly designed for a use with the SP component Recordset feature.	

Related Scenario

For an example of **tParseRecordSet** in use, see section *Scenario 2: Using PreparedStatement objects to query data*.

tPostgresPlusBulkExec

tPostgresPlusBulkExec properties

The **tPostgresplusOutputBulk** and **tPostgresplusBulkExec** components are generally used together as part of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tPostgresPlusOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases/PostgresPlus	
Function	**tPostgresPlusBulkExec** executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tPostgresPlusBulkExec** allows gains in performance during Insert operations to a DB2 database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tPostgresPlusConnection** component on the **Component List** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again.

		Create table: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Action*	Select the action to be carried out **Bulk insert Bulk update** Depending on the action selected, the required information varies.
	Field terminated by	Character, string or regular expression to separate fields.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This dedicated component offers performance and flexibility of DB2 query handling.	

Related scenarios

For **tPostgresPlusBulkExec** related topics, see:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresPlusClose

tPostgresPlusClose properties

Component family	Databases/Postgres	
Function	**tPostgresPlusClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tPostgresPlusConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with PostgresPlus components, especially with **tPostgresPlusConnection** and **tPostgresPlusCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tPostgresPlusCommit

tPostgresPlusCommit Properties

This component is closely related to **tPostgresPlusConnection** and **tPostgresPlusRollback**. It usually does not make much sense to use JDBC components independently in a transaction.

Component family	Databases/PostgresPlus	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tPostgresPlusConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tPostgresPlusCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with PostgresPlus components, especially with the **tPostgresPlusConnection** and **tPostgresPlusRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresPlusConnection** and **tPostgresPlusRollback**. It usually doesn't make much sense to use PostgresPlus components without using the **tPostgresPlusConnection** component to open a connection for the current transaction.

For **tPostgresPlusCommit** related scenario, see section *tMysqlConnection*

tPostgresPlusConnection

tPostgresPlusConnection Properties

This component is closely related to **tPostgresPlusCommit** and **tPostgresPlusRollback**. It usually doesn't make much sense to use one of PostgresPlus components without using the **tPostgresPlusConnection** component to open a connection for the current transaction.

Component family	Databases/PostgresPlus	
Function	Opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	Enter your DB authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with PostgresPlus components, especially with the **tPostgresPlusCommit** and **tPostgresPlusRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresPlusCommit** and **tPostgresPlusRollback**. It usually doesn't make much sense to use one of PostgresPlus components without using the **tPostgresPlusConnection** component to open a connection for the current transaction.

For **tPostgresPlusConnection** related scenario, see section *tMysqlConnection*

tPostgresPlusInput

tPostgresPlusInput properties

Component family	Databases/ PostgresPlus	
Function	**tPostgresPlusInput** reads a database and extracts fields based on a query.	
Purpose	**tPostgresPlusInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tPostgresplusConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Use cursor*	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possible SQL queries for Postgresql databases.	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.

tPostgresPlusOutput

tPostgresPlusOutput properties

Component family	Databases/PostgresPlus	
Function	**tPostgresPlusOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tPostgresPlusOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box when using a configured **tPostgresPlusConnection** component. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.

		Truncate table: The table content is deleted. You don not have the possibility to rollback the operation.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Ensure that the Nullable check box is selected for the corresponding columns in the schema.
	Use batch size	Select this check box to activate the batch mode for data processing. In the **Batch Size** field that appears when this check box is selected,

		you can type in the number you need to define the batch size to be processed. This check box is available only when you have selected the **Insert**, the **Update** or the **Delete** option in the **Action on data** field.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a PostgresPlus database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySqlOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related scenarios

For **tPostgresPlusOutput** related topics, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.
- section *Scenario 1: Adding a new column and altering data in a DB table*.

tPostgresPlusOutputBulk

tPostgresPlusOutputBulk properties

The **tPostgresplusOutputBulk** and **tPostgresplusBulkExec** components are generally used together as part of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tPostgresPlusOutputBulkExec** component, detailed in a separate section. The advantage of using two separate components is that the data can be transformed before it is loaded in the database.

Component family	Databases/PostgresPlus	
Function	Writes a file with columns based on the defined delimiter and the PostgresPlus standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the PostgresPlus database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Append	Select this check box to add the new rows at the end of the file
	Include header	Select this check box to include the column header to the file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component is to be used along with **tPostgresPlusBulkExec** component. Used together they offer gains in performance while feeding a PostgresPlus database.	

Related scenarios

For use cases in relation with **tPostgresplusOutputBulk**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresPlusOutputBulkExec

tPostgresPlusOutputBulkExec properties

The **tPostgresplusOutputBulk** and **tPostgresplusBulkExec** components are generally used together as part of a two step process. In the first step, an output file is generated. In the second step, this file is used in the INSERT operation used to feed a database. These two steps are fused together in the **tPostgresPlusOutputBulkExec** component.

Component family	Databases/PostgresPlus	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a PostgresPlus database.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
Advanced settings	*Action*	Select the action to be carried out **Bulk insert Bulk update** Depending on the action selected, the required information varies.
	File type	Select the type of file being handled.
	Null string	String displayed to indicate that the value is null.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Field terminated by	Character, string or regular expression to separate fields.
	Text enclosure	Character used to enclose text.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular tranformation is required on the data to be loaded onto the database.	

Related scenarios

For use cases in relation with **tPostgresPlusOutputBulkExec**, see the following scenarios:

- section *Scenario: Inserting transformed data in MySQL database*.
- section *Scenario: Inserting data in MySQL database*.
- section *Scenario: Truncating and inserting file data into Oracle DB*.

tPostgresPlusRollback

tPostgresPlusRollback properties

This component is closely related to **tPostgresPlusCommit** and **tPostgresPlusConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/PostgresPlus	
Function	**tPostgresPlusRollback** cancels the transaction committed in the connected DB.	
Purpose	This component avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tPostgresPlusConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with PostgresPlus components, especially with **tPostgresPlusConnection** and **tPostgresPlusCommit**.	
Limitation	n/a	

Related scenarios

For **tPostgresPlusRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tPostgresPlusRow

tPostgresPlusRow properties

Component family	Databases/Postgresplus	
Function	**tPostgresPlusRow** is the specific component for the database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tPostgresPlusRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tPostgresPlusConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table name	Name of the table to be read.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover, the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Related scenarios

For related topics, see:

- section *Scenario 3: Combining two flows for selective output*
- section *Scenario: Resetting a DB auto-increment*.
- section *Scenario 1: Removing and regenerating a MySQL table index*.

tPostgresPlusSCD

tPostgresPlusSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tPostgresPlusSCD*.

tPostgresPlusSCDELT

tPostgresPlusSCDELT belongs to two component families: Business Intelligence and Databases. For more information on it, see section *tPostgresPlusSCDELT*.

tSasInput

Before being able to benefit from all functional objectives of the SAS components, make sure to install the following three modules: sas.core.jar, sas.intrnet.javatools.jar and sas.svc.connection.jar in the path lib > java in your Talend Open Studio for Big Data directory. You can later verify, if needed whether the modules are successfully installed through the Modules view of the Studio.

tSasInput properties

Component family	Databases/SAS	
Function	**tSasInput** reads a database and extracts fields based on a query.	
Purpose	**tSasInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the component that follows via a **Row** > **Main** connection.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host name	SAS server IP address.
	Port	Listening port number of server.
	Librefs	Enter the directory name that holds the table to read followed by its access path. For example: *"TpSas 'C:/SAS/TpSas'"*
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Table Name	Enter the name of the table to read preceded by the directory name that holds it. For example: `"TpSas.Customers"`.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
	Query	Type in your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component covers all possible SQL queries for databases using SAS connections.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For related topics, see:

- section *Scenario 1: Displaying selected data from DB table*.
- section *Scenario 2: Using StoreSQLQuery variable*.
- section *Scenario: Dynamic context use in MySQL DB insert*.

tSasOutput

Before being able to benefit from all functional objectives of the SAS components, make sure to install the following three modules: sas.core.jar, sas.intrnet.javatools.jar and sas.svc.connection.jar in the path lib > java in your Talend Open Studio for Big Data directory. You can later verify, if needed whether the modules are successfully installed through the Modules view of the Studio.

tSasOutput properties

Component family	Databases/SAS	
Function	**tSasOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tSasOutput** executes the action defined on the table and/or on the data contained in the table, based on the incoming flow from the preceding component in the Job.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tSASConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	SAS URL	Enter the URL to connect to the desired DB.
	Driver JAR	In the drop down list, select a desired available driver, or download one from a local directory through clicking the three-dot button.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to read.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting*

		*the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Clear data in table	Select this check box to delete data in the selected table before any operation.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as a new column.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tSasOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all of the SQL queries possible. This component must be used as an output component. It allows you to carry out actions on a table or on the data of a table in a SAS database. It also allows you to create a reject flow using a **Row > Rejects** link to filter data in error. For an example of **tMySQLOutput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For scenarios in which **tSasOutput** might be used, see:

- section *Scenario: Writing a row to a table in the MySql database via an ODBC connection*.

- section *Scenario 1: Adding a new column and altering data in a DB table*.

tSQLiteClose

tSQLiteClose properties

Component family	Databases/SQLite	
Function	**tSQLiteClose** closes the transaction committed in the connected DB.	
Purpose	Close a transaction.	
Basic settings	*Component list*	Select the **tSQLiteConnection** component in the list if more than one connection are planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with SQLite components, especially with **tSQLiteConnection** and **tSQLiteCommit**.	
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tSQLiteCommit

tSQLiteCommit Properties

This component is closely related to **tSQLiteConnection** and **tSQLiteRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/SQLite	
Function	**tSQLiteCommit** validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tSQLiteConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	This check box is selected by default. It allows you to close the database connection once the commit is done. Clear this check box to continue to use the selected connection once the component has performed its task. *If you want to use a **Row** > **Main** connection to link **tSQLiteCommit** to your Job, your data will be commited row by row. In this case, do not select the **Close connection** check box or your connection will be closed before the end of your first row commit.*
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with SQLite components, especially with **tSQLiteConnection** and **tSQLiteRollback**.	
Limitation	n/a	

Related scenario

This component is closely related to **tSQLiteConnection** and **tSQLiteRollback**. It usually does not make much sense to use one of these without using a **tSQLiteConnection** component to open a connection for the current transaction.

For **tSQLiteCommit** related scenario, see section *Scenario: Inserting data in mother/daughter tables*.

tSQLiteConnection

SQLiteConnection properties

This component is closely related to **tSQLiteCommit** and **tSQLiteRollback**. It usually does not make much sense to use one of these without using a **tSQLiteConnection** to open a connection for the current transaction.

Component family	Databases/SQLite	
Function	**tSQLiteConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows you to commit all of the Job data to an output database in just a single transaction, once the data has been validated.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Database	Name of the database.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. This allows you to share one single DB connection among several DB connection components from different Job levels that can be either parent or child. *This option is incompatible with the **Use dynamic job** and **Use an independent process to run subjob** options of the **tRunJob** component. Using a shared database connection together with a **tRunJob** component with either of these two options enabled will cause your Job to fail.* **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with SQLite components, especially with **tSQLiteCommit** and **tSQLiteRollback**.	
Limitation	n/a	

Related scenarios

This component is closely related to **tSQLiteCommit** and **tSQLiteRollback**. It usually does not make much sense to use one of these without using a **tSQLiteConnection** component to open a connection for the current transaction.

For **tSQLiteConnection** related scenario, see section *tMysqlConnection*

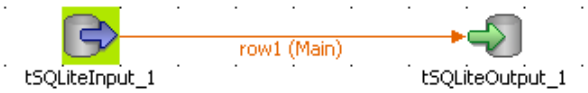
tSQLiteInput

tSQLiteInput Properties

Component family	Databases	
Function	**tSQLiteInput** reads a database file and extracts fields based on an SQL query. As it embeds the SQLite engine, no need of connecting to any database server.	
Purpose	**tSQLiteInput** executes a DB query with a defined command which must correspond to the schema definition. Then it passes on rows to the next component via a Main row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see see *Talend Open Studio for Big Data User Guide*.
	Database	Filepath to the SQLite database file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
	Query	Type in your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is standalone as it includes the SQLite engine. This is a startable component that can iniate a data flow processing.	

Scenario: Filtering SQlite data

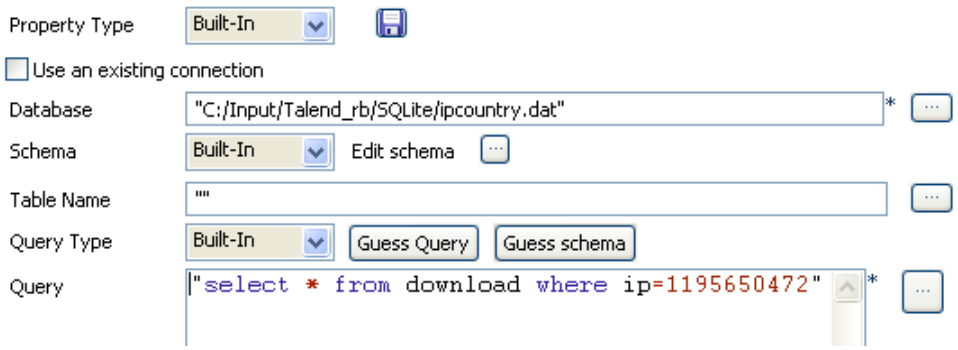
This scenario describes a rather simple job which uses a select statement based on a filter to extract rows from a source SQLite Database and feed an output SQLite table.



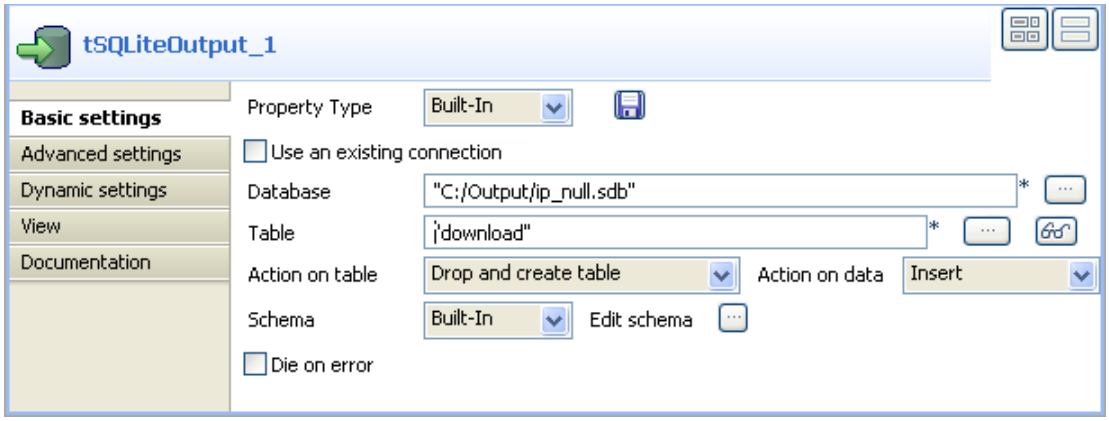
- Drop from the **Palette**, a **tSQLiteInput** and a **tSQLiteOutput** component from the **Palette** to the design workspace.
- Connect the input to the output using a row main link.
- On the **tSQLiteInput** Basic settings, type in or browse to the SQLite Database input file.

	id	version	download_date	ip	type	type_os
1	20027	TOS-Win32-200610	13/11/2006	1191947907	1	101
2	20028	TOS-Win32-200610	13/11/2006	1195650472	1	102
3	20030	TOS-Win32-200610	13/11/2006	3375565745	1	103
4	20031	TOS-Win32-200610	13/11/2006	1195650472	1	104
5	20032	TOS-Win32-200610	13/11/2006	1195650472	1	105
6	20033	TOS-Win32-200610	13/11/2006	1104872453	1	106
7	20034	TOS-Win32-200610	13/11/2006	1104872453	1	107
8	20036	TOS-Win32-200610	13/11/2006	1190898057	1	108
9	20037	TOS-Win32-200610	13/11/2006	1190898057	1	109
10	20038	TOS-Win32-200610	13/11/2006	1348977142	1	110
11	20040	TOS-Win32-200610	13/11/2006	3581349521	1	11
12	20041	TOS-Win32-200610	13/11/2006	1190898057	1	12
13	20043	TOS-Win32-200610	13/11/2006	1196485544	1	13
14	20044	TOS-Win32-200610	13/11/2006	1066743463	1	14
15	20045	TOS-Win32-200610	13/11/2006	1196485544	1	15
16	20046	TOS-Win32-200610	13/11/2006	2624217794	1	16

- The file contains hundreds of lines and includes an **ip** column which the select statement will based on
- On the **tSQLite** Basic settings, edit the schema for it to match the table structure.



- In the **Query** field, type in your select statement based on the *ip* column.
- On the **tSQLiteOutput** component **Basic settings** panel, select the **Database** filepath.



- Type in the **Table** to be fed with the selected data.
- Select the **Action on table** and **Action on Data**. In this use case, the action on table is *Drop and create* and the action on data is *Insert*.
- The schema should be synchronized with the input schema.
- Save the job and run it.

	id	version	download_date	ip	type	type_os
1	20028	TOS-Win32-200610	13/11/2006	1195650472	1	102
2	20031	TOS-Win32-200610	13/11/2006	1195650472	1	104
3	20032	TOS-Win32-200610	13/11/2006	1195650472	1	105

The data queried is returned in the defined SQLite file.

tSQLiteOutput

tSQLiteOutput Properties

Component family	Databases	
Function	**tSQLiteOutput** writes, updates, makes changes or suppresses entries in an SQLite database. As it embeds the SQLite engine, no need of connecting to any database server.	
Purpose	**tSQLiteOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Database	Filepath to the Database file
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops.

		Update: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the **Update** and **Delete** operations are based. You can do that by clicking **Edit Schema** and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the **Advanced settings** view where you can simultaneously define primary keys for the **Update** and **Delete** operations. To do that: Select the **Use field options** check box and then in the **Key in update column**, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the **Key in delete column** for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component must be connected to an Input component. It allows you to carry out actions on a table or on the data of a table in an SQLite database. It also allows you to create reject flows using a **Row > Reject** link to filter erroneous data. For an example of **tSQLiteOuput** in use, see section *Scenario 3: Retrieve data in error with a Reject link*.	

Related Scenario

For scenarios related to **tSQLiteOutput**, see section *Scenario 3: Retrieve data in error with a Reject link*.

tSQLiteRollback

tSQLiteRollback properties

This component is closely related to **tSQLiteCommit** and **tSQLiteConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/SQLite	
Function	**tSQLiteRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tSQLiteConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with SQLite components, especially with **tSQLiteConnection** and **tSQLiteCommit**.	
Limitation	n/a	

Related scenarios

For **tSQLiteRollback** related scenario, see section *Scenario: Rollback from inserting data in mother/daughter tables*.

tSQLiteRow

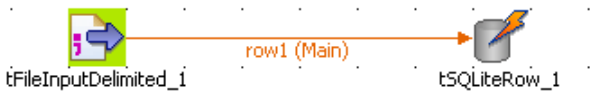
tSQLiteRow Properties

Component family	Databases	
Function	**tSQLiteRow** executes the defined query onto the specified database and uses the parameters bound with the column.	
Purpose	A prepared statement uses the input flow to replace the placeholders with the values for each parameters defined. This component can be very useful for updates.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the **Component list** to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see *Talend Open Studio for Big Data User Guide*.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Query type	In *Talend Open Studio for Big Data*, the Query type is always **Built-in**.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow. Select this column from the **use column** list. This option allows the component to have a different schema from that of the preceding component. Moreover,

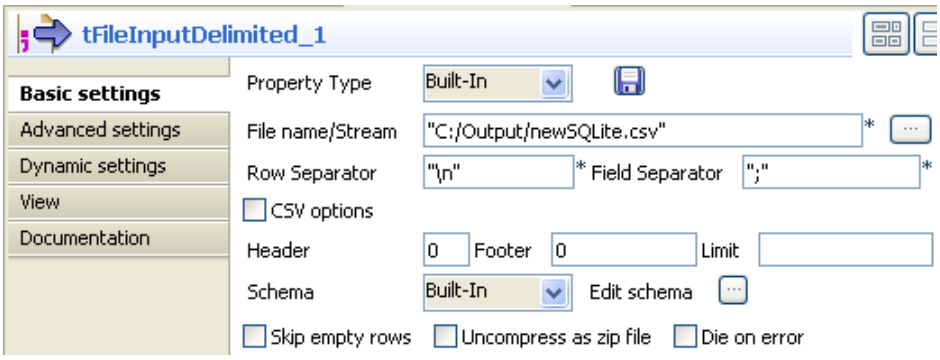
		the column that holds the QUERY's recordset should be set to the type of *Object* and this component is usually followed by **tParseRecordSet**.
	Use PreparedStatement	Select this checkbox if you want to query the database using a PreparedStatement. In the **Set PreparedStatement Parameter** table, define the parameters represented by "?" in the SQL instruction of the **Query** field in the **Basic Settings** tab. **Parameter Index**: Enter the parameter position in the SQL instruction. **Parameter Type**: Enter the parameter type. **Parameter Value**: Enter the parameter value. This option is very useful if you need to execute the same query several times. Performance levels are increased
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStat Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility of the DB query and covers all possible SQL queries.	

Scenario: Updating SQLite rows

This scenario describes a job which updates an SQLite database file based on a prepared statement and using a delimited file.



- Drop a **tFileInputDelimited** and a **tSQLiteRow** component from the **Palette** to the design workspace.
- On the **tFileInputDelimited Basic settings** panel, browse to the input file that will be used to update rows in the database.

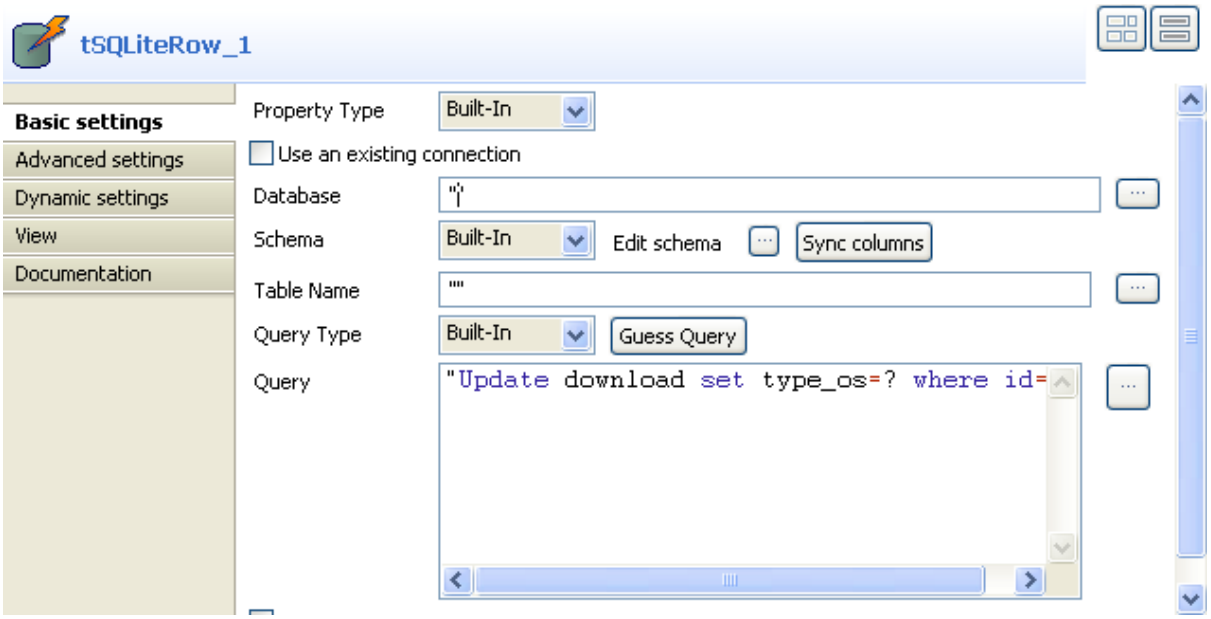


- There is no **Header** nor **Footer**. The **Row separator** is a carriage return and the **Field separator** is a semi-colon.
- Click the **[...]** button next to **Edit schema** and define the schema structure.

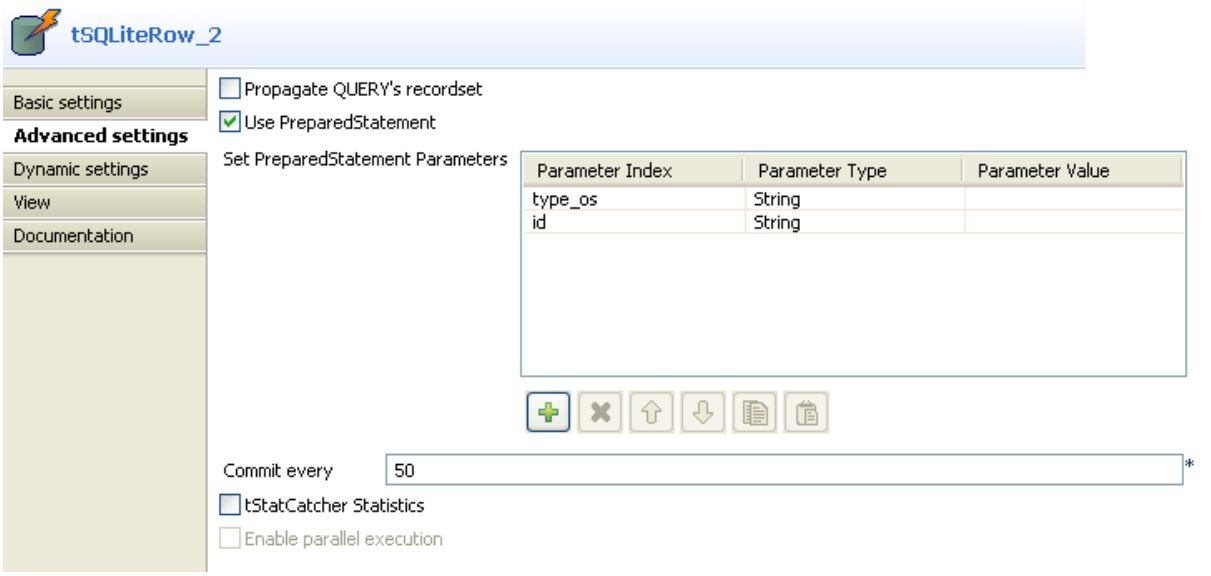
tFileInputDelimited_1

Column	Key	Type	Nullable	Length	Precision	Comment
id	☐	int	☑	6		
version	☐	String	☑	40		
download_date	☐	String	☑	20		
ip	☐	String	☑	20		
type	☐	int	☑	1		
type_os	☐	int	☑	3		

- Make sure the length and type are respectively correct and large enough to define the columns.
- Then in the **tSQLiteRow Basic settings** panel, set the **Database** filepath to the file to be updated.



- The schema is read-only as it is required to match the input schema.
- Type in the query. In this use case, we updated the *type_os* for the *id* defined in the Input flow. The statement is as follows: `“Update download set type_os=? where id=?”`.
- Then select the **Use PreparedStatement** check box to display the placeholders’ parameter table.



- In the Input parameters table, add as many lines as necessary to cover all placeholders. In this scenario, *type_os* and *id* are to be defined.
- Set the **Commit every** field.
- Save the job and press **F6** to run it.

The *download* table from the SQLite database is thus updated with new *type_os* code according to the delimited input file.

Related scenarios

For a related scenario, see:

- section *Scenario 3: Combining two flows for selective output*

DotNET components

This chapter details the main components which you can find in the **DotNET** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The DotNET family comprises the most popular database connectors that are utilized to integrate with .NET objects.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tDotNETInstantiate

tDotNETInstantiate properties

Component family	DotNET	
Function	**tDotNETInstantiate** instantiates an object in the .NET for later reuse.	
Purpose	**tDotNETInstantiate** invokes the constructor of a .NET object that is intended for later reuse.	
Basic settings	*Dll to load*	Type in the path, or browse to the DLL library containing the classe(es) of interest or enter the assembly's name to be used. For example, *System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089* for an OleDb assembly.
	Fully qualified class name(i.e. ClassLibrary1.NameSpace2.Class1)	Enter a fully qualified name for the class of interest.
	Value(s) to pass to the constructor	Click the plus button to add one or more values to be passed to the constructor for the object. Or, leave this table empty to call a default constructor for the object. The valid value(s) should be the parameters required by the class to be used.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component can be used as a start component in a flow or an independent subjob To use this component, you must first install the runtime DLLs, for example *janet-win32.dll* for Windows 32-bit version and *janet-win64.dll* for Windows 64-bit version, from the corresponding Microsoft Visual C++ Redistributable Package. This allows you to avoid errors like the *UnsatisfiedLinkError* on dependent DLL. So ensure that the runtime and all of the other DLLs which the DLL to be called depends on are installed and their versions are consistent among one another. The required DLLs can be installed in the *System32* folder or in the *bin* folder of the Java runtime to be used. If you need to export a Job using this component to run it outside the Studio, you have to specify the runtime container of interest by setting the *-Djava.library.path* argument accordingly.	

Related scenario

For a related scenario, see section *Scenario: Utilizing .NET in Talend*.

tDotNETRow

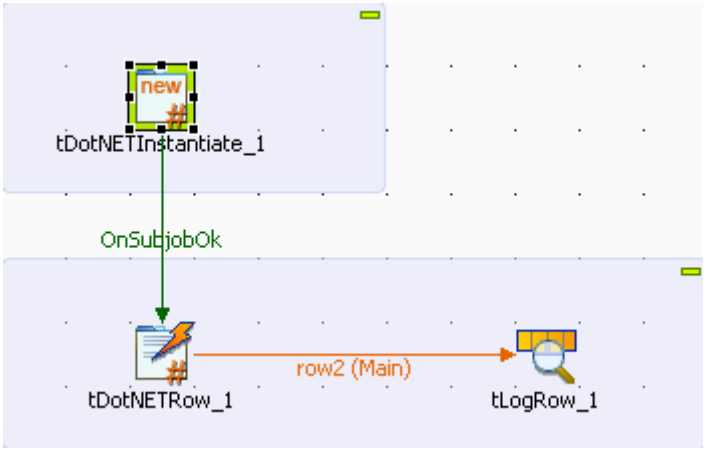
tDotNETRow properties

Component family	DotNET	
Function	**tDotNETRow** sends data to and from libraries and classes within .NET or other custom DLL files.	
Purpose	**tDotNETRow** helps you facilitate data transform by utilizing custom or built-in .NET classes.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Use a static method	Select this check box to invoke a static method in .NET and this will disable **Use an existing intance** check box.
	Propagate a data to output	Select this check box to propagate a transformed data to output.
	Use an existing instance	Select this check box to reuse an existing instance of a .NET object from the **Existing instance to use** list. **Existing instance to use**: Select an existing instance of .NET objects created by the other .NET components from the list. This check box will be disabled if you have selected **Use a static method** and selecting this check box will disable **Dll to load**, **Fully qualified class name(i.e. ClassLibrary1.NameSpace2.Class1)** and **Value(s) to pass to the constructor**.
	Dll to load	Type in the path, or browse to the DLL library containing the class(es) of interest or enter the assembly's name to be used. For example, *System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089* for an OleDb assembly.
	Fully qualified class name(i.e. ClassLibrary1.NameSpace2.Class1)	Enter a fully qualified name for the class of interest.
	Method name	Fill this field with the name of the method to be invoked in .NET.
	Value(s) to pass to the constructor	Click the plus button to add one or more lines for values to be passed to the constructor for the object. Or, leave this table empty to call a default constructor for the object. The valid value(s) should be the parameters required by the class to be used.
	Method Parameters	Click the plus button to add one or more lines for parameters to be passed to the method.
	Output value target column	Select a column in the output row from the list to put value into it.
Advanced settings	*Create a new instance at each row*	Select this check box to create a new instance at each row that passes through the component.
	Method doesn't return a value	Select this check box to invoke a method without returning a value as a result of the processing.
	Returns an instance of a .NET Object	Select this check box to return an instance of a .NET object as a result of a invoked method.
	Store the returned value for later use	Select this check box to store the returned value of a method for later reuse in another **tDotNETRow** component.

	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is utilized to integrate with .NET objects. To use this component, you must first install the runtime DLLs, for example *janet-win32.dll* for Windows 32-bit version and *janet-win64.dll* for Windows 64-bit version, from the corresponding Microsoft Visual C++ Redistributable Package. This allows you to avoid errors like the *UnsatisfiedLinkError* on dependent DLL. So ensure that the runtime and all of the other DLLs which the DLL to be called depends on are installed and their versions are consistent among one another. The required DLLs can be installed in the *System32* folder or in the *bin* folder of the Java runtime to be used. If you need to export a Job using this component to run it outside the Studio, you have to specify the runtime container of interest by setting the *-Djava.library.path* argument accordingly.	

Scenario: Utilizing .NET in Talend

This scenario describes a three-component Job that uses a DLL library containing a class called *Test1.Class1* Class and invokes a method on it that processes the value and output the result onto the console.



Prerequisites

Before replicating this scenario, you need first to build up your runtime environment.

- Create the DLL to be loaded by **tDotNETInstantiate**

  This example class built into .NET reads as follows:

```
using System;
using System.Collections.Generic;
using System.Text;


namespace Test1
{
    public class Class1
    {
        string s = null;
        public Class1(string s)
        {
            this.s = s;
```

```
        }

        public string getValue()
        {
            return "Return Value from Class1: " + s;
        }

    }
    }
```

This class reads the input value and adds the text *Return Value from Class1:* in front of this value. It is compiled using the latest .NET.

- Install the runtime DLL from the latest .NET. In this scenario, we use *janet-win32.dll* on *Windows 32-bit* version and place it in the *System32* folder.

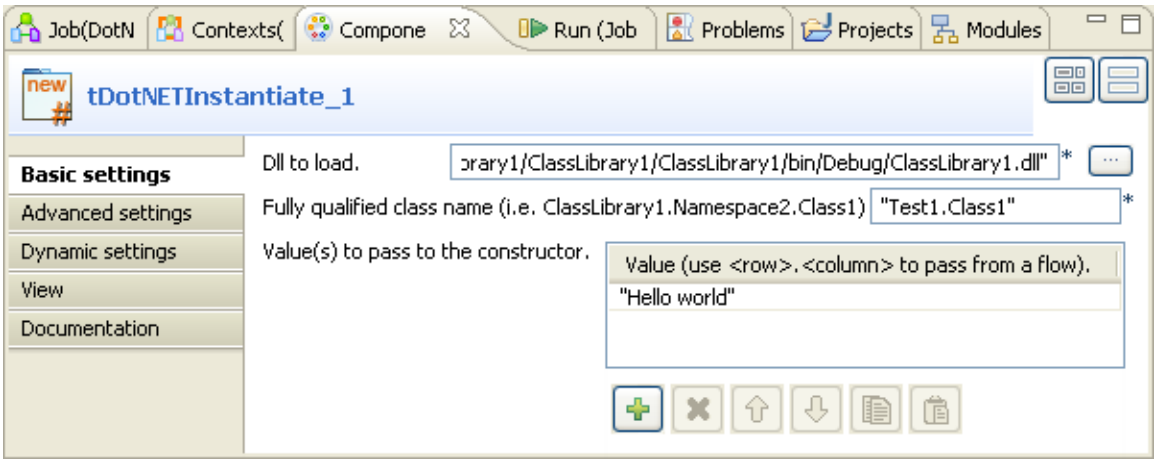
  Thus the runtime DLL is compatible with the DLL to be loaded.

Connecting components

1. Drop the following components from the **Palette** to the design workspace: **tDotNETInstantiate**, **tDotNETRow** and **tLogRow**.
2. Connect **tDotNETInstantiate** to **tDotNETRow** using a **Trigger On Subjob OK** connection.
3. Connect **tDotNETRow** to **tLogRow** using a **Row Main** connection.

Configuring tDotNETInstantiate

1. Double-click **tDotNETInstantiate** to display its **Basic settings** view and define the component properties.



2. Click the three-dot button next to the **Dll to load** field and browse to the DLL file to be loaded. Alternatively, you can fill the field with an assembly. In this example, we use :

   "C:/Program Files/ClassLibrary1/bin/Debug/ClassLibrary1.dll""

3. Fill the **Fully qualified class name** field with a valid class name to be used. In this example, we use:

   "Test1.Class1"

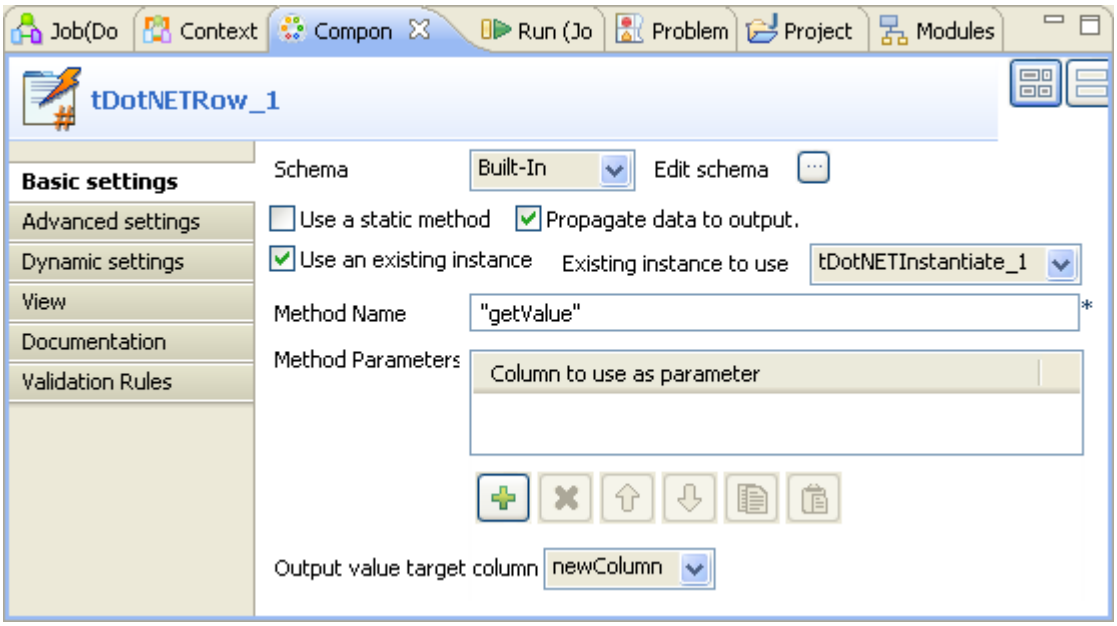
4. Click the plus button beneath the **Value(s) to pass to the constructor** table to add a new line for the value to be passed to the constructor.

In this example, we use:

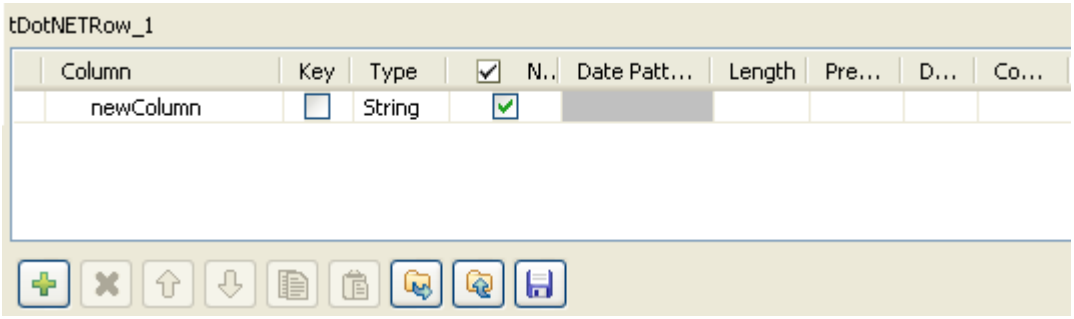
```
"Hello world"
```

Configuring tDotNETRow

1. Double-click **tDotNETRow** to display its **Basic settings** view and define the component properties.



2. Select **Propagate data to output** check box.
3. Select **Use an existing instance** check box and select **tDotNETInstantiate_1** from the **Existing instance to use** list on the right.
4. Fill the **Method Name** field with a method name to be used. In this example, we use *"getValue"*, a custom method.
5. Click the three-dot button next to **Edit schema** to add one column to the schema.

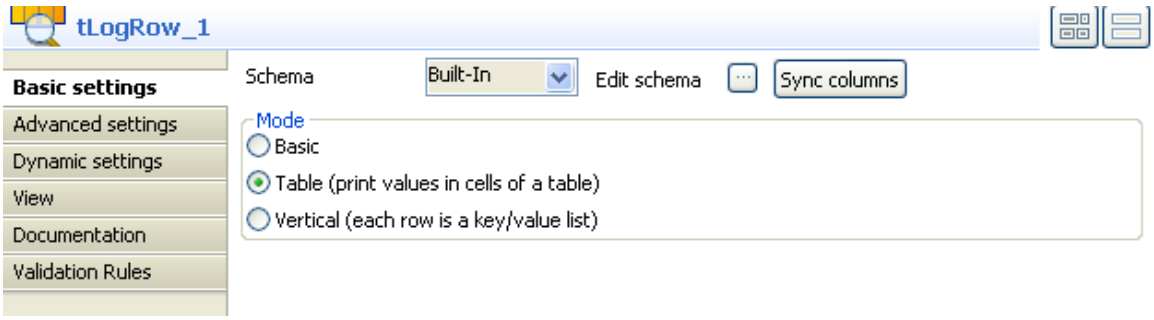


Click the plus button beneath the table to add a new column to the schema and click **OK** to save the setting.

6. Select **newColumn** from the **Output value target column** list.

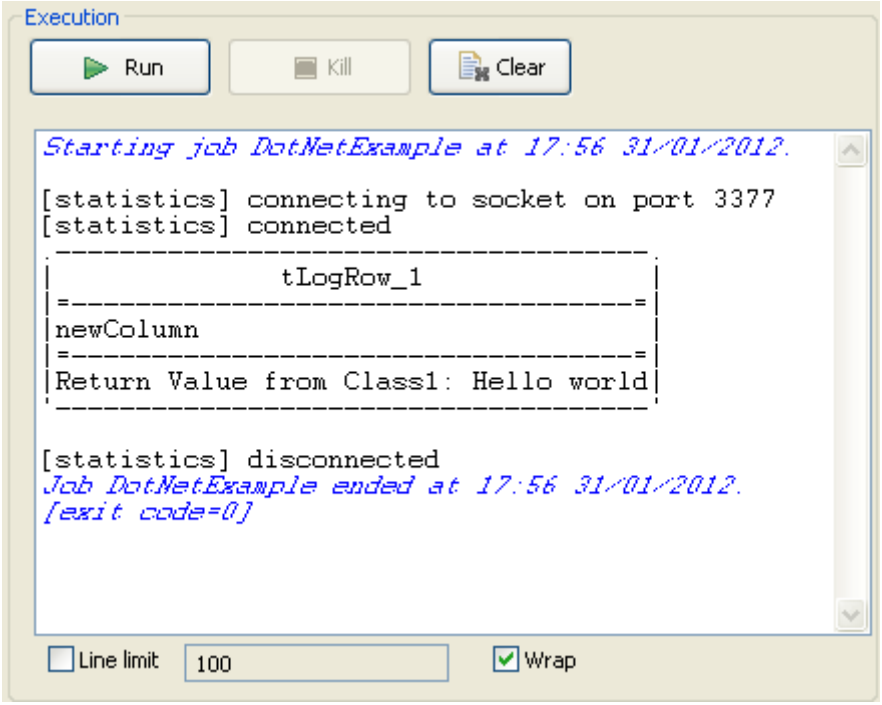
Configuring tLogRow

1. Double-click **tLogRow** to display its **Basic settings** view and define the component properties.



2. Click **Sync columns** button to retrieve the schema defined in the preceding component.
3. Select **Table** in the **Mode** area.

Save your Job and press **F6** to execute it.



From the result, you can read that the text `Return Value from Class1` is added in front of the retrieved value `Hello world`.

ELT components

This chapter details the main components that you can find in the **ELT** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The ELT family groups together the most popular database connectors and processing components, all dedicated to the ELT mode where the target DBMS becomes the transformation engine.

This mode supports all of the most popular databases including Teradata, Oracle, Vertica, Netezza, Sybase, etc.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tCombinedSQLAggregate

tCombinedSQLAggregate properties

Component family	ELT/CombinedSQL	
Function	**tCombinedSQLAggregate** collects data values from one or more columns of a table for statistical purposes. This component has real-time capabilities since it runs the data transformation on the DBMS itself.	
Purpose	Helps to provide a set of matrix based on values or calculations.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered according to the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations.
		Input Column: Select the input column label to match the output column's expected content, in case the output label of the aggregation set needs to be different.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select any of the following operations to perform on data: **count**, **min**, **max**, **avg**, **sum**, **first**, **last**, **distinct** and **count (distinct)**.
		Input column: Select the input column from which you want to collect the values to be aggregated.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is an intermediary component. The use of the corresponding connection and commit components is recommended when using this component to allow a unique connection to be open and then closed during the Job execution.	
Limitation	n/a	

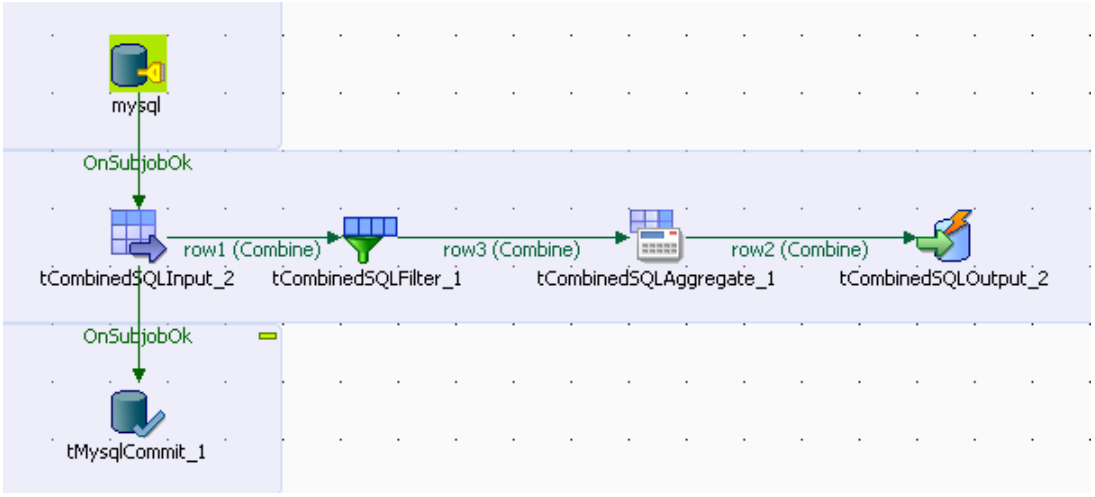
Scenario: Filtering and aggregating table columns directly on the DBMS

The following scenario creates a Job that opens a connection to a MySQL database and:

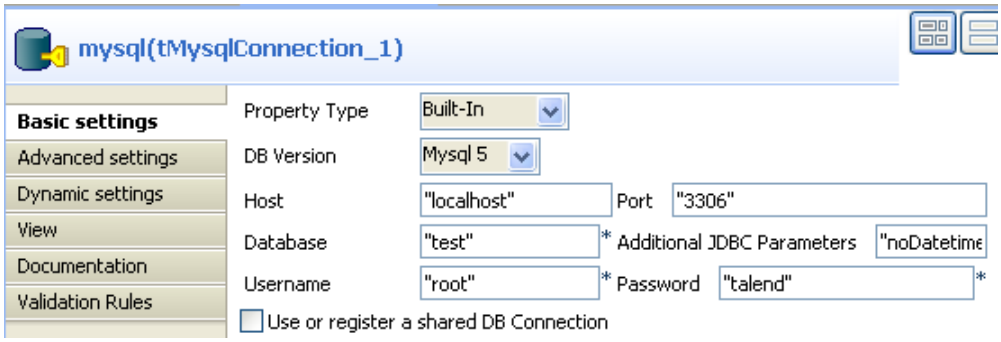
- instantiates the schema from a database table in part (for column filtering),
- filters two columns in the same table to get only the data that meets two filtering conditions,
- collects data from the filtered column(s), grouped by specific value(s) and writes aggregated data in a target database table.

To filter and aggregate database table columns:

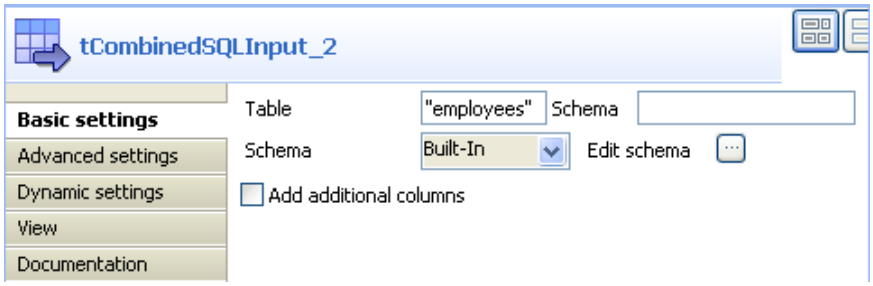
- Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tCombinedSQLInput**, **tCombinedSQLFilter**, **tCombinedSQLAggregate**, **tCombinedSQLOutput** and **tMysqlCommit**.
- Connect **tMysqlConnection**, **tCombinedSQLInput** and **tMysqlCommit** using **OnSubjobOk** links.
- Connect **tCombinedSQLInput**, **tCombinedSQLFilter**, **tCombinedSQLAggregate** and **tCombinedSQLOutput** using a **Combine** link.



- In the design workspace, select **tMysqlConnection** and click the **Component** tab to define its basic settings.
- In the **Basic settings** view, set the database connection details manually.



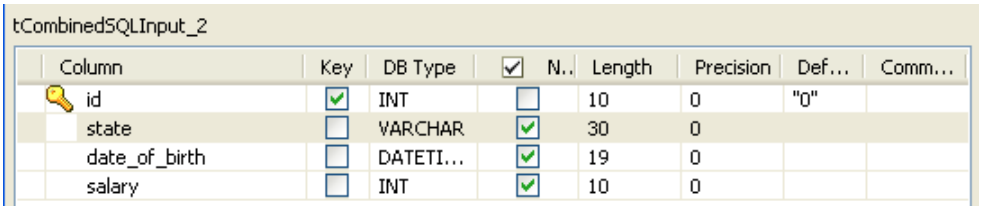
- In the design workspace, select **tCombinedSQLInput** and click the **Component** tab to access the configuration panel.



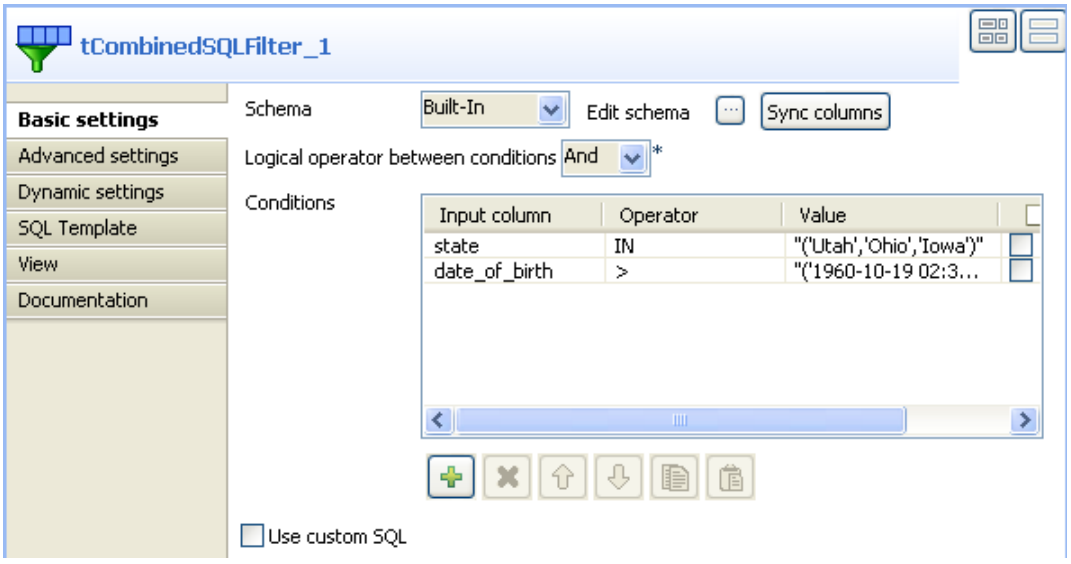
- Enter the source table name in the **Table** field, and click the three-dot button next to **Edit schema** to define the data structure.

The schema defined through **tCombinedSQLInput** can be different from that of the source table as you can just instantiate the desired columns of the source table. Therefore, **tCombinedSQLInput** also plays a role of column filtering.

In this scenario, the source database table has seven columns: *id*, *first_name*, *last_name*, *city*, *state*, *date_of_birth*, and *salary* while **tCombinedSQLInput** only instantiates four columns that are needed for the aggregation: *id*, *state*, *date_of_birth*, and *salary* from the source table.



- In the design workspace, select **tCombinedSQLFilter** and click the **Component** tab to access the configuration panel.



- Click the **Sync columns** button to retrieve the schema from the previous component, or configure the schema manually by selecting **Built-in** from the **Schema** list and clicking the **[...]** button next to **Edit schema**.

When you define the data structure for **tCombinedSQLFilter**, column names automatically appear in the **Input column** list in the **Conditions** table.

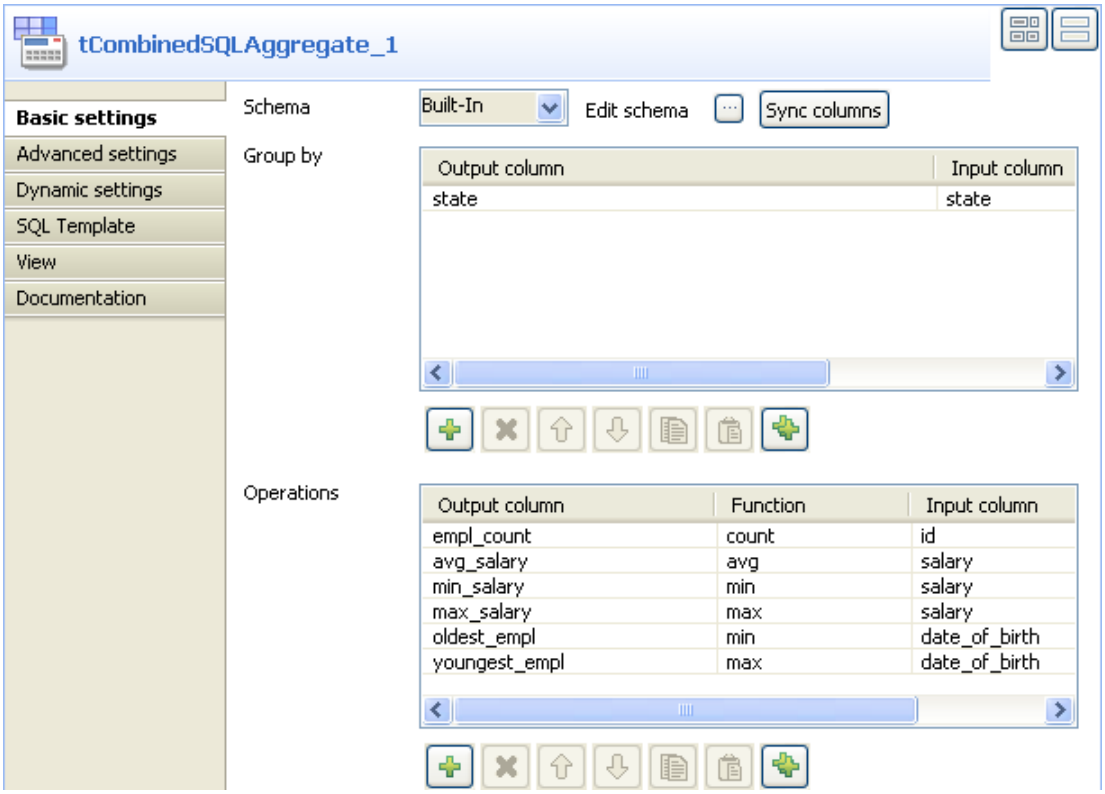
In this scenario, the **tCombinedSQLFilter** component instantiates four columns: *id*, *state*, *date_of_birth*, and *salary*.

- In the **Conditions** table, set input parameters, operators and expected values in order to only extract the records that fulfill these criteria.

In this scenario, the **tCombinedSQLFilter** component filters the *state* and *date_of_birth* columns in the source table to extract the employees who were born after Oct. 19, 1960 and who live in the states *Utah*, *Ohio* and *Iowa*.

- Select **And** in the **Logical operator between conditions** list to apply the two conditions at the same time. You can also customize the conditions by selecting the **Use custom SQL** box and editing the conditions in the code box.
- In the design workspace, select **tCombinedSQLAggregate** and click the **Component** tab to access the configuration panel.
- Click the **Sync columns** button to retrieve the schema from the previous component, or configure the schema manually by selecting **Built-in** from the **Schema** list and clicking on the **[...]** button.

The **tCombinedSQLAggregate** component instantiates four columns: *id*, *state*, *date_of_birth*, and *salary*, coming from the previous component.

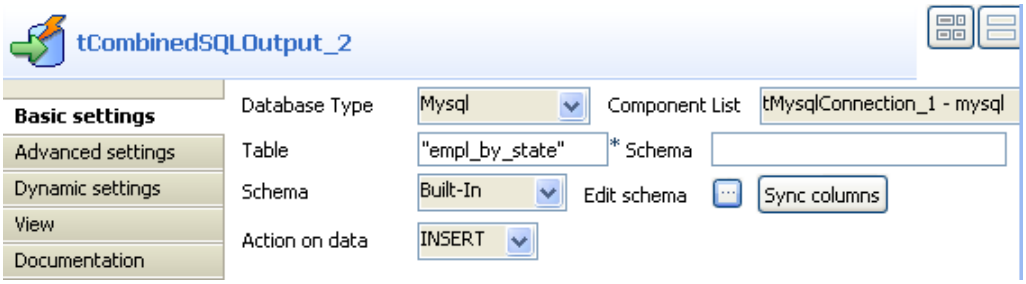


The **Group by** table helps you define the data sets to be processed based on a defined column. In this example: *State*.

- In the **Group by** table, click the **[+]** button to add one line.
- In the **Output column** drop-down list, select *State*. This column will be used to hold the data filtered on *State*.

The **Operations** table helps you define the type of aggregation operations to be performed. The **Output column** list available depends on the schema you want to output (through the **tCombinedSQLOutput** component). In this scenario, we want to group employees based on the state they live. We want then count the number of employees per state, calculate the average/lowest/highest salaries as well as the oldest/youngest employees for each state.

- In the **Operations** table, click the **[+]** button to add one line and then click in the **Output column** list to select the output column that will hold the computed data.
- In the **Function** field, select the relevant operation to be carried out.
- In the design workspace, select **tCombinedSQLOutput** and click the **Component** tab to access the configuration panel.



- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- In the **Table** field, enter the name of the target table which will store the results of the aggregation operations.

In this example, the **Schema** field doesn't need to be filled out as the database is not Oracle.

- Click the three-dot button next to **Edit schema** to define the data structure of the target table.

In this scenario, **tCombinedSQLOutput** instantiates seven columns coming from the previous component in the Job design (**tCombinedSQLAggregate**): *state*, *empl_count*, *avg_salary*, *min_salary*, *max_salary*, *oldest_empl* and *youngest_empl*.

- In the design workspace, select **tCombinedSQLCommit** and click the **Component** tab to access the configuration panel.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Save your Job and press **F6** to execute it.

Rows are inserted into a seven-column table *empl_by_state* in the database. The table shows, per defined state, the number of employees, the average salary, the lowest and highest salaries as well as the oldest and youngest employees.

	state	empl_count	avg_salary	min_salary	max_salary	oldest_empl	youngest_empl
☐	Utah	3	10167	5087	16180	1961-10-19 02	1986-12-31 18:21
☐	Ohio	3	10646	6269	19380	1968-10-19 02	1975-06-27 01:08
☐	Iowa	2	13149	8118	18180	1972-10-15 04	1975-06-27 01:08

tCombinedSQLFilter

tCombinedSQLFilter Properties

Component family	ELT/CombinedSQL	
Function	**tCombinedSQLFilter** allows you to alter the schema of a source table through column name mapping and to define a row filter on that table. Therefore, it can be used to filter columns and rows at the same time. This component has real-time capabilities since it runs the data filtering on the DBMS itself.	
Purpose	Helps to filter data by reorganizing, deleting or adding columns based on the source table and to filter the given data source using the filter conditions.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Logical operator between conditions	Select the logical operator between the filter conditions defined in the **Conditions** panel. Two operators are available: **Or**, **And**.
	Conditions	Select the type of WHERE clause along with the values and the columns to use for row filtering.
		Input Column: Select the column to filter in the list.
		Operator: Select the type of the WHERE clause: =, < >, >, <, >=, <=, **LIKE**, **IN**, **NOT IN**, and **EXIST IN**.
		Values: Type in the values to be used in the WHERE clause.
		Negate: Select this check box to enable the condition that is opposite to the current setting.
	Use custom SQL	Customize a WHERE clause by selecting this check box and editing in the **SQL Condition** field.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is an intermediary component. The use of the corresponding connection and commit components is recommended when using this component to allow a unique connection to be open and then closed during the Job execution.	
Limitation	n/a	

Related Scenario

For a related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tCombinedSQLInput

tCombinedSQLInput properties

Component family	ELT/CombinedSQL	
Function	**tCombinedSQLInput** extracts fields from a database table based on its schema. This component also has column filtering capabilities since its schema can be different from that of the database table.	
Purpose	**tCombinedSQLInput** extracts fields from a database table based on its schema definition. Then it passes on the field list to the next component via a Combine row link. The schema of **tCombinedSQLInput** can be different from that of the source database table but must correspond to it in terms of the column order.	
Basic settings	*Table*	Name of the source database table.
	Schema	Name of the source table's schema. This field has to be filled if the database is Oracle.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Add additional columns	This option allows you to call SQL functions to perform actions on columns, provided that these are not insert, update or delete actions, or actions that require pre-processing.
		Name: Type in the name of the schema column to be altered.
		SQL expression: Type in the SQL statement to be executed in order to alter the data in the corresponding column.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is an intermediary component. The use of the corresponding connection and commit components is recommended when using this component to allow a unique connection to be open and then closed during the Job execution.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tCombinedSQLOutput

tCombinedSQLOutput properties

Component family	ELT/CombinedSQL	
Function	**tCombinedSQLOutput** inserts records to an existing database table.	
Purpose	**tCombinedSQLOutput** inserts records from the incoming flow to an existing database table.	
Basic settings	*Database Type*	Select the database type.
	Component list	Select the relevant DB connection component in the list if more than one connection is used for the current Job.
	Table	Name of the target database table.
	Schema	Name of the target database table's schema. This field has to be filled if the database is Oracle.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Action on data	Select **INSERT** from the list to insert the records from the incoming flow to the target database table.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is an intermediary component. The use of the corresponding connection and commit components is recommended when using this component to allow a unique connection to be open and then closed during the Job execution.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tELTGreenplumInput

tELTGreenplumInput properties

The three ELT Greenplum components are closely related, in terms of their operating conditions. These components should be used to handle Greenplum DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Greenplum	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. For *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Type in the default table name.
	Default Schema Name	Type in the default schema name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTGreenplumInput** is to be used along with the **tELTGreenplumMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTGreenplumInput**, see:

- section *Scenario: Mapping data using a simple implicit join*
- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTGreenplumMap

tELTGreenplumMap properties

The three ELT Greenplum components are closely related, in terms of their operating conditions. These components should be used to handle Greenplum DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Greenplum	
Function	Helps you to build the SQL statement graphically, using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Greenplum Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	For *Talend Open Studio for Big Data*, the property type is always **Built-in**.

		Built-in: No property data stored centrally.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTGreenplumMap** is used along with **tELTGreenplumInput** and **tELTGreenplumOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Scenario: Mapping data using a simple implicit join

In this scenario, a **tELTGreenplumMap** component is deployed to retrieve the data from the source table *employee_by_statecode*, compares its *statecode* column against the table *statecode*, and then maps the desired columns from the two tables to the output table *employee_by_state*.

Before the Job execution, the three tables, *employee_by_statecode*, *statecode* and *employee_by_state* look like:

id	name	statecode
1	Andy	86
3	Mike	39
2	John	52

state	statecode
China	86
Mexico	52
Italy	39

id	name	state

Dropping components

1. Drop **tGreenplumConnection**, **tELTGreenplumInput** (X2), **tELTGreenplumMap**, **tELTGreenplumOutput**, **tGreenplumCommit**, **tGreenplumInput** and **tLogRow** from the **Palette** onto the workspace.

2. Rename **tGreenplumConnection** as **connect_to_greenplum_host**, two **tELTGreenplumInput** components as **employee+statecode** and **statecode**, **tELTGreenplumMap** as **match +map**, **tELTGreenplumOutput** as **map_data_output**, **tGreenplumCommit** as **commit_to_host**, **tGreenplumInput** as **read_map_output_table** and **tLogRow** as **show_map_data**.

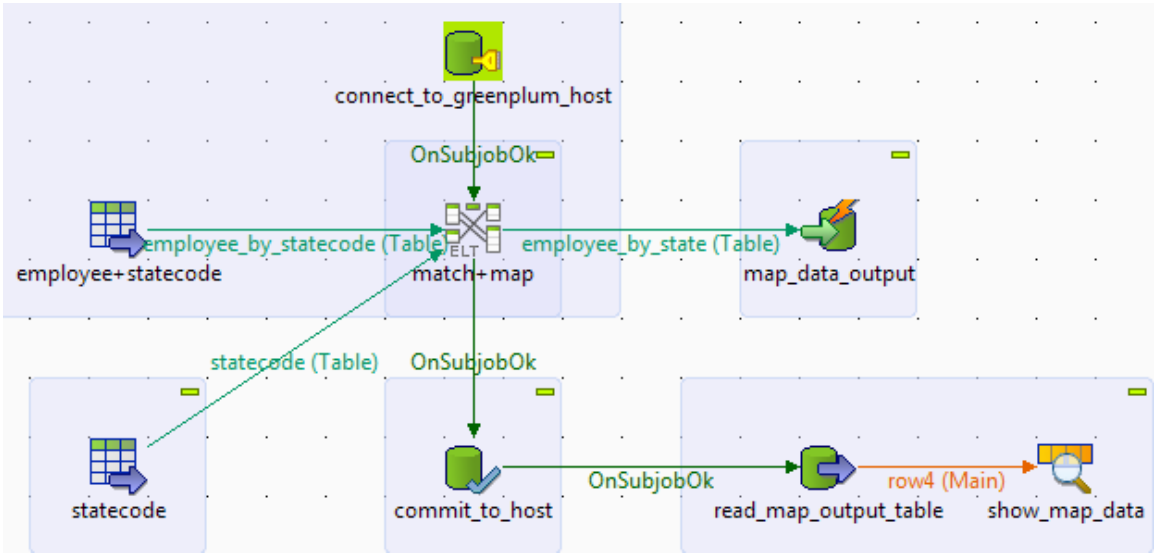
3. Link **tGreenplumConnection** to **tELTGreenplumMap** using an **OnSubjobOk** trigger.

   Link **tELTGreenplumMap** to **tGreenplumCommit** using an **OnSubjobOk** trigger.

   Link **tGreenplumCommit** to **tGreenplumInput** using an **OnSubjobOk** trigger.

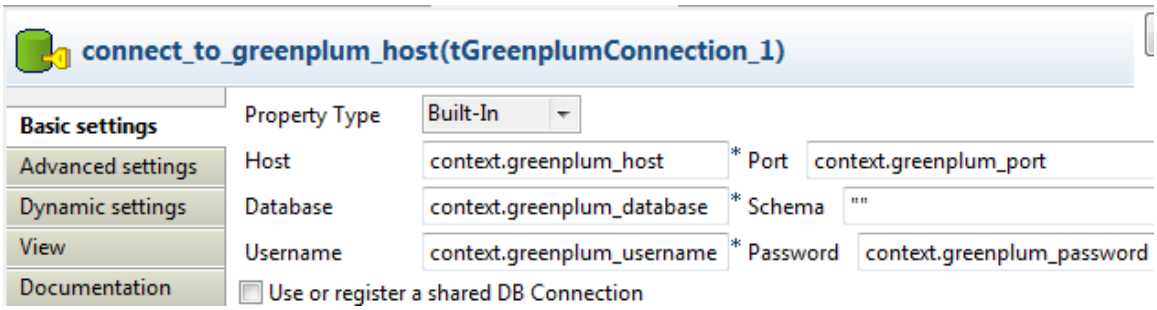
4. Link **tGreenplumInput** to **tLogRow** using a **Row > Main** connection.

   The two **tELTGreenplumInput** components and **tELTGreenplumOutput** will be linked to **tELTGreenplumMap** later once the relevant tables have been defined.



Configuring the components

1. Double-click **tGreenplumConnection** to open its **Basic settings** view in the **Component** tab.



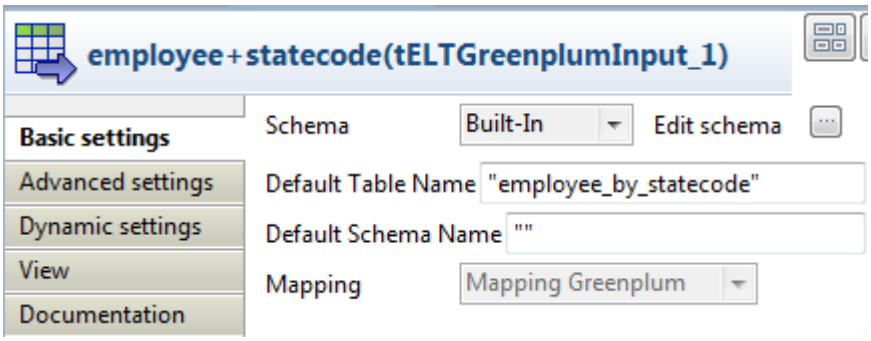
   In the **Host** and **Port** fields, enter the context variables for the Greenplum server.

   In the **Database** field, enter the context variable for the Greenplum database.

   In the **Username** and **Password** fields, enter the context variables for the authentication credentials.

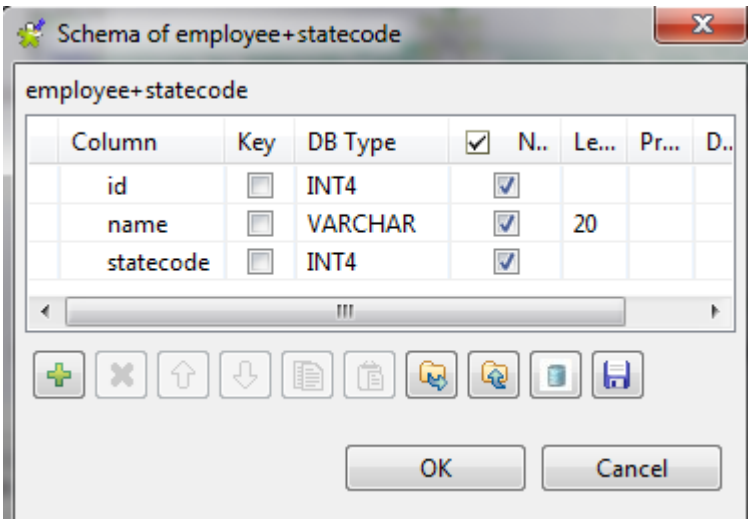
   For more information on context variables, see the *Talend Open Studio for Big Data User Guide*.

2. Double-click **employee+statecode** to open its **Basic settings** view in the **Component** tab.



   In the **Default table name** field, enter the name of the source table, namely *employee_by_statecode*.

   Click the **[...]** button next to the **Edit schema** field to open the schema editor.

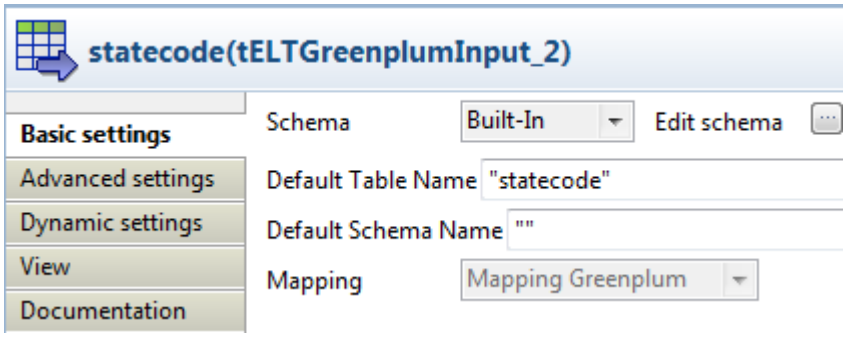


Click the **[+]** button to add three columns, namely *id*, *name* and *statecode*, with the data type as *INT4*, *VARCHAR*, and *INT4* respectively.

Click **OK** to close the schema editor.

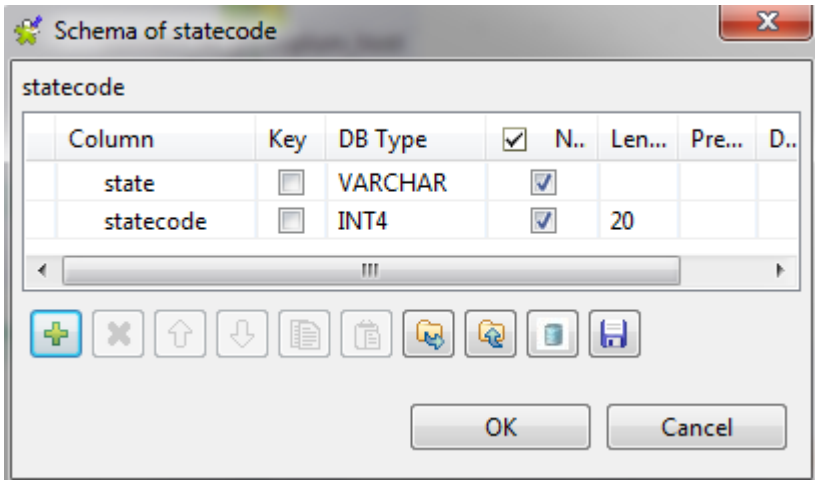
Link **employee+statecode** to **tELTGreenplumMap** using the output *employee_by_statecode*.

3. Double-click **statecode** to open its **Basic settings** view in the **Component** tab.



In the **Default table name** field, enter the name of the lookup table, namely *statecode*.

4. Click the **[...]** button next to the **Edit schema** field to open the schema editor.

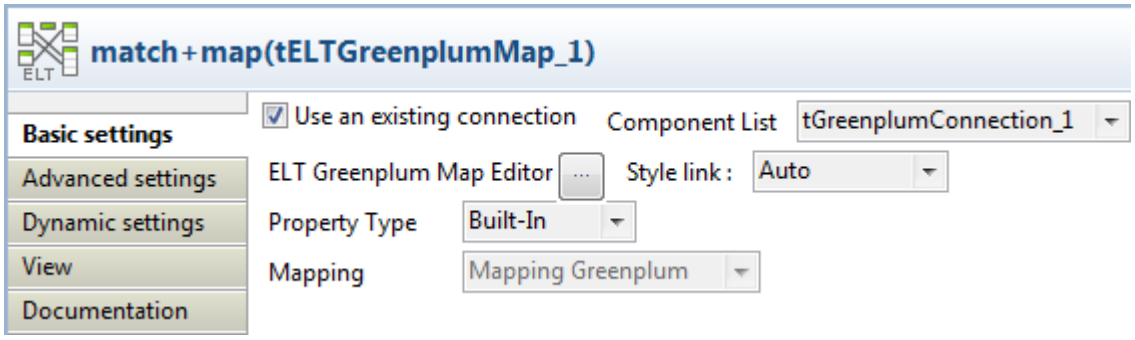


Click the **[+]** button to add two columns, namely *state* and *statecode*, with the data type as *VARCHAR* and *INT4* respectively.

Click **OK** to close the schema editor.

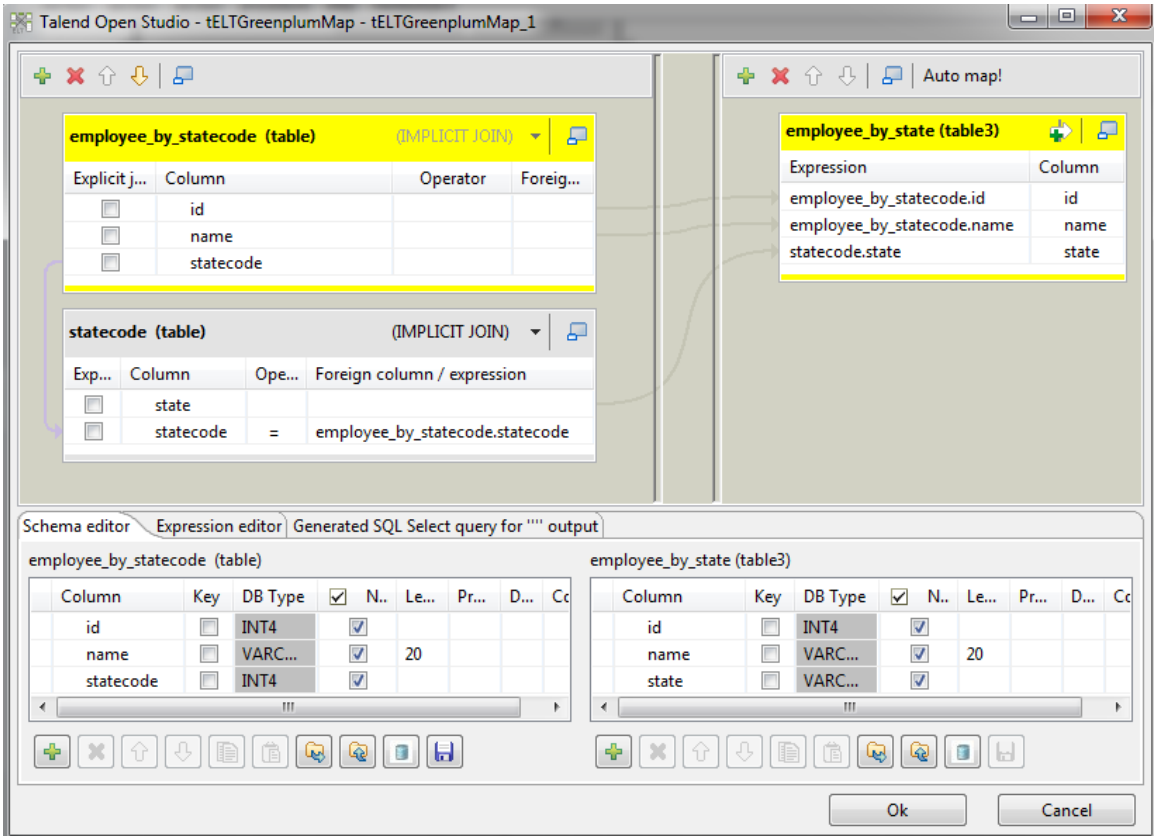
Link **statecode** to **tELTGreenplumMap** using the output *statecode*.

5. Click **tELTGreenplumMap** to open its **Basic settings** view in the **Component** tab.

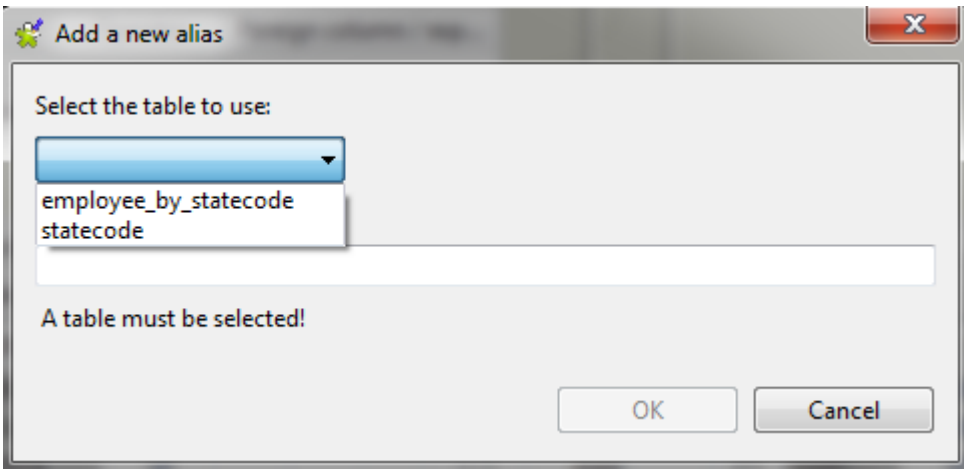


Select the **Use an existing connection** check box.

6. Click the **[...]** button next to the **ELT Greenplum Map Editor** field to open the map editor.



7. Click the **[+]** button on the upper left corner to open the table selection box.



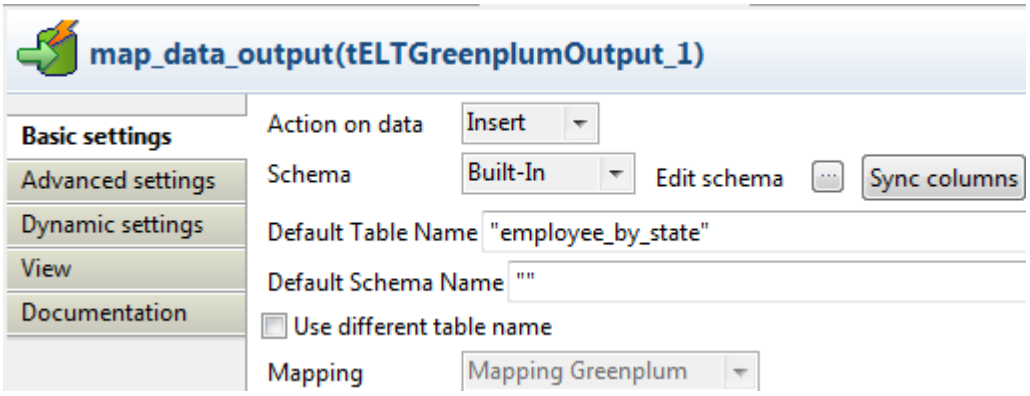
Select tables *employee_by_statecode* and *statecode* in sequence and click **Ok**.

The tables appear on the left panel of the editor.

8. On the upper right corner, click the **[+]** button to add an output table, namely *employee_by_state*.

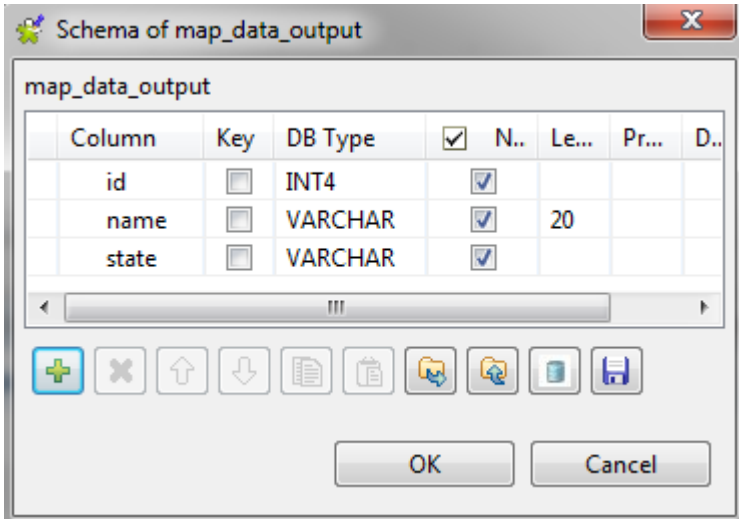
   Click **Ok** to close the map editor.

9. Double-click **tELTGreenplumOutput** to open its **Basic settings** view in the **Component** tab.



   In the **Default table name** field, enter the name of the output table, namely *employee_by_state*.

10. Click the **[...]** button next to the **Edit schema** field to open the schema editor.

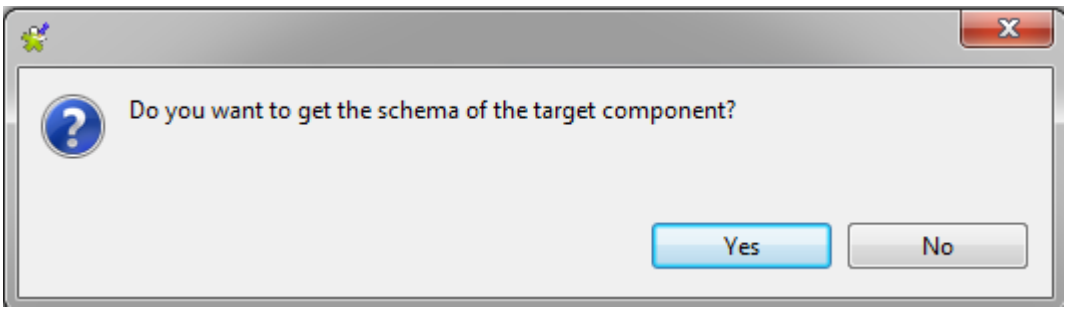


Click the **[+]** button to add three columns, namely *id*, *name* and *state*, with the data type as *INT4*, *VARCHAR*, and *VARCHAR* respectively.

Click **OK** to close the schema editor.

Link **tELTGreenplumMap** to **tELTGreenplumOutput** using the table output *employee_by_state*.

Click **OK** on the pop-up window below to retrieve the schema of **tELTGreenplumOutput**.



Now the map editor's output table *employee_by_state* shares the same schema as that of **tELTGreenplumOutput**.

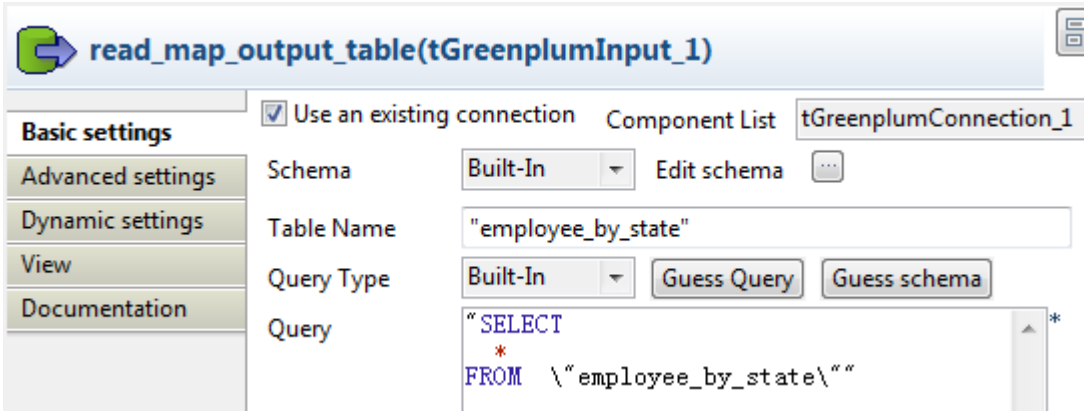
11. Double-click **tELTGreenplumMap** to open the map editor.

    Drop the column *statecode* from table *employee_by_statecode* to its counterpart of the table *statecode*, looking for the records in the two tables that have the same *statecode* values.

    Drop the columns *id* and *name* from table *employee_by_statecode* as well as the column *statecode* from table *statecode* to their counterparts in the output table *employee_by_state*.

    Click **Ok** to close the map editor.

12. Double-click **tGreenplumInput** to open its **Basic settings** view in the **Component** tab.

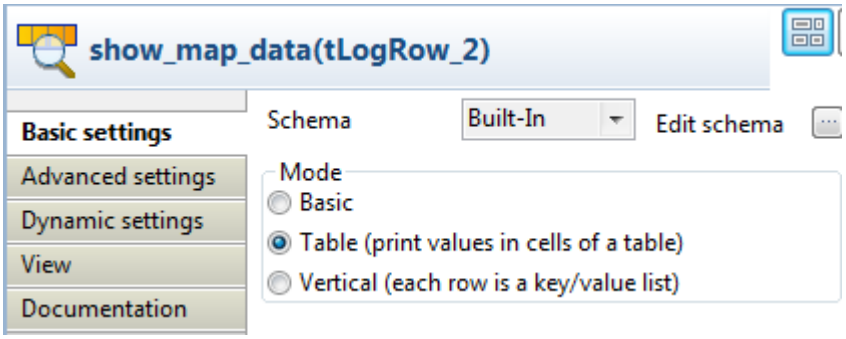


    Select the **Use an existing connection** check box.

    In the **Table name** field, enter the name of the source table, namely *employee_by_state*.

    In the **Query** field, enter the query statement, namely `"SELECT * FROM \"employee_by_state\""`.

13. Double-click **tLogRow** to open its **Basic settings** view in the **Component** tab.



In the **Mode** area, select **Table (print values in cells of a table** for a better display.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to run the Job.

```
-->> 3 rows inserted.
.--+----+------.
|show_map_data  |
|=-+----+------=|
|id|name|state  |
|=-+----+------=|
|2 |John|Mexico|
|1 |Andy|China  |
|3 |Mike|Italy  |
'--+----+------'

[statistics] disconnected
```

As shown above, the desired employee records have been written to the table *employee_by_state*, presenting clearer geographical information about the employees.

Related scenario:

For related scenarios, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTGreenplumOutput

tELTGreenplumOutput properties

The three ELT Greenplum components are closely related, in terms of their operating conditions. These components should be used to handle Greenplum DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Greenplum	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Greenplum database	
Basic settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Adds new entries to the table. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. For *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name, between double quotation marks.
	Default Schema Name	Enter the default schema name,between double quotation marks.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTGreenplumOutput** is to be used along with the **tELTGreenplumMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTGreenplumOutput**, see:

- section *Scenario: Mapping data using a simple implicit join*
- section *Scenario 1: Aggregating table columns and filtering*

- section *Scenario 2: ELT using an Alias table*

tELTHiveInput

tELTHiveInput properties

The three ELT Hive components are closely related, in terms of their operating conditions. These components should be used to handle Hive DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Hive	
Function	This component provides, for the **tELTHiveMap** component that follows, the input schema of the Hive table to be used.	
Purpose	This component helps to replicate the schema, which the **tELTHiveMap** component that follows will use, of the input Hive table.	
Basic settings	*Schema*	**Built-in**: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and Job designs. Related topic: see the user guide of the Studio for integration. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Edit schema	Click **Edit Schema** to make changes to the schema.
	Default table name	Enter the default name of the input table to be used.
Advanced settings	*tStat Catcher Statistics*	Select this check box to collect log data at the component level.
Usage	**tELTHiveMap** is used along with a **tELTHiveInput** and **tELTHiveOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed. The ELT components do not handle actual data flow but only schema information.	

Related scenario

For a related scenario, see section *Scenario: Joining table columns and writing them into Hive*

tELTHiveMap

tELTHiveMap properties

The three ELT Hive components are closely related, in terms of their operating conditions. These components should be used to handle Hive DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Hive	
Function	This component uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Purpose	This component helps to graphically build the Hive QL statement in order to transform data.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to reuse connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Hive Map editor	The ELT Map editor helps you to define the output schema as well as build graphically the Hive QL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
Version	*Distribution*	Select the product you are using as the Hadoop distribution from the drop-down list.

		The options in the list vary depending on the component you are using.
	Hive version	Select the version of the Hadoop distribution you are using.
	Connection mode	Select a connection mode from the list. The options vary depending on the distribution you are using.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database. According to the documentation of Hive, the only database name supported is *default*. For further information, see https://cwiki.apache.org/Hive/hiveclient.html.
	Username and *Password*	DB user authentication data.
Hadoop properties	*Set Jobtracker URI*	Select this check box to indicate the location of the Hadoop Jobtracker service to be used. For example, we assume that you have chosen a machine called *machine1* as the JobTracker, then set its location as *machine1:portnumber*. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
	Set NameNode URI	Select this check box to indicate the location of the NameNode of the Hadoop cluster to be used. This property is required when the query you want to use is executed in Windows and it is a `Select` query. For example, `SELECT your_column_name FROM your_table_name` You can keep this check box clear, when a query is executed in Windows but does not require the Hadoop Map/Reduce framework, for example, `select * from your_table_name`; however, setting the exact value for this property does not provoke any issues with the execution in that situation. For further information about the Hadoop Map/Reduce framework, see the Map/Reduce tutorial in the Hadoop documentation.
Advanced settings	*Hadoop properties*	If you need to use custom configuration for the Hadoop of interest, complete this table with the property or properties to be customized. Then at runtime, the customized property or properties will override those corresponding ones defined earlier for the same Hadoop. For further information about the properties required by Hadoop, see the Hadoop documentation.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
	Temporary path	If you do not want to set the Jobtracker and the NameNode when you execute the query `select * from your_table_name`, you need to set this temporary path. For example, */C:/select_all* in Windows.
Usage	**tELTHiveMap** is used along with a **tELTHiveInput** and **tELTHiveOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed. The ELT components do not handle actual data flow but only schema information.	

Connecting Hive ELT components

The ELT components do not handle any data as such but table schema information that will be used to build the Hive QL query to execute.

Therefore the only connection required to connect these components together is a simple link.

The output name you give to this link when creating it should always be the exact name of the table to be accessed as this parameter will be used in the Hive QL statement generated.

Related topic: see *Talend Open Studio for Big Data User Guide*.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

- As you would do it in the regular Map editor, simply drag & drop the content from the input schema towards the output table defined.
- Use the **Ctrl** and **Shift** keys for multiple selection of contiguous or non contiguous table columns.

You can implement explicit joins to retrieve various data from different tables.

- Select the **Explicit join** check box for the relevant column, and select a type of join from the **Join** list.
- Possible joins include: **Inner Join**, **Left Outer Join**, **Right Outer Join** or **Full Outer Join** and **Cross Join**.
- By default the **Inner Join** is selected.

You can also create **Alias** tables to retrieve various data from the same table.

- In the Input area, click the button to create an Alias.
- Define the table to base the alias on.
- Type in a new name for the alias table, preferably not the same as the main table.

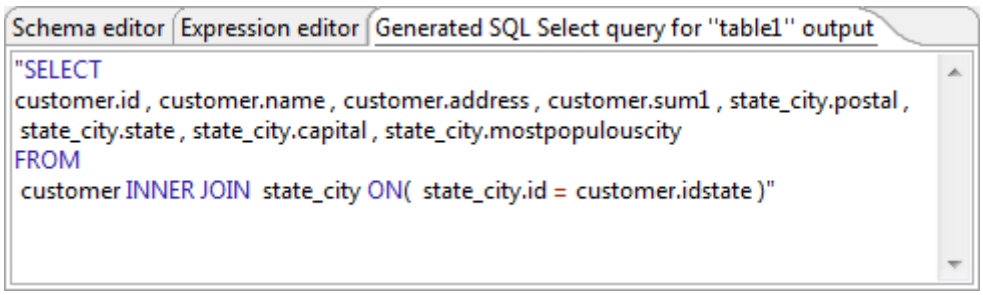
Adding where clauses

You can also restrict the Select statement based on a Where clause. Click the **Add filter row** button at the top of the output table and type in the relevant restriction to be applied.

Ensure that all input components are linked correctly to the ELT Map component to be able to implement all inclusions, joins and clauses.

Generating the Hive QL statement

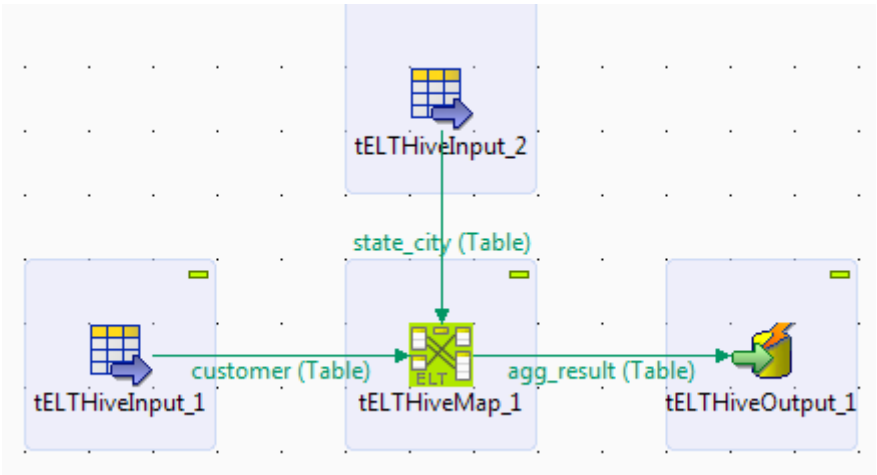
The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement. For example,



The clause are also included automatically.

Scenario: Joining table columns and writing them into Hive

This scenario uses a four-component Job to join the columns selected from two Hive tables and write them into another Hive table.



Preparing the Hive tables

1. Create the Hive table you want to write data in. In this scenario, this table is named as *agg_result*, and you can create it using the following statement in **tHiveRow**:

```
create table agg_result (id int, name string, address string, sum1 string, postal
 string, state string, capital string, mostpopulouscity string) partitioned by (type
 string) row format delimited fields terminated by ';' location '/user/ychen/hive/
table/agg_result'
```

   In this statement, *'/user/ychen/hive/table/agg_result'* is the directory used in this scenario to store this created table in HDFS. You need to replace it with the directory you want to use in your environment.

   For further information about **tHiveRow**, see section *tHiveRow*.

2. Create two input Hive tables containing the columns you want to join and aggregate these columns into the output Hive table, *agg_result*. The statements to be used are:

```
create table customer (id int, name string, address string, idState int, id2 int,
 regTime string, registerTime string, sum1 string, sum2 string) row format delimited
 fields terminated by ';' location '/user/ychen/hive/table/customer'
```

and

```
create table state_city (id int, postal string, state string, capital int,
 mostpopulouscity string) row format delimited fields terminated by ';' location '/
user/ychen/hive/table/state_city'
```

3. Use **tHiveRow** to load data into the two input tables, *customer* and *state_city*. The statements to be used are:

```
"LOAD DATA LOCAL INPATH 'C:/tmp/customer.csv' OVERWRITE INTO TABLE customer"
```

and

```
"LOAD DATA LOCAL INPATH 'C:/tmp/State_City.csv' OVERWRITE INTO TABLE state_city"
```

The two files, *customer.csv* and *State_City.csv*, are two local files we created for this scenario. You need to create your own files to provide data to the input Hive tables. The data schema of each file should be identical with their corresponding table.

You can use **tRowGenerator** and **tFileOutputDelimited** to create these two files easily. For further information about these two components, see section *tRowGenerator* and section *tFileOutputDelimited*.

For further information about the Hive query language, see https://cwiki.apache.org/Hive/languagemanual.html.

Linking the components

1. In the **Integration** perspective of the Studio, create an empty Job from the **Job Designs** node in the **Repository** tree view.

   For further information about how to create a Job, see the user guide of the Studio for integration.

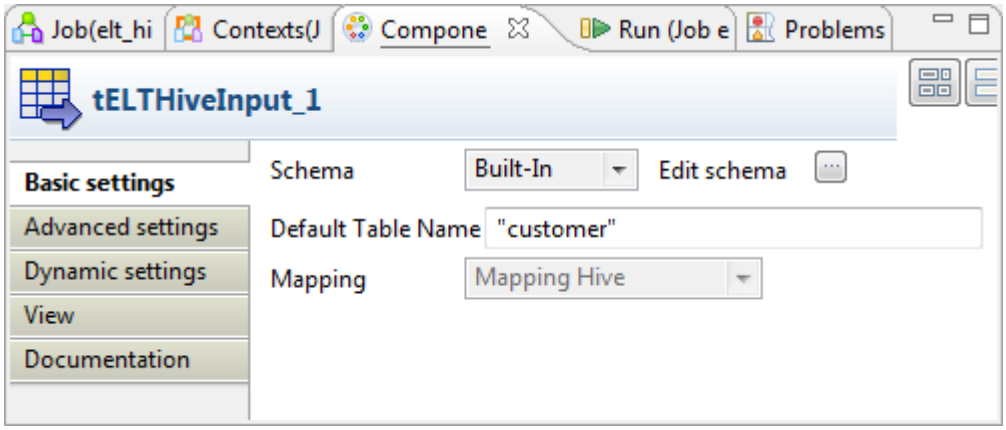
2. Drop two **tELTHiveInput** components and **tELTHiveMap** and **tELTHiveOutput** onto the workspace.

3. Connect them using the **Row** > **Main** link.

   Each time when you connect two components, a wizard pops up to prompt you to name the link you are creating. This name must be the same as that of the Hive table you want the active component to process. In this scenario, the input tables the two **tELTHiveInput** components will handle are *customer* and *state_city* and the output table **tELTHiveOutput** will handle is *agg_result*.

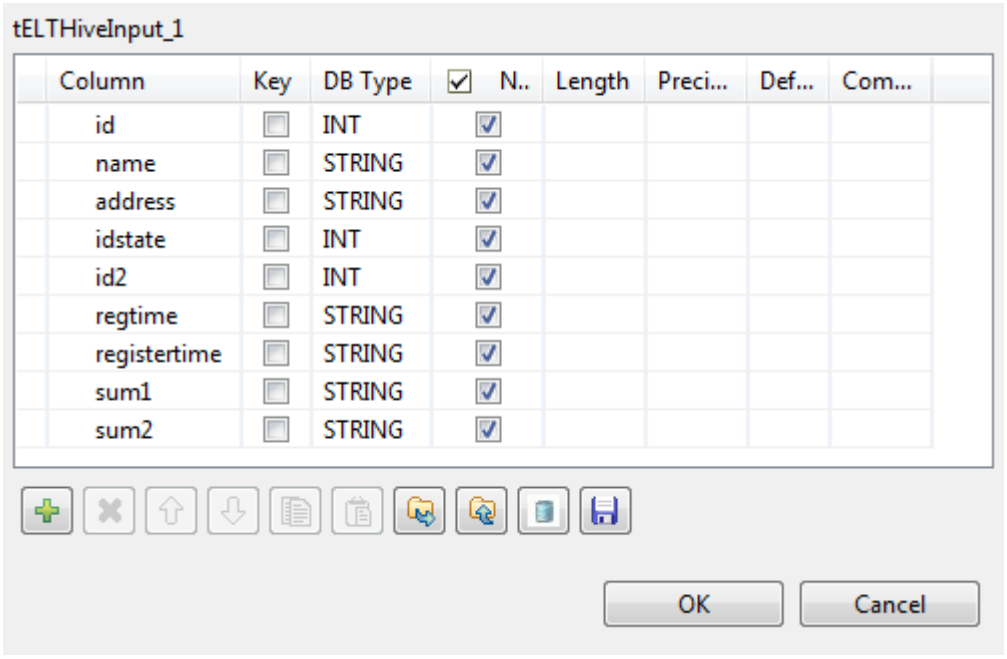
Configuring the input schemas

1. Double-click the **tELTHiveInput** component using the *customer* link to open its **Component** view.



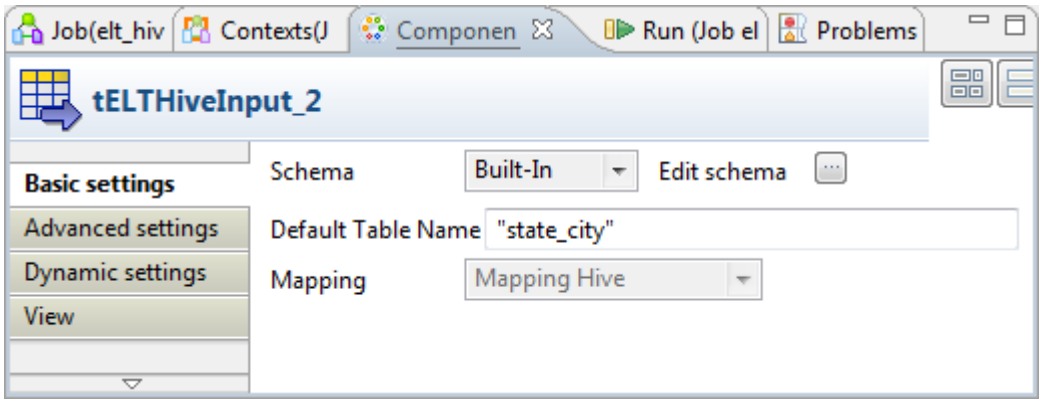
2. Click the **[...]** button next to **Edit schema** to open the schema editor.

3. Click the button as many times as required to add columns and rename them to replicate the schema of the *customer* table we created earlier in Hive.



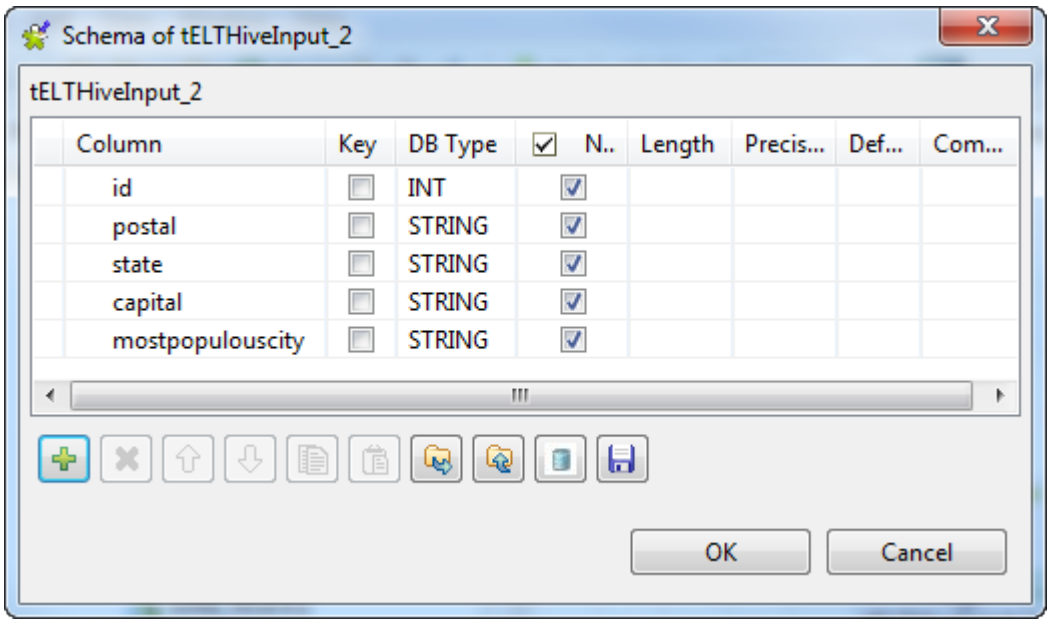
4. In the **Default table name** field, enter the name of the input table, *customer*, to be processed by this component.

5. Double-click the other **tELTHiveInput** component using the *state_city* link to open its **Component** view.



6. Click the [**...**] button next to **Edit schema** to open the schema editor.

7. Click the button as many times as required to add columns and rename them to replicate the schema of the *state_city* table we created earlier in Hive.

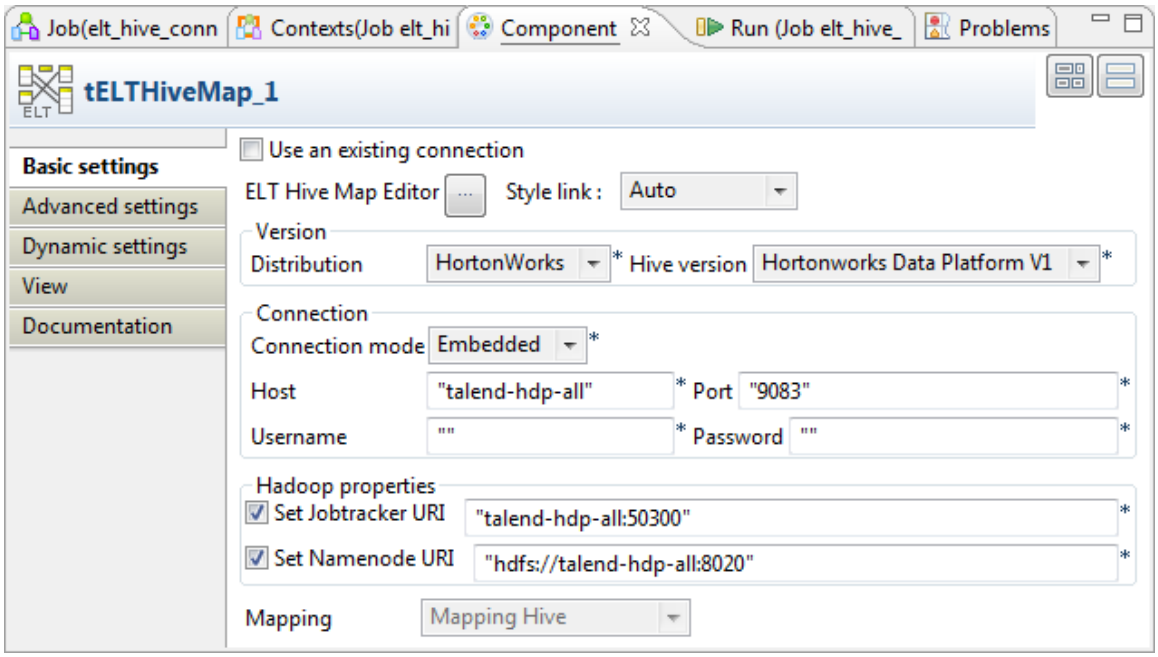


8. In the **Default table name** field, enter the name of the input table, *state_city*, to be processed by this component.

Mapping the input and the output schemas

Configuring the connection to Hive

1. Click **tELTHiveMap**, then, click **Component** to open its **Component** view.



2. In the **Version** area, select the Hadoop distribution you are using and the Hive version.

3. In the **Connection mode** list, select the connection mode you want to use. If your distribution is HortonWorks, this mode is **Embedded** only.

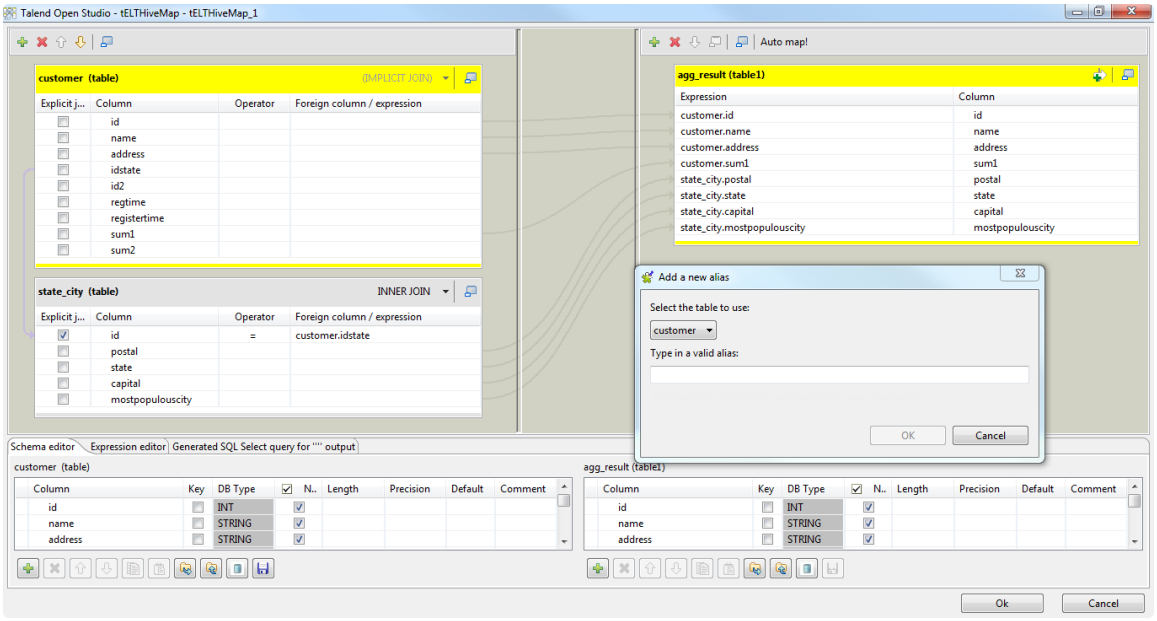
4. In the **Host** field and the **Port** field, enter the authentication information for the component to connect to Hive. For example, the host is *talend-hdp-all* and the port is *9083*.

5. Select the **Set Jobtracker URI** check box and enter the location of the Jobtracker. For example, *talend-hdp-all:50300*.

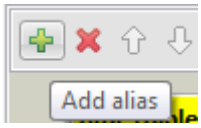
6. Select the **Set NameNode URI** check box and enter the location of the NameNode. For example, *hdfs://talend-hdp-all:8020*.

Mapping the schemas

1. Click **ELT Hive Map Editor** to map the schemas



2. On the input side (left in the figure), click the **Add alias** button to add the table to be used.

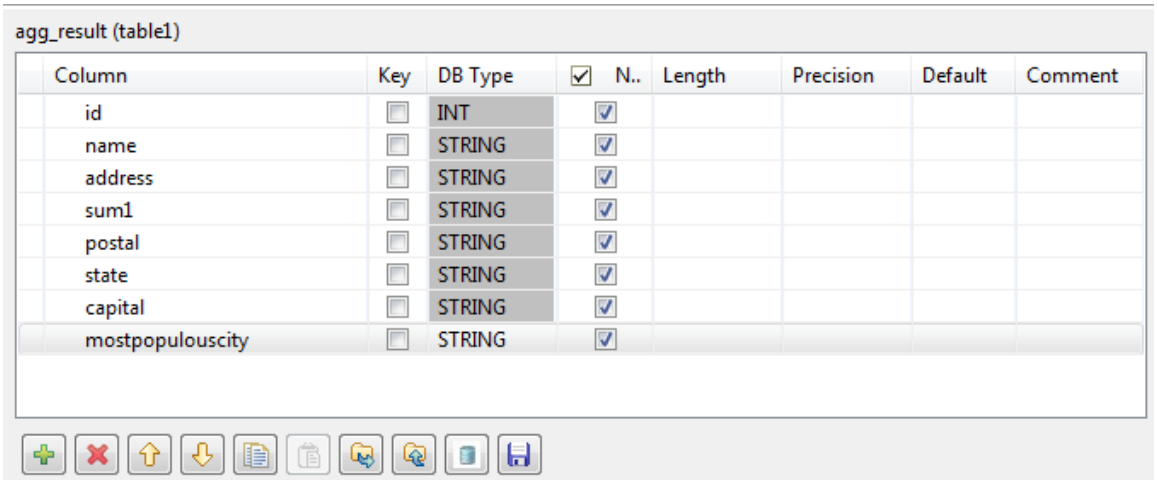


3. In the pop-up window, select the *customer* table, then click **OK**.

4. Repeat the operations to select the *state_city* table.

5. Drag and drop the *idstate* column from the *customer* table onto the *id* column of the *state_city* table. Thus an inner join is created automatically.

6. On the output side (the right side in the figure), the *agg_result* table is empty at first. Click at the bottom of this side to add as many columns as required and rename them to replicate the schema of the *agg_result* table you created earlier in Hive.

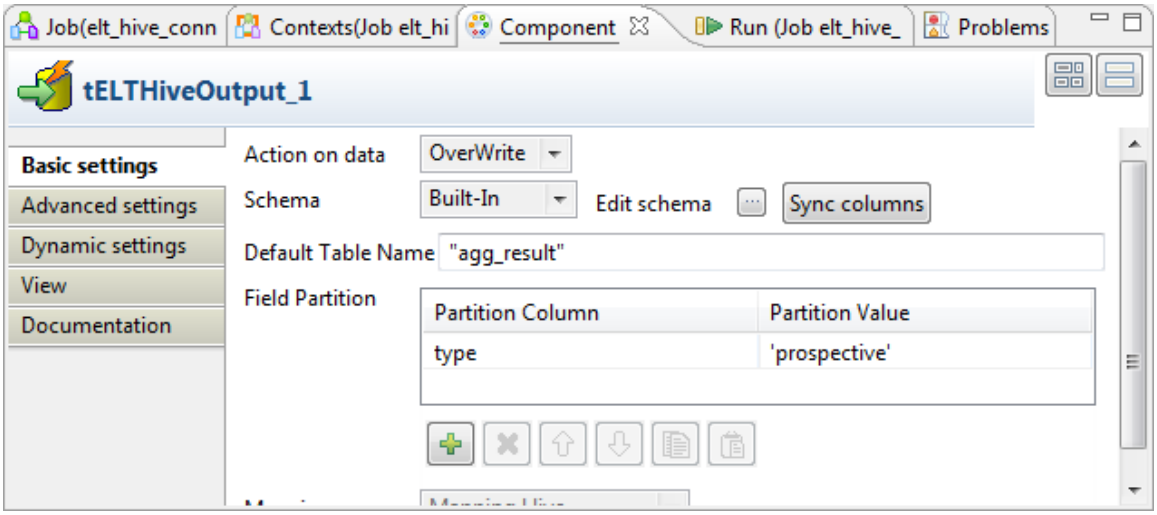


The *type* column is the partition column of the *agg_result* table and should not be replicated in this schema. For further information about the partition column of the Hive table, see the Hive manual.

7. From the *customer* table, drop *id*, *name*, *address*, and *sum1* to the corresponding columns in the *agg_result* table.
8. From the *state_city* table, drop *postal*, *state*, *capital* and *mostpopulouscity* to the corresponding columns in the *agg_result* table.
9. Click **OK** to validate these changes.

Configuring the output schema

1. Double-click **tELTHiveOutput** to open its **Component** view.



2. If this component does not have the same schema of the preceding component, a warning icon appears. In this case, click the **Sync columns** button to retrieve the schema from the preceding one and once done, the warning icon disappears.
3. In the **Default table name** field, enter the output table you want to write data in. In this example, it is *agg_result*.
4. In the **Field partition** table, click to add one row. This allows you to write data in the partition column of the *agg_result* table.

This partition column was defined the moment we created the *agg_result* table using `partitioned by (type string)` in the Create statement presented earlier. This partition column is *type*, which describes the type of a customer.

5. In **Partition column**, enter *type* without any quotation marks and in **Partition value**, enter *prospective* in single quotation marks.

Executing the Job

- Press **F6** to run this Job.

Once done, verify *agg_result* in Hive using, for example,

```
select * from agg_result;
```

```
557     Metro Appliance Service 2045 St. johns Ave.     91052.0 WY      Wyoming Cheyenne        Cheyenne        prospective
2746    North Park Lighting Galleries   1065 Hillcrest Circle   24311.0 WY      Wyoming Cheyenne        Cheyenne        prospective
4037    Resnick DDS     3181 Barkwood Ct.       22011.0 WY      Wyoming Cheyenne        Cheyenne        prospective
374     Palmer Golf Supplies    1126 Hillcrest Ave      72952.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3116    Elm Place Business Supplies     937 Judson Ave. 74250.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3121    Parkway Auto Body       2510 Crain      63350.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3092    Marcy Elaine Bakery     1019 Deerfield Rd.      74340.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3187    Lidhal design Corp.     1233 Arbor Rd.  31534.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3209    Vecot Inc.      4259 W. Lake Ave.       32224.0 WY      Wyoming Cheyenne        Cheyenne        prospective
2274    AA Advertising  313 Grove Ave.  4154.0  WY      Wyoming Cheyenne        Cheyenne        prospective
2171    First National Bank of Ottowa   1287 Glencoe Ave.       32400.0 WY      Wyoming Cheyenne        Cheyenne        prospective
427     Toggle & Myerson Ltd    550 Sheridan    91702.0 WY      Wyoming Cheyenne        Cheyenne        prospective
866     Foy Aviation    2220 Grant Blvd.        8957.0  WY      Wyoming Cheyenne        Cheyenne        prospective
693     Doerner's Awnings and Canopies  1715 Chicago Ave.       29550.0 WY      Wyoming Cheyenne        Cheyenne        prospective
4567    Camp Stuff      367Bluebird Terr.       74248.0 WY      Wyoming Cheyenne        Cheyenne        prospective
2677    Air France      1882 St. Johns  42844.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3310    Resnick DDS     1 Twin Oaks Place       22343.0 WY      Wyoming Cheyenne        Cheyenne        prospective
1482    MedicServe Medical Supply       201-C Frontage Rd.      31968.0 WY      Wyoming Cheyenne        Cheyenne        prospective
1457    Portugal, Liddy, & Son Ltd.     1714 Clifton Rd.        57212.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3318    George Lee Antiques     225 Jeffrys Place       21768.0 WY      Wyoming Cheyenne        Cheyenne        prospective
4822    Parties Galore Golf Course      857 Woodbine Rd 69104.0 WY      Wyoming Cheyenne        Cheyenne        prospective
4815    Foy Aviation    43078 Kennedy Dr.       74466.0 WY      Wyoming Cheyenne        Cheyenne        prospective
1456    Maintence Services of Maine     1065 Hillcrest Circle   34752.0 WY      Wyoming Cheyenne        Cheyenne        prospective
1436    Bako Delicatessen       12 Broadview Lane       53460.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3019    American Apartments     1535 Knollwood Drive    7667.0  WY      Wyoming Cheyenne        Cheyenne        prospective
3357    Air India       22 Clay rd.     32744.0 WY      Wyoming Cheyenne        Cheyenne        prospective
443     North Shore Appliance Service   2220 Grant Blvd.        22289.0 WY      Wyoming Cheyenne        Cheyenne        prospective
3387    Doll House Rock 632 Roger Williams      33128.0 WY      Wyoming Cheyenne        Cheyenne        prospective
2686    Lake Shore Publishing   1795 Lake Cook Rd.      53907.0 WY      Wyoming Cheyenne        Cheyenne        prospective
825     janice Mann Accounting Service  629 Kincaid Ave.        18008.0 WY      Wyoming Cheyenne        Cheyenne        prospective
Time taken: 2.374 seconds
```

This figure present only a part of the table. You can find that the selected input columns are aggregated and written into the *agg_result* table and the partition column is filled with the value *prospective*.

tELTHiveOutput

tELTHiveOutput properties

The three ELT Hive components are closely related, in terms of their operating conditions. These components should be used to handle Hive DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Hive	
Function	This component executes the query built by the preceding **tELTHiveMap** component to write data into the specified Hive table.	
Purpose	This component works alongside **tELTHiveMap** to write data into the Hive table.	
Basic settings	*Action on data*	Select the action to be performed on the data to be written in the Hive table. With the **Insert** option, the data to be written in the Hive table will be appended to the existing data if there is any.
	Schema	**Built-in**: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and Job designs. Related topic: see the user guide of the Studio for integration. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Edit schema	Click **Edit Schema** to make changes to the schema.
	Default table name	Enter the default name of the output table you want to write data in.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field that appears.
	Field Partition	In **Partition Column**, enter the name, without any quotation marks, of the partition column of the Hive table you want to write data in. In **Partition Value**, enter the value you want to use, in single quotation marks, for its corresponding partition column.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	**tELTHiveMap** is used along with a **tELTHiveInput** and **tELTHiveOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. If the Studio used to connect to a Hive database is operated on Windows, you must manually create a file called *tmp* in the root of the disk where this Studio is installed. The ELT components do not handle actual data flow but only schema information.	

Related scenario

For a related scenario, see section *Scenario: Joining table columns and writing them into Hive*

tELTJDBCInput

tELTJDBCInput properties

The three ELT JDBC components are closely related, in terms of their operating conditions. These components should be used to handle JDBC DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/JDBC	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Type in the default table name.
	Default Schema Name	Type in the default schema name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTJDBCInput** is to be used along with the **tELTJDBCMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTJDBCInput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTJDBCMap

tELTJDBCMap properties

The three ELT JDBC components are closely related, in terms of their operating conditions. These components should be used to handle JDBC DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/JDBC	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT JDBC Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either Built-in or Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTJDBCMap** is used along with **tELTJDBCInput** and **tELTJDBCOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenario:

For related scenarios, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTJDBCOutput

tELTJDBCOutput properties

The three ELT JDBC components are closely related, in terms of their operating conditions. These components should be used to handle JDBC DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/JDBC	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the JDBC database	
Basic settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Adds new entries to the table. If duplicates are found, Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name, between double quotation marks.
	Default Schema Name	Enter the default schema name,between double quotation marks.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTJDBCOutput** is to be used along with the **tELTJDBCMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTJDBCOutput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTMSSqlInput

tELTMSSqlInput properties

The three ELT MSSql components are closely related, in terms of their operating conditions. These components should be used to handle MSSql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/MSSql	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Type in the default table name.
	Default Schema Name	Type in the default schema name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTMySSqlInput** is to be used along with the **tELTMSSsqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTMSSqlInput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTMSSqlMap

tELTMSSqlMap properties

The three ELT MSSql components are closely related, in terms of their operating conditions. These components should be used to handle MSSql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/MSSql	
Function	Helps you to build the SQL statement graphically, using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT MSSql Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either Built-in or Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTMSSqlMap** is used along with a **tELTMSSqlInput** and **tELTMSSqlOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenario:

For related scenarios, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTMSSqlOutput

tELTMSSqlOutput properties

The three ELT MSSql components are closely related, in terms of their operating conditions. These components should be used to handle MSSql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/MSSql	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the MSSql database	
Basic settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Adds new entries to the table. If duplicates are found, Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to modify the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name, between double quotation marks.
	Default Schema Name	Enter the default schema name,between double quotation marks.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTMSSqlOutput** is to be used along with the **tELTMSSqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tELTMSSqlOutput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTMysqlInput

tELTMysqlInput properties

The three ELT Mysql components are closely related, in terms of their operating conditions. These components should be used to handle Mysql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Mysql	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. Click **Edit Schema** to modify the schema. Note that if you make the modification, the schema switches automatically to the **Built-in** mode. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Enter the default table name, between double quotation marks.
Usage	**tELTMysqlInput** is to be used along with the **tELTMysqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTMysqlInput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTMysqlMap

tELTMysqlMap properties

The three ELT Mysql components are closely related, in terms of their operating conditions. These components should be used to handle Mysql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Mysql	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Mysql Map editor	The ELT Map editor allows you to define the output schema as well as build graphically the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
Usage	**tELTMysqlMap** is used along with a **tELTMysqlInput** and **tELTMysqlOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. The ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

The ELT components do not handle any data as such but table schema information that will be used to build the SQL query to execute.

Therefore the only connection required to connect these components together is a simple link.

The output name you give to this link when creating it should always be the exact name of the table to be accessed as this parameter will be used in the SQL statement generated.

Related topic: see *Talend Open Studio for Big Data User Guide*.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

- As you would do it in the regular Map editor, simply drag & drop the content from the input schema towards the output table defined.
- Use the **Ctrl** and **Shift** keys for multiple selection of contiguous or non contiguous table columns.

You can implement explicit joins to retrieve various data from different tables.

- Select the **Explicit join** check box for the relevant column, and selct a type of join from the **Join** list.
- Possible joins include: **Inner Join**, **Left Outer Join**, **Right Outer Join** or **Full Outer Join** and **Cross Join**.
- By default the **Inner Join** is selected.

You can also create **Alias** tables to retrieve various data from the same table.

- In the Input area, click on the plus **[+]** button to create an Alias.
- Define the table to base the alias on.
- Type in a new name for the alias table, preferably not the same as the main table.

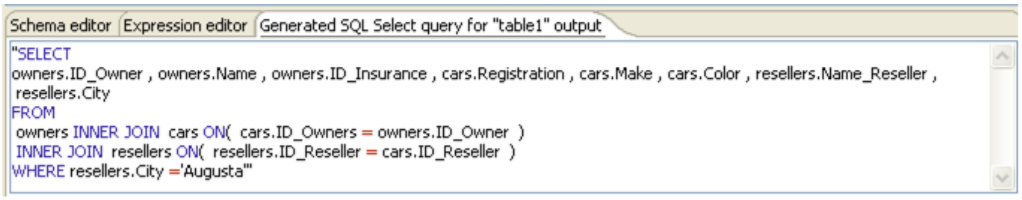
Adding where clauses

You can also restrict the Select statement based on a Where clause. Click the **Add filter row** button at the top of the output table and type in the relevant restriction to be applied.

Make sure that all input components are linked correctly to the ELT Map component to be able to implement all inclusions, joins and clauses.

Generating the SQL statement

The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.

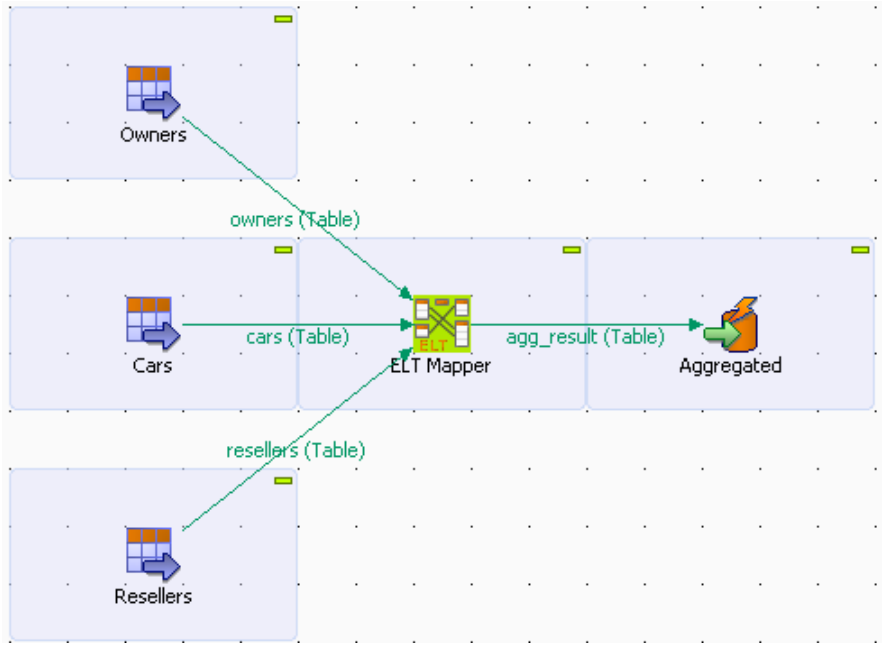


The clause are also included automatically.

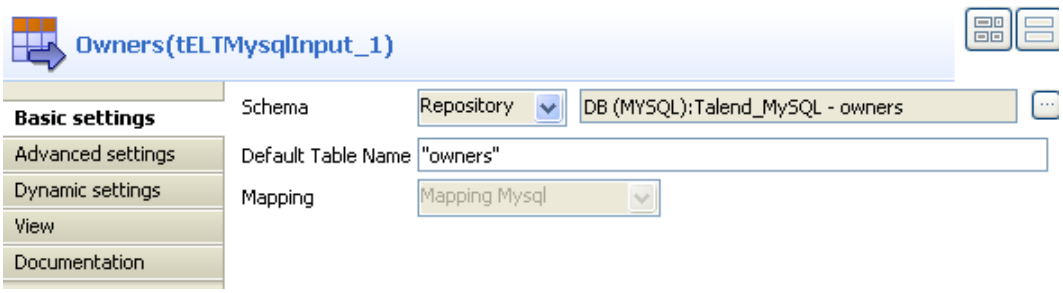
Scenario 1: Aggregating table columns and filtering

This scenario is for reference only. If you are using Talend Open Studio for Big Data, the property type and schema of a component are always built-in, which means you have to configure the database connection details and schemas manually.

This scenario describes a Job that gathers together several input DB table schemas and implementing a clause to filter the output using an SQL statement.



- Drop the following components from the **Palette** onto the design workspace: three **tELTMysqlInput** components, a **tELTMysqlMap**, and a **tELTMysqlOutput**. Label these components to best describe their functionality.
- Double-click the first **tELTMysqlInput** component to display its **Basic settings** view.



- Select **Repository** from the **Schema** list, click the three dot button preceding **Edit schema**, and select your DB connection and the desired schema from the **[Repository Content]** dialog box.

  The selected schema name appears in the **Default Table Name** field automatically.

  In this use case, the DB connection is *Talend_MySQL* and the schema for the first input component is *owners*.

- Set the second and third **tELTMysqlInput** components in the same way but select *cars* and *resellers* respectively as their schema names.

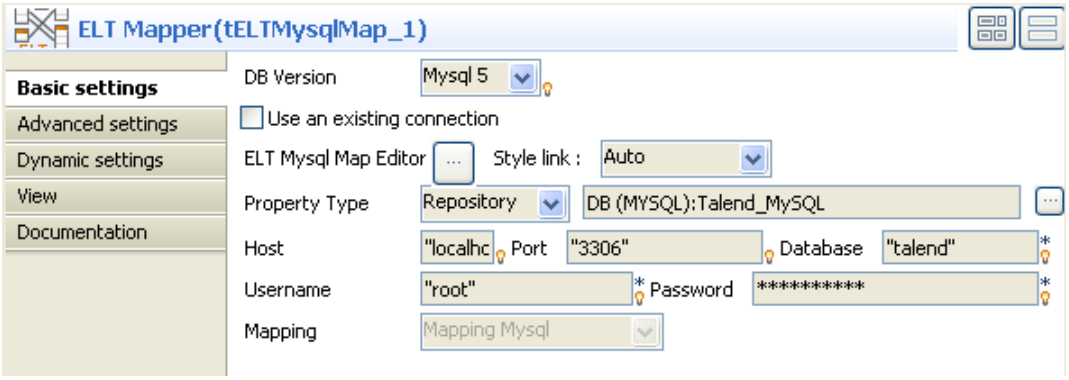
In this use case, all the involved schemas are stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information concerning metadata, see *Talend Open Studio for Big Data* **User Guide**.

You can also select the three input components by dropping the relevant schemas from the **Metadata** area onto the design workspace and double-clicking **tELTMysqlInput** from the **[Components]** dialog box. Doing so allows you to skip the steps of labeling the input components and defining their schemas manually.

- Connect the three **tELTMysqlInput** components to the **tELTMysqlMap** component using links named following strictly the actual DB table names: *owners*, *cars* and *resellers*.

- Connect the **tELTMysqlMap** component to the **tELTMysqlOutput** component and name the link *agg_result*, which is the name of the database table you will save the aggregation result to.

- Click the **tELTMysqlMap** component to display its **Basic settings** view.

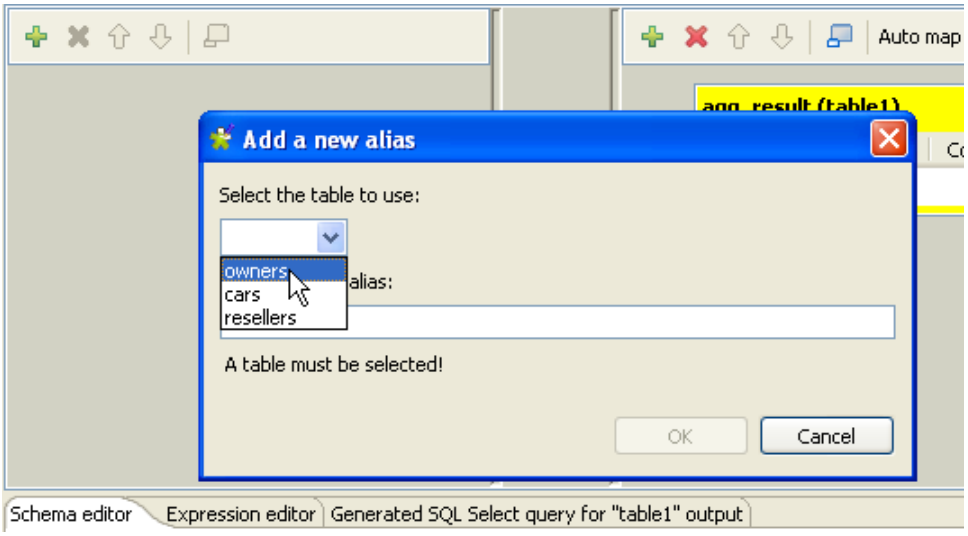


- Select **Repository** from the **Property Type** list, and select the same DB connection that you use for the input components.

  All the database details are automatically retrieved.

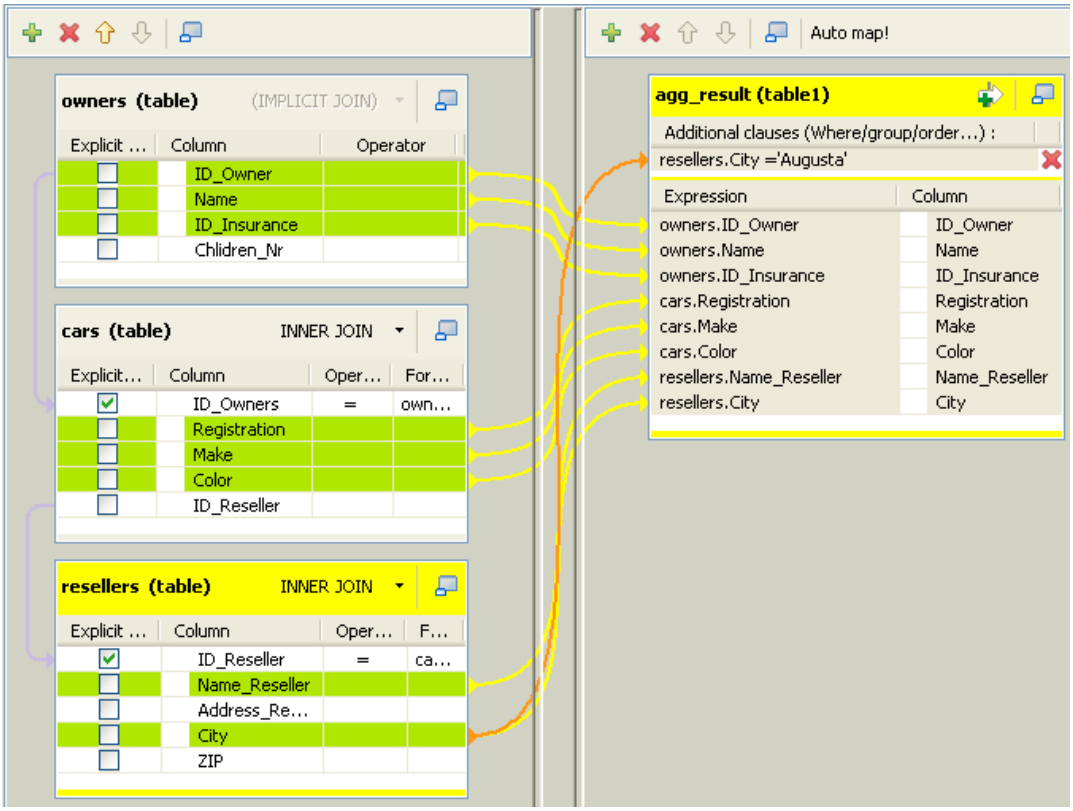
- Leave all the other settings as they are.

- Double-click the **tELTMysqlMap** component to launch the ELT Map editor to set up joins between the input tables and define the output flow.

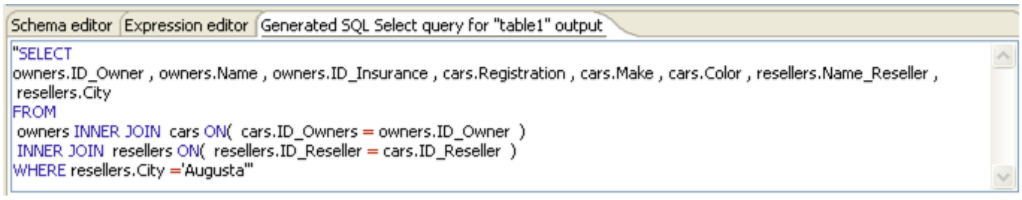


- Add the input tables by clicking the green plus button at the upper left corner of the ELT Map editor and selecting the relevant table names in the **[Add a new alias]** dialog box.
- Drop the *ID_Owner* column from the *owners* table to the corresponding column of the *cars* table.
- In the *cars* table, select the **Explicit join** check box in front of the *ID_Owner* column.

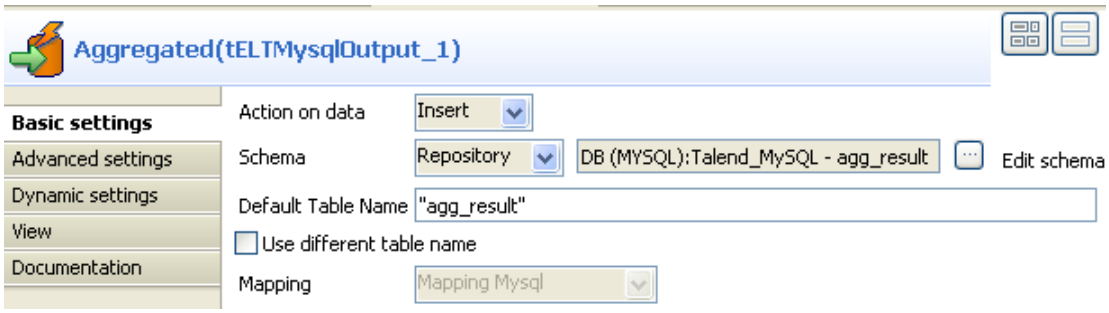
  As the default join type, **INNER JOIN** is displayed on the **Join** list.
- Drop the *ID_Reseller* column from the *cars* table to the corresponding column of the *resellers* table to set up the second join, and define the join as an inner join in the same way.
- Select the columns to be aggregated into the output table, *agg_result*.
- Drop the *ID_Owner*, *Name*, and *ID_Insurance* columns from the *owners* table to the output table.
- Drop the *Registration*, *Make*, and *Color* columns from the *cars* table to the output table.
- Drop the *Name_Reseller* and *City* columns from the *resellers* table to the output table.
- With the relevant columns selected, the mappings are displayed in yellow and the joins are displayed in dark violet.
- Set up a filter in the output table. Click the **Add filter row** button on top of the output table to display the **Additional clauses** expression field, drop the *City* column from the *resellers* table to the expression field, and complete a WHERE clause that reads `resellers.City ='Augusta'`.



- Click the **Generated SQL Select query** tab to display the corresponding SQL statement.



- Click **OK** to save the ELT Map settings.
- Double-click the **tELTMysqlOutput** component to display its **Basic settings** view.



- Select an action from the **Action on data** list as needed.
- Select **Repository** as the schema type, and define the output schema in the same way as you defined the input schemas. In this use case, select *agg_result* as the output schema, which is the name of the database table used to store the mapping result.

You can also use a built-in output schema and retrieve the schema structure from the preceding component; however, make sure that you specify an existing target table having the same data structure in your database.

- Leave all the other settings as they are.

- Save your Job and press **F6** to launch it.

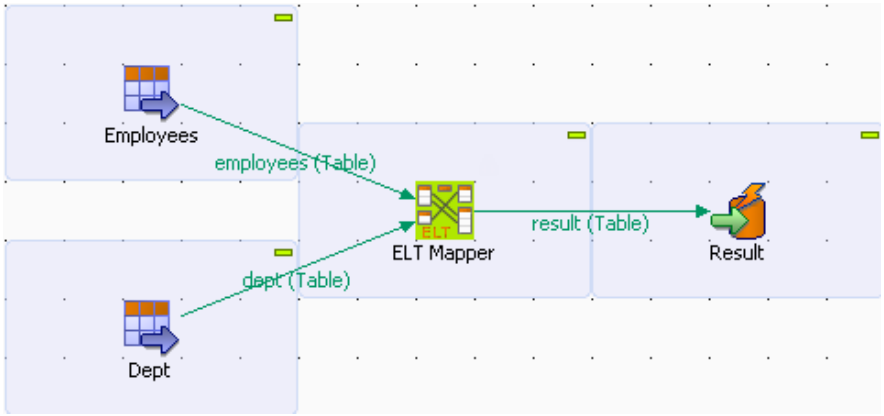
  All selected data is inserted in the *agg_result* table as specified in the SQL statement.

	ID_Owner	Name	ID_Insurance	Registration	Make	Color	Name_Reseller	City
☐	69	John ROOSEVELT	84	UDR 821	Peugeot	green	Bill GARFIELD	AUGUSTA
☐	11	Andrew COOLIDGE	104	ZYX 387	BMW	yellow	Ulysses POLK	AUGUSTA
☐	10	Martin ADAMS	167	ZFF 348	Audi	blue	Rutherford HARRISON	AUGUSTA
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

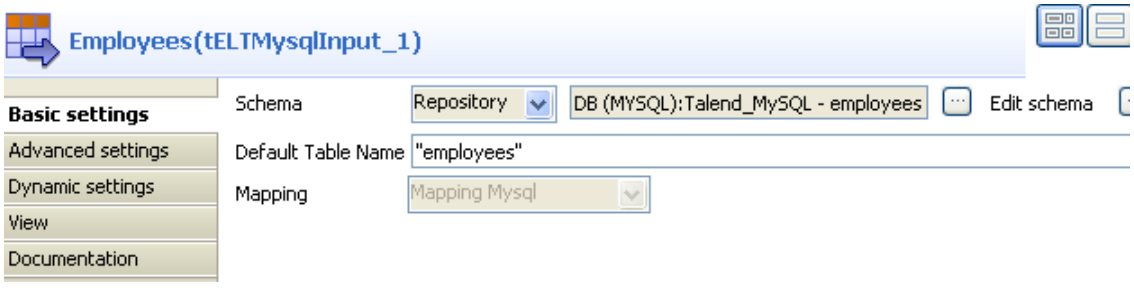
Scenario 2: ELT using an Alias table

This scenario is for reference only. If you are using Talend Open Studio for Big Data, the property type and schema of a component are always built-in, which means you have to configure the database connection details and schemas manually.

This scenario describes a Job that maps information from two input tables and an alias table, serving as a virtual input table, to an output table. The *employees* table contains employees' IDs, their department numbers, their names, and the IDs of their respective managers. The managers are also considered as employees and hence included in the *employees* table. The *dept* table contains the department information. The alias table retrieves the names of the managers from the *employees* table.



- Drop two **tELTMysqlInput** components, a **tELTMysqlMap** component, and a **tELTMysqlOutput** component to the design workspace, and label them to best describe their functionality.
- Double-click the first **tELTMysqlInput** component to display its **Basic settings** view.



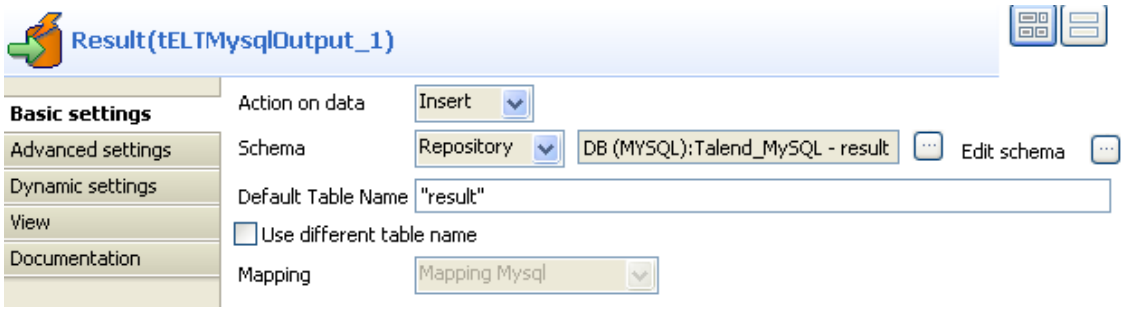
- Select **Repository** from the **Schema** list, and define the DB connection and schema by clicking the three dot button preceding **Edit schema**.

  The DB connection is *Talend_MySQL* and the schema for the first input component is *employees*.

In this use case, all the involved schemas are stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information concerning metadata, see *Talend Open Studio for Big Data* **User Guide**.

- Set the second **tELTMysqlInput** component in the same way but select *dept* as its schema.

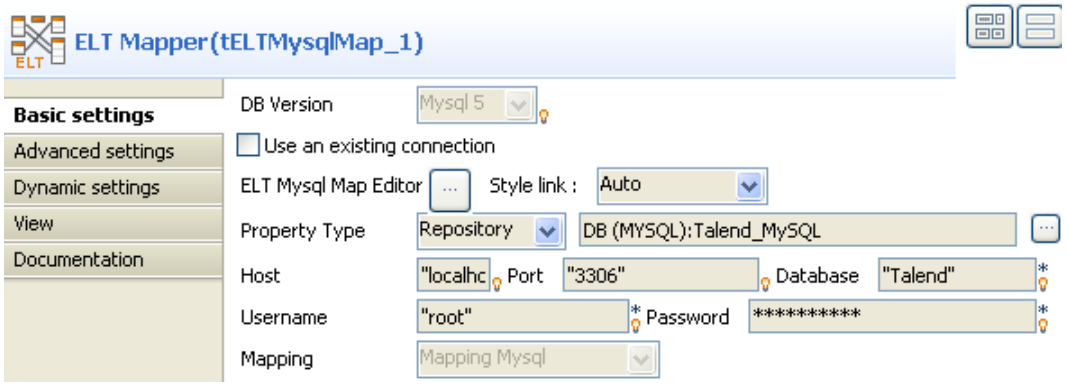
- Double-click the **tELTMysqlOutput** component to display its **Basic settings** view.



- Select an action from the **Action on data** list as needed, **Insert** in this use case.
- Select **Repository** as the schema type, and define the output schema in the same way as you defined the input schemas. In this use case, select *result* as the output schema, which is the name of the database table used to store the mapping result.

  The output schema contains all the columns of the input schemas plus a *ManagerName* column.

- Leave all the other parameters as they are.
- Connect the two **tELTMysqlInput** components to the **tELTMysqlMap** component using **Link** connections named strictly after the actual input table names, *employees* and *dept* in this use case.
- Connect the **tELTMysqlMap** component to the **tELTMysqlOutput** component using a **Link** connection. When prompted, click **Yes** to allow the ELT Mapper to retrieve the output table structure from the output schema.
- Click the **tELTMysqlMap** component and select the **Component** tab to display its **Basic settings** view.



- Select **Repository** from the **Property Type** list, and select the same DB connection that you use for the input components.

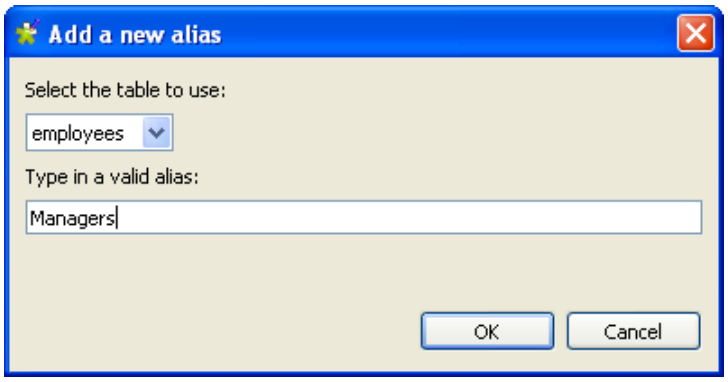
  All the DB connection details are automatically retrieved.

- Leave all the other parameters as they are.
- Click the three-dot button next to **ELT Mysql Map Editor** or double-click the **tELTMysqlMap** component on the design workspace to launch the ELT Map editor.

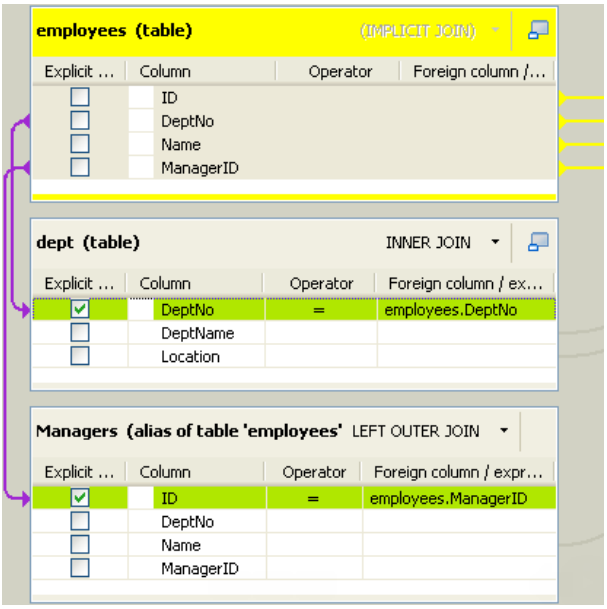
  With the **tELTMysqlMap** component connected to the output component, the output table is displayed in the output area.

- Add the input tables, *employees* and *dept*, in the input area by clicking the green plus button and selecting the relevant table names in the **[Add a new alias]** dialog box.

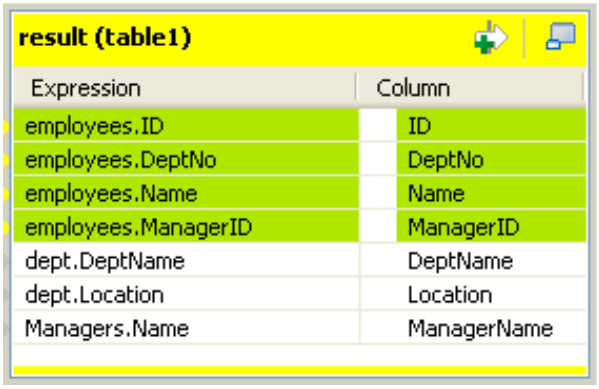
- Create an alias table based on the *employees* table by selecting *employees* from the **Select the table to use** list and typing in *Managers* in the **Type in a valid alias** field in the the **[Add a new alias]** dialog box.



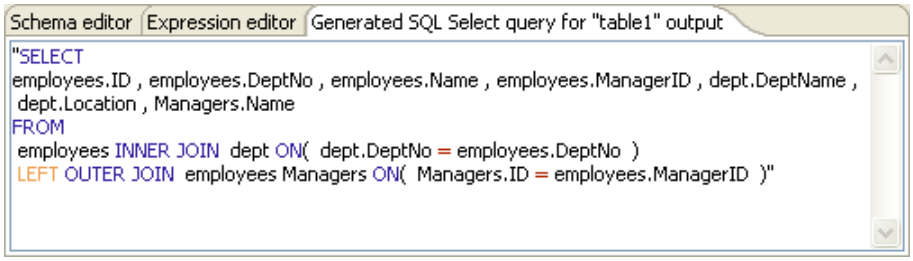
- Drop the *DeptNo* column from the *employees* table to the *dept* table.
- Select the **Explicit join** check box in front of the *DeptNo* column of the *dept* table to set up an inner join.
- Drop the *ManagerID* column from the *employees* table to the *ID* column of the *Managers* table.
- Select the **Explicit join** check box in front of the *ID* column of the *Managers* table and select **LEFT OUTER JOIN** from the **Join** list to allow the output rows to contain Null values.



- Drop all the columns from the *employees* table to the corresponding columns of the output table.
- Drop the *DeptName* and *Location* columns from the *dept* table to the corresponding columns of the output table.
- Drop the *Name* column from the *Managers* table to the *ManagerName* column of the output table.



- Click on the **Generated SQL Select query** tab to display the SQL query statement to be executed.



- Save your Job and press **F6** to run it.

  The output database table *result* contains all the information about the employees, including the names of their respective managers.

	ID	DeptNo	Name	ManagerID	DeptName	Location	ManagerName
☐	1	10	Alex	(NULL)	R&D	New York	(NULL)
☐	2	20	Peter	(NULL)	Accounting	Dallas	(NULL)
☐	3	10	Mark	1	R&D	New York	Alex
☐	4	10	Michael	1	R&D	New York	Alex
☐	5	20	Christophe	2	Accounting	Dallas	Peter
☐	6	10	Stephane	3	R&D	New York	Mark
☐	7	10	Cedric	3	R&D	New York	Mark
☐	8	10	Bill	4	R&D	New York	Michael
☐	9	20	Jack	2	Accounting	Dallas	Peter
☐	10	10	Andrews	4	R&D	New York	Michael
*	NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

tELTMysqlOutput

tELTMysqlOutput properties

The three ELT Mysql components are closely related, in terms of their operating conditions. These components should be used to handle Mysql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Mysql	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Mysql database	
Basic settings *Use tCreateTable as substitute for this function.*	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to modify the schema. Note that if you make the modification, the schema switches automatically to the **Built-in** mode. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name, between inverted commas.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field which appears.
Usage	**tELTMysqlOutput** is to be used along with the **tELTMysqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTMysqlOutput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*

- section *Scenario 2: ELT using an Alias table*

tELTNetezzaInput

tELTNetezzaInput properties

The three ELT Netezza components are closely related, in terms of their operating conditions. These components should be used to handle Netezza DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Netezza	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. For *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Type in the default table name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTNetezzaInput** is to be used along with the **tELTNetezzaMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*
- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTNetezzaMap

tELTNetezzaMap properties

The three ELT Netezza components are closely related, in terms of their operating conditions. These components should be used to handle Netezza DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Netezza	
Function	Helps you to build the SQL statement graphically, using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Netezza Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line (fastest)**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	For *Talend Open Studio for Big Data*, the property type is always **Built-in**.
		Built-in: No property data stored centrally.

	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTNetezzaMap** is used along with **tELTNetezzaInput** and **tELTNetezzaOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*.
- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTNetezzaOutput

tELTNetezzaOutput properties

The three ELT Netezza components are closely related, in terms of their operating conditions. These components should be used to handle Netezza DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Netezza	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Netezza database	
Basic settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Adds new entries to the table. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. For *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name, between double quotation marks.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field that appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTNetezzaOutput** is to be used along with the **tELTNetezzaMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For related scenarios, see:

- section *Scenario: Mapping data using a simple implicit join*

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTOracleInput

tELTOracleInput properties

The three ELT Oracle components are closely related, in terms of their operating conditions. These components should be used to handle Oracle DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Oracle	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. Click **Edit Schema** to modify the schema. Note that if you make the modification, the schema switches automatically to the **Built-in** mode. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Enter the default table name, between double quotation marks.
	Default Schema Name	Enter the default schema name,between double quotation marks.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTOracleInput** is to be used along with the **tELTOracleMap**. Note that the Output link to be used with these components must must correspond strictly to the syntax of the table name The ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTOracleInput**, see section *Scenario: Updating Oracle DB entries*.

tELTOracleMap

tELTOracleMap properties

The three ELT Oracle components are closely related, in terms of their operating conditions. These components should be used to handle Oracle DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Oracle	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate **tOracleConnection** component from the **Component list** if you want to re-use connection parameters that you have already defined When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Oracle Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	**Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either Built-in or Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Connection type	Drop-down list of the available drivers.
	DB Version	Select the Oracle version you are using.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Mapping	Automatically set mapping parameter.
Advanced settings	*Additional JDBC Parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	Use Hint Options	Select this check box to activate the hint configuration area to help you optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */`. - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTOracleMap** is used along with a **tELTOracleInput** and **tELTOracleOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

For detailed information regarding ELT component connections, see section *Connecting ELT components*.

Related topic: see *Talend Open Studio for Big Data* **User Guide**.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

For detailed information regarding the table schema mapping and joining, see section *Mapping and joining tables*.

When you need to join a lot of tables or need to join tables by multiple join conditions with outer joins, it is recommended to use the **LEFT OUTER JOIN (+)** and the **RIGHT OUTER JOIN (+)** options that allow you to use the Oracle private keywords. For further information about these two private keywords, see the site: *http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/queries006.htm*

Adding where clauses

For details regarding the clause handling, see section *Adding where clauses*.

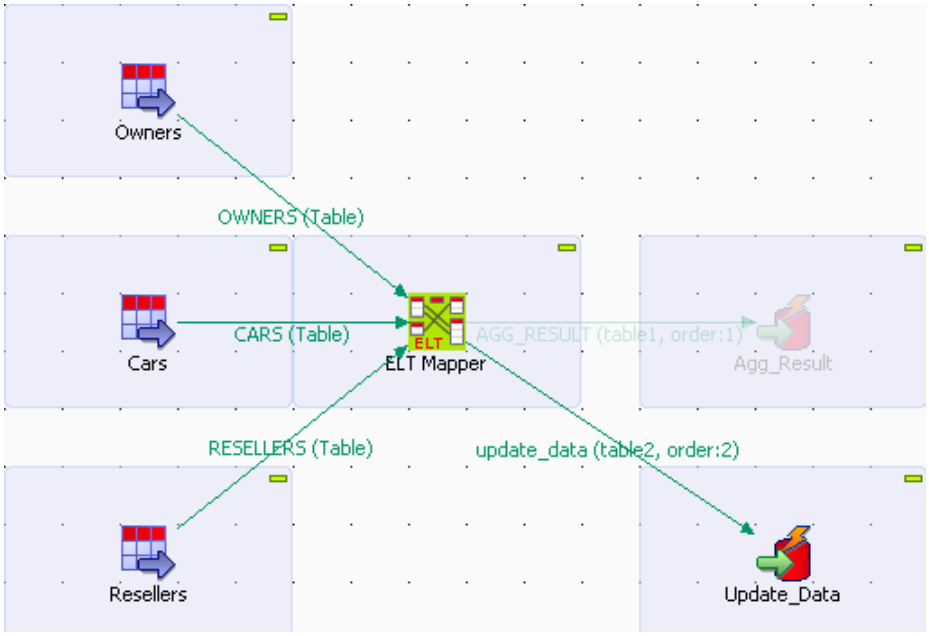
Generating the SQL statement

The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.

The clause defined internally in the ELT Mapper are also included automatically.

Scenario: Updating Oracle DB entries

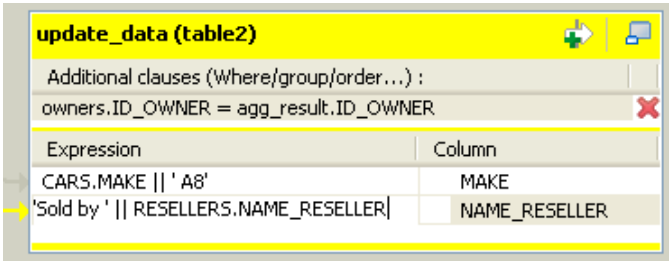
This scenario is based on the data aggregation scenario, section *Scenario 1: Aggregating table columns and filtering*. As the data update action is available in Oracle DB, this scenario describes a Job that updates particular data in the *agg_result* table.



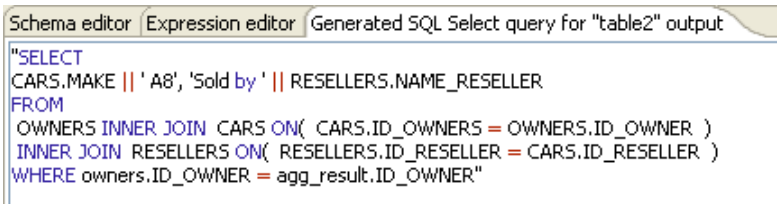
- As described in section *Scenario 1: Aggregating table columns and filtering*, set up a Job for data aggregation using the corresponding ELT components for Oracle DB, **tELTOracleInput**, **tELTOracleMap**, and **tELTOracleOutput**, and execute the Job to save the aggregation result in a database table named *agg_result*.

When defining filters in the ELT Map editor, note that strings are case sensitive in Oracle DB.

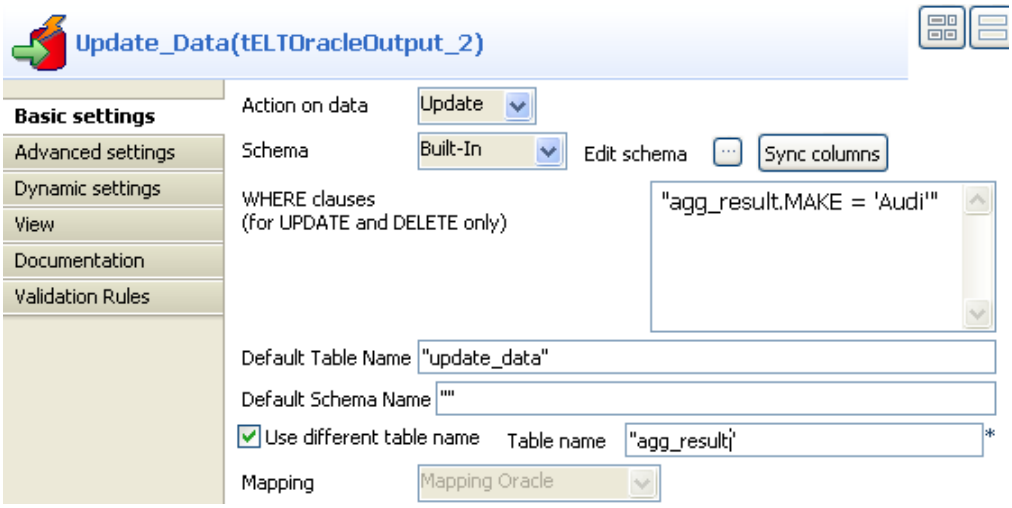
- Launch the ELT Map editor and add a new output table named *update_data*.
- Add a filter row to the *update_data* table to set up a relationship between input and output tables: `owners.ID_OWNER = agg_result.ID_OWNER`.
- Drop the *MAKE* column from the *cars* table to the *update_data* table.
- Drop the *NAME_RESELLER* column from the *resellers* table to the *update_data* table.
- Add a model enclosed in single quotation marks, *A8* in this use case, to the *MAKE* column from the cars table, preceded by a double pipe.
- Add *Sold by* enclosed in single quotation marks in front of the *NAME_RESELLER* column from the *resellers* table, with a double pipe in between.



- Check the **Generated SQL select query** tab to be executed.



- Click **OK** to validate the changes in the ELT Mapper.
- Deactivate the **tELTOracleOutput** component labeled *Agg_Result* by right-clicking it and selecting **Deactivate Agg_Result** from the contextual menu.
- Drop a new **tELTOracleOutput** component from the **Palette** to the design workspace, and label it *Update_Data* to better identify its functionality.
- Connect the **tELTOracleMap** component to the new **tELTOracleOutput** component using the link corresponding to the new output table defined in the ELT Mapper, *update_data* in this use case.
- Double-click the new **tELTOracleOutput** component to display its **Basic settings** view.



- From the **Action on data** list, select **Update**.
- Check the schema, and click **Sync columns** to retrieve the schema structure from the preceding component if necessary.
- In the **WHERE clauses** area, add a clause that reads `agg_result.MAKE = 'Audi'` to update data relating to the make of *Audi* in the database table *agg_result*.
- Fill the **Default Table Name** field with the name of the output link, *update_data* in this use case.
- Select the **Use different table name** check box, and fill the **Table name** field with the name of the database table to be updated, *agg_result* in this use case.

- Leave the other parameters as they are.
- Save your Job and press **F6** to run it.

  The relevant data in the database table is updated as defined.

ID_OW...	NAME	ID_INS...	REGISTR...	MAKE	COLOR	NAME_RESELLER	CITY
10	Martin ADAMS	167	ZFF 348	Audi A8	grey	Sold by Rutherford HARRISON	AUGUSTA
11	Andrew COOLIDGE	104	ZYX 387	BMW	yellow	Ulysses POLK	AUGUSTA
69	John ROOSEVELT	84	UDR 821	Peugeot	green	Bill GARFIELD	AUGUSTA

tELTOracleOutput

tELTOracleOutput properties

The three ELT Oracle components are closely related, in terms of their operating conditions. These components should be used to handle Oracle DB schemas to generate Insert, Update or Delete statements, including clauses, which are to be executed in the DB output table defined.

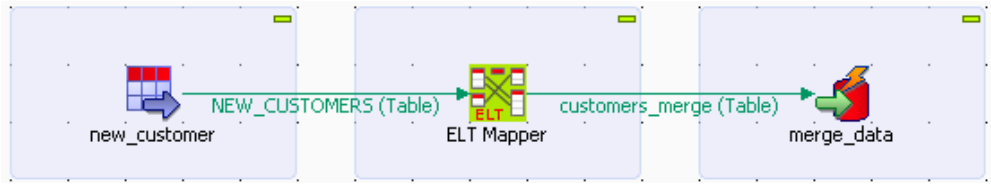
Component family	ELT/Map/Oracle	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Mysql database.	
Basic Settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, the Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow. **MERGE**: Updates or adds data to the table. The options available for the MERGE operation are different to those available for the Insert, Update or Delete operations.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to modify the schema. Note that if you make the modification, the schema switches automatically to the **Built-in** mode. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Use Merge Update (for **MERGE**)	Select this check box to update the data in the output table. **Column** : Lists the columns in the entry flow. **Update** : Select the check box which corresponds to the name of the column you want to update. **Use Merge Update Where Clause** : Select this check box and enter the WHERE clause required to filter the data to be updated, if necessary. **Use Merge Update Delete Clause**: Select this check box and enter the WHERE clause required to filter the data to be deleted and updated, if necessary.

	Use Merge Insert (for **MERGE**)	Select this check box to insert the data in the table. **Column**: Lists the entry flow columns. **Check All**: Select the check box corresponding to the name of the column you want to insert. **Use Merge Update Where Clause**: Select this check box and enter the WHERE clause required to filter the data to be inserted.
	Default Table Name	Enter a default name for the table, between double quotation marks.
	Default Schema Name	Enter a name for the default Oracle schema, between double quotation marks.
	Use different table name	Select this check box to define a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*Use Hint Options*	Select this check box to activate the hint configuration area when you want to use a hint to optimize a query's execution. In this area, parameters are: - **HINT**: specify the hint you need, using the syntax `/*+ */`. - **POSITION**: specify where you put the hint in a SQL statement. - **SQL STMT**: select the SQL statement you need to use.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTOracleOutput** is to be used along with the **tELTOracleInput** and **tELTOracleMap** components. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

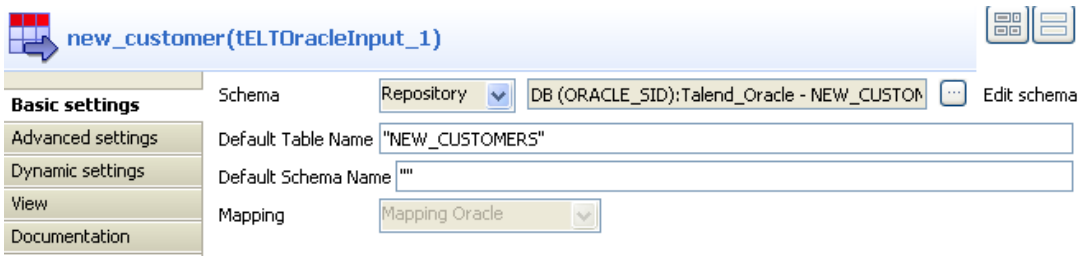
Scenario: Using the Oracle MERGE function to update and add data simultaneously

This scenario is for reference only. If you are using Talend Open Studio for Big Data, the property type and schema of a component are always built-in, which means you have to configure the database connection details and schemas manually.

This scenario describes a Job that allows you to add new customer information and update existing customer information in a database table using the Oracle MERGE command.



- Drop the following components from the **Palette** to the design workspace: **tELTOracleInput**, **tELTOracleMap**, and **tELTOracleOutput**, and label them to identify their functionality.
- Double-click the **tELTOracleInput** component to display its **Basic settings** view.



- Select **Repository** from the **Schema** list, click the three dot button preceding **Edit schema**, and select your DB connection and the desired schema from the **[Repository Content]** dialog box.

  The selected schema name appears in the **Default Table Name** field automatically.

  In this use case, the DB connection is *Talend_Oracle* and the schema is *new_customers*.

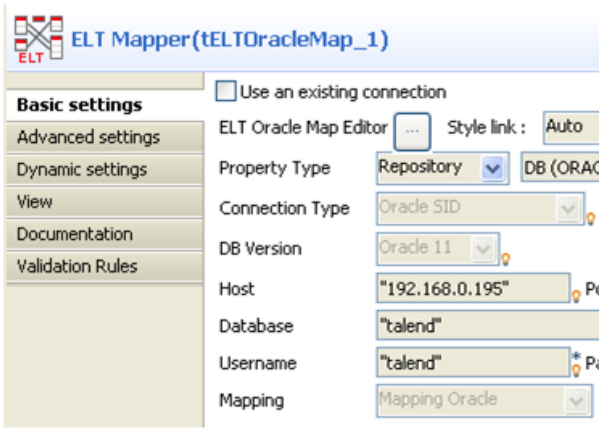
  In this use case, the input schema is stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information concerning metadata, see *Talend Open Studio for Big Data* **User Guide**.

  You can also select the input component by dropping the relevant schema from the **Metadata** area onto the design workspace and double-clicking **tELTOracleInput** from the **[Components]** dialog box. Doing so allows you to skip the steps of labeling the input component and defining its schema manually.

- Connect the **tELTOracleInput** component to the **tELTOraclelMap** component using the link named strictly after the actual DB table name, *new_customers* in this use case.

- Connect the **tELTOraclelMap** component to the **tELTOracleOutput** component and name the link *customers_merge*, which is the name of the database table you will save the merge result to.

- Click the **tELTOracleMap** component to display its **Basic settings** view.



- Select **Repository** from the **Property Type** list, and select the same DB connection that you use for the input components.

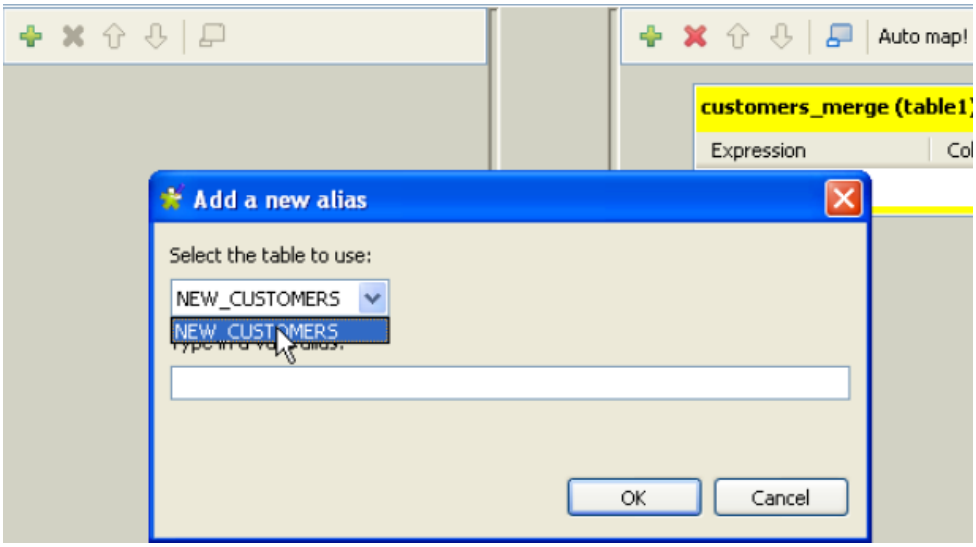
  All the database details are automatically retrieved.

- Leave the other settings as they are.

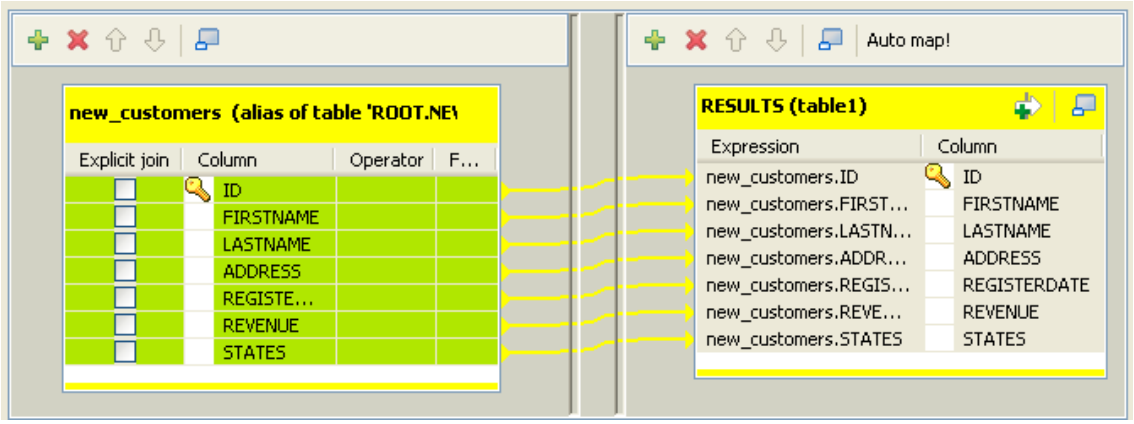
- Double-click the **tELTOracleMap** component to launch the ELT Map editor to set up the data transformation flow.

- Display the input table by clicking the green plus button at the upper left corner of the ELT Map editor and selecting the relevant table name in the **[Add a new alias]** dialog box.

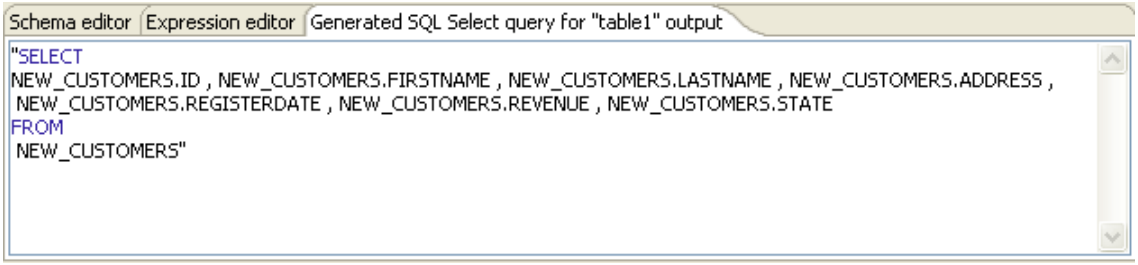
  In this use case, the only input table is *new_customers*.



- Select all the columns in the input table and drop them to the output table.

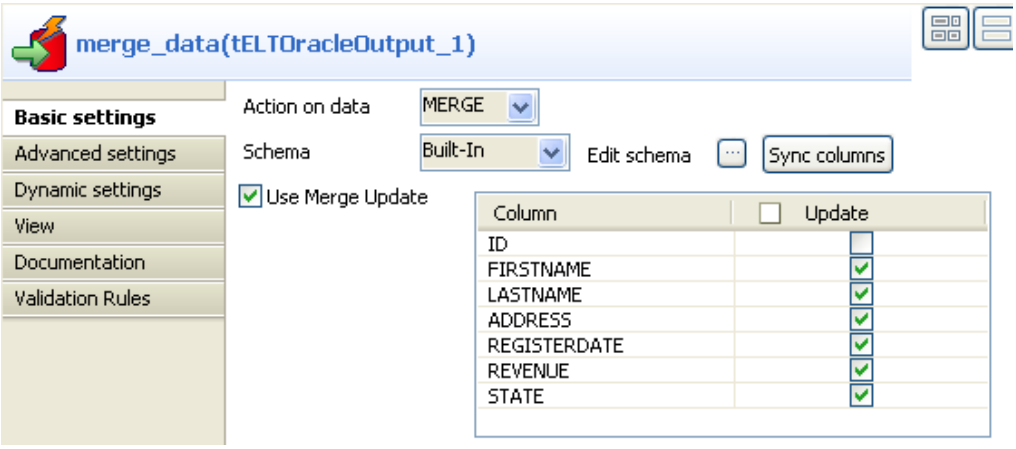


- Click the **Generated SQL Select query tab** to display the query statement to be executed.



- Click **OK** to validate the ELT Map settings and close the ELT Map editor.
- In the design workspace, double-click the **tELTOracleOutput** component to display its **Basic settings** view.
- From the **Action on data** list, select **MERGE**.
- Click the **Sync columns** button to retrieve the schema from the preceding component.
- Select the **Use Merge Update** check box to update the data using Oracle's MERGE function.
- In the table that appears, select the check boxes for the columns you want to update.

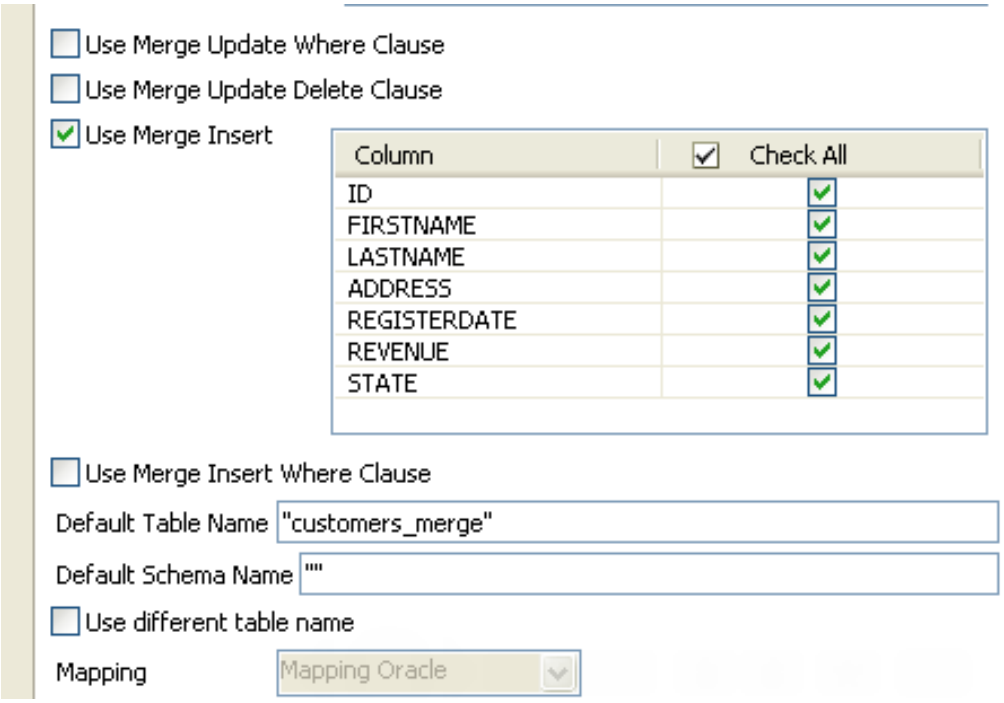
  In this use case, we want to update all the data according to the customer ID. Therefore, select all the check boxes except the one for the *ID* column.



The columns defined as the primary key CANNOT and MUST NOT be made subject to updates.

- Select the **Use Merge Insert** check box to insert new data while updating the existing data by leveraging Oracle's MERGE function.
- In the table that appears, select the check boxes for the columns into which you want to insert new date.

  In this use case, we want to insert all the new customer data. Therefore, select all the check boxes by clicking the **Check All** check box.
- Fill the **Default Table Name** field with the name of the target table already existing in your database. In this example, fill in *customers_merge*.
- Leave the other parameters as they are.



- Save your Job and press **F6** to run it.

  The data is updated and inserted in the database. The query used is displayed on the console.

```
[statistics] connecting to socket on port 3712
[statistics] connected
Merge with :
MERGE INTO customers_merge target USING (SELECT NEW_CUSTOMERS.ID ,
NEW_CUSTOMERS.FIRSTNAME , NEW_CUSTOMERS.LASTNAME , NEW_CUSTOMERS.ADDRESS ,
NEW_CUSTOMERS.REGISTERDATE , NEW_CUSTOMERS.REVENUE , NEW_CUSTOMERS.STATE  FROM
NEW_CUSTOMERS) source ON (target.ID=source.ID) WHEN MATCHED THEN UPDATE SET
target.FIRSTNAME=source.FIRSTNAME,target.LASTNAME=source.LASTNAME,target.ADDRESS=s
ource.ADDRESS,target.REGISTERDATE=source.REGISTERDATE,target.REVENUE=source.REVENU
E,target.STATE=source.STATE  WHEN NOT MATCHED THEN INSERT (
ID,FIRSTNAME,LASTNAME,ADDRESS,REGISTERDATE,REVENUE,STATE) VALUES (
source.ID,source.FIRSTNAME,source.LASTNAME,source.ADDRESS,source.REGISTERDATE,sour
ce.REVENUE,source.STATE)
Merge with :
MERGE INTO customers_merge target USING (SELECT NEW_CUSTOMERS.ID ,
NEW_CUSTOMERS.FIRSTNAME , NEW_CUSTOMERS.LASTNAME , NEW_CUSTOMERS.ADDRESS ,
NEW_CUSTOMERS.REGISTERDATE , NEW_CUSTOMERS.REVENUE , NEW_CUSTOMERS.STATE  FROM
NEW_CUSTOMERS) source ON (target.ID=source.ID) WHEN MATCHED THEN UPDATE SET
target.FIRSTNAME=source.FIRSTNAME,target.LASTNAME=source.LASTNAME,target.ADDRESS=s
ource.ADDRESS,target.REGISTERDATE=source.REGISTERDATE,target.REVENUE=source.REVENU
E,target.STATE=source.STATE  WHEN NOT MATCHED THEN INSERT (
ID,FIRSTNAME,LASTNAME,ADDRESS,REGISTERDATE,REVENUE,STATE) VALUES (
source.ID,source.FIRSTNAME,source.LASTNAME,source.ADDRESS,source.REGISTERDATE,sour
ce.REVENUE,source.STATE)

[statistics] disconnected
```

tELTPostgresqlInput

tELTPostgresqlInput properties

The three ELT Postgresql components are closely related, in terms of their operating conditions. These components should be used to handle Postgresql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Postgresql	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Enter the default table name, between double quotation marks.
	Default Schema Name	Enter the default schema name, between double quotation marks.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTPostgresqlInput** is to be used along with the **tELTPostgresqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTPostgresqlInput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTPostgresqlMap

tELTPostgresqlMap properties

The three ELT Postgresql components are closely related, in terms of their operating conditions. These components should be used to handle Postgresql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Postgresql	
Function	Helps to build the SQL statement graphically, using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate Connection component from the **Component list** if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Postgresql Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.

		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties for the DB connection you are creating. This option is not available if you have selected the **Use an existing connection** check box in the **Basic settings**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTPostgresqlMap** is used along with a **tELTPostgresqlInput** and **tELTPostgresqlOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenario:

For related scenarios, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTPostgresqlOutput

tELTPostgresqlOutput properties

The three ELT Postgresql components are closely related, in terms of their operating conditions. These components should be used to handle Mysql DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Postgresql	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Postgresql database	
Basic settings	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter the default table name between double quotation marks.
	Default Schema Name	Enter the default schema name between double quotation marks
	Use different table name	Select this check box to enter a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTPostgresqlOutput** is to be used along with the **tELTPostgresqlMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTPostgresqlOutput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*

- section *Scenario 2: ELT using an Alias table*

tELTSybaseInput

tELTSybaseInput properties

The three ELT Sybase components are closely related, in terms of their operating conditions. These components should be used to handle Sybase DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Sybase	
Function	Provides the table schema for the SQL statement to execute	
Purpose	Allows you to add as many Input tables as required, for Insert statements which can be complex.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number and nature of the fields to be processed. The schema is either built-in (local) or stored remotely in the Repository. The Schema defined is then passed on to the ELT Mapper for inclusion in the Insert SQL statement. Click on **Edit Schema**, to modify the schema. Note that if you modify the schema, it automatically becomes built-in.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Enter a default name for the table, between double quotation marks.
	Default Schema Name	Enter a default name for the Sybase schema, between double quotation marks.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tELTSybaseInput** is intended for use with **tELTSybaseMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. ELT components only handle schema information. They do not handle actual data flow..	

Related scenarios

For scenarios in which **tELTSybaseInput** may be used, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*.

tELTSybaseMap

tELTSybaseMap properties

The three ELT Sybase components are closely related in terms of their operating conditions. These components should be used to handle Sybase DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Sybase	
Function	Allows you construct a graphical build of the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input to feed the parameters required to execute the SQL statement. The statement can include inner or outer joins to be implemented between tables or between a table and its aliases	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate **tSybaseConnection** component from the **Component** list if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, ensure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Sybase Map Editor	The ELT Map editor allows you to define the output schema and make a graphical build of the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Can be either **Built-in** or **Repository**.
		Built-in : No property data is stored centrally.

		Repository : Select the Repository file where the component properties are stored. The following fields are pre-filled using collected data
	Host	Database server IP address
	Port	Listening port number of DB server
	Database	Name of the database
	Username et *Password*	DB user authentication data.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at component level.
Usage	**tELTSybaseMap** is intended for use with **tELTSybaseInput** and **tELTSybaseOutput**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. The ELT components only handle schema information. They do not handle actual data flow.	

Related scenarios

For scenarios in which **tELTSybaseMap** may be used, see the following **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTSybaseOutput

tELTSybaseOutput properties

The three ELT Sybase components are closely related in terms of their operating conditions. These components should be used to handle Sybase DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Componant family	ELT/Map/Sybase	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement in the Mysql database	
Basic settings *Use tCreate Table as substitute for this function.*	*Action on data*	On the data of the table defined, you can perform the following operation: Insert: Add new entries to the table. If duplicates are found, the Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number and nature of the fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. The Schema defined is then passed on to the ELT Mapper for inclusion in the Insert SQL statement. Click on **Edit Schema**, to modify the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter a default name for the table, between double quotation marks.
	Default Schema Name	Enter a default name for the Sybase schema, between double quotation marks.
	Use different table name	Select this check box to enter a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at component level.
Usage	**tELTSybaseOutput** is intended for use with the **tELTMysqlInput** and **tELTSybaseMap** components. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name.. ELT components only handle schema information. They do not handle actual data flow.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For scenarios in which **tELTSybaseOutput** may be used, see the following **tELTMysqlMap** scenarios :

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTTeradataInput

tELTTeradataInput properties

The three ELT Teradata components are closely related, in terms of their operating conditions. These components should be used to handle Teradata DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Teradata	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows you to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement. Click **Edit Schema** to modify the schema. Note that if you make the modifcation, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Default Table Name	Enter a default name for the table, between double quotation marks.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at component level.
Usage	**tELTTeradataInput** is to be used along with the **tELTTeradataMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For use cases in relation with **tELTTeradataInput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*
- section *Scenario 2: ELT using an Alias table*

tELTTeradataMap

tELTTeradataMap properties

The three ELT Teradata components are closely related, in terms of their operating conditions. These components should be used to handle Teradata DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Teradata	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Use an existing connection*	Select this check box and select the appropriate **tTeradataConnection** component from the **Component** list if you want to re-use connection parameters that you have already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, make sure that the available connection components are sharing the intended connection. For more information on how to share a DB connection across Job levels, see **Use or register a shared DB connection** in any database connection component corresponding to the database you are using, in *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components*. Otherwise, you can as well deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	ELT Teradata Map editor	The ELT Map editor allows you to define the output schema as well as build graphically the SQL statement to be executed. The column names of schema can be different from the column names in the database.
	Style link	Select the way in which links are displayed. **Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Bezier curve**: Links between the schema and the Web service parameters are in the form of curve. **Line**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Property type	Either Built-in or Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
Usage	**tELTTeradataMap** is used along with a **tELTTeradataInput** and **tELTTeradataOutput**. Note that the Output link to be used with these components must faithfully reflect the name of the tables. The ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

For detailed information regarding ELT component connections, see section *Connecting ELT components*.

Related topic: see *Talend Open Studio for Big Data User Guide*.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

For detailed information regarding the table schema mapping and joining, see section *Mapping and joining tables*.

Adding WHERE clauses

For details regarding the clause handling, see section *Adding where clauses*.

Generating the SQL statement

The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.

The clause defined internally in the ELT Mapper are also included automatically.

Related scenarios

For use cases in relation with **tELTTeradataMap**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tELTTeradataOutput

tELTTeradataOutput properties

The three ELT Teradata components are closely related, in terms of their operating conditions. These components should be used to handle Teradata DB schemas to generate Insert statements, including clauses, which are to be executed in the DB output table defined.

Component family	ELT/Map/Teradata	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert, Update and Delete statement to the Teradata database	
Basic settings *Use tCreate Table as substitute for this function.*	*Action on data*	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Updates entries in the table. **Delete**: Deletes the entries which correspond to the entry flow.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to modify the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where clauses for (for UPDATE and DELETE only)	Enter a clause to filter the data to be updated or deleted during the update or delete operations.
	Default Table Name	Enter a default name for the table, between double quotation marks.
	Use different table name	Select this check box to enter a different output table name, between double quotation marks, in the **Table name** field which appears.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at component level.
Usage	**tELTTeradataOutput** is to be used along with the **tELTTeradataMap**. Note that the Output link to be used with these components must correspond strictly to the syntax of the table name. Note that the ELT components do not handle actual data flow but only schema information.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For use cases in relation with **tELTTeradataOutput**, see **tELTMysqlMap** scenarios:

- section *Scenario 1: Aggregating table columns and filtering*.
- section *Scenario 2: ELT using an Alias table*.

tSQLTemplateAggregate

tSQLTemplateAggregate properties

Component family	ELT/SQLTemplate	
Function	**tSQLTemplateAggregate** collects data values from one or more columns with the intent to manage the collection as a single unit. This component has real-time capabilities since it runs the data transformation on the DBMS itself.	
Purpose	Helps to provide a set of matrix based on values or calculations.	
Basic settings	*Database Type*	Select the database type you want to connect to from the list.
	Component List	Select the relevant DB connection component in the list if you use more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to collect values from.
	Target table name	Name of the table you want to write the collected and transformed data in.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select any of the following operations to perform on data: **count**, **min**, **max**, **avg**, **sum**, and **count (distinct)**.
		Input column position: Select the input column from which you want to collect the values to be aggregated.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered according to the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations.
		Input Column position: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
SQL Template	*SQL Template List*	**To add a default system SQL template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by **Talend**.

		Note: You can create your own SQL template and add them to the **SQL Template List**. **To create a user-defined SQL template:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code. -Click the **Add** button to add the new created template to the **SQL Template list**. For more information, see *Talend Open Studio for Big Data User Guide*.
Usage	This component is used as an intermediate component with other relevant DB components, especially the DB connection and commit components.	
Limitation	n/a	

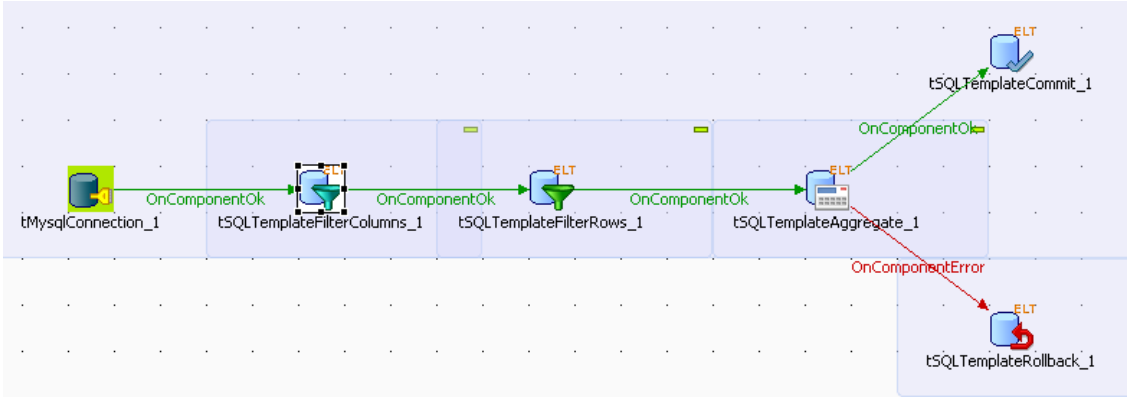
Scenario: Filtering and aggregating table columns directly on the DBMS

The following scenario creates a Job that opens a connection to a Mysql database and:

- instantiates the schemas from a database table whose rows match the column names specified in the filter,
- filters a column in the same database table to have only the data that matches a WHERE clause,
- collects data grouped by specific value(s) from the filtered column and writes aggregated data in a target database table.

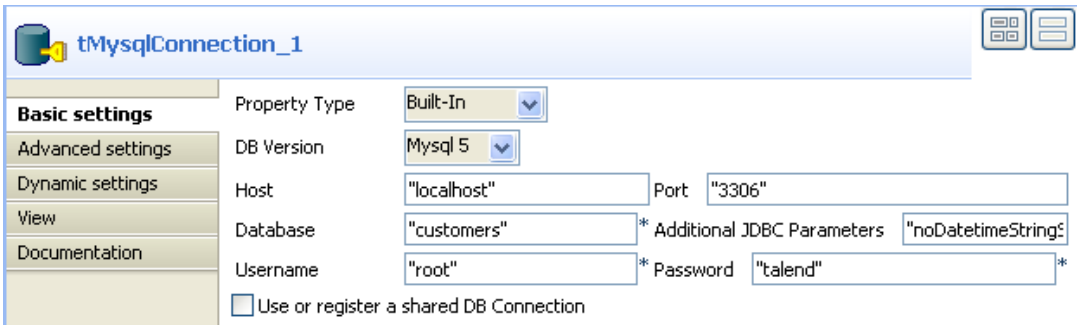
To filter and aggregate database table columns:

- Drop the following components from the **Palette** onto the design workspace: **tELTMysqlconnection**, **tSQLTemplateFilterColumns**, **tSQLTemplateFilterRows**, **tSQLTemplateAggregate**, **tSQLTemplateCommit**, and **tSQLTemplateRollback**.
- Connect the five first components using **OnComponentOk** links.
- Connect **tSQLTemplateAggregate** to **tSQLTemplateRollback** using an **OnComponentError** link.

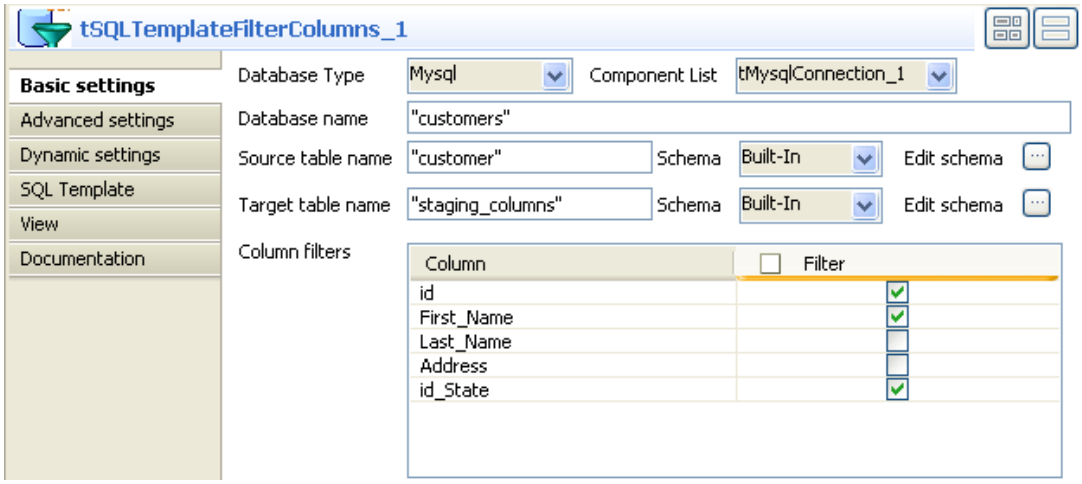


- In the design workspace, select **tMysqlConnection** and click the **Component** tab to define the basic settings for **tMysqlConnection**.
- In the **Basic settings** view, set the database connection details manually or select **Repository** from the **Property Type** list and select your DB connection if it has already been defined and stored in the **Metadata** area of the **Repository** tree view.

For more information about Metadata, see *Talend Open Studio for Big Data User Guide*.



- In the design workspace, select **tSQLTemplateFilterColumns** and click the **Component** tab to define its basic settings.



- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

When you define the data structure for the source table, column names automatically appear in the **Column** list in the **Column filters** panel.

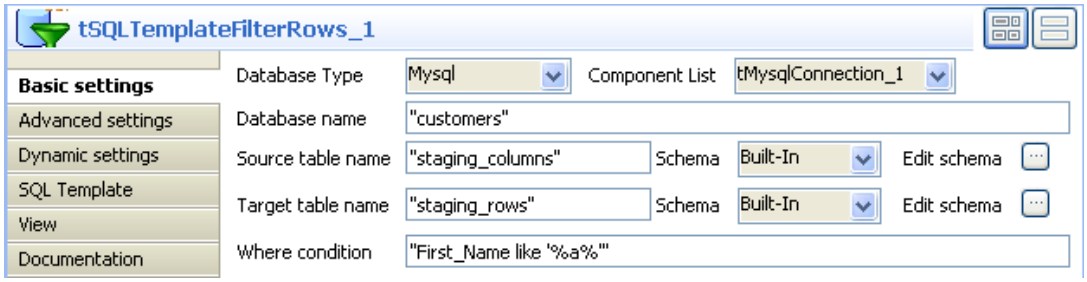
In this scenario, the source table has five columns: *id*, *First_Name*, *Last_Name*, *Address*, and *id_State*.

- In the **Column filters** panel, set the column filter by selecting the check boxes of the columns you want to write in the source table.

In this scenario, the **tSQLTemplateFilterColumns** component instantiates only three columns: *id*, *First_Name*, and *id_State* from the source table.

In the **Component** view, you can click the **SQL Template** tab and add system SQL templates or create your own and use them within your Job to carry out the coded operation. For more information, see section *tSQLTemplateFilterColumns Properties*.

- In the design workspace, select **tSQLTemplateFilterRows** and click the **Component** tab to define its basic settings.



- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

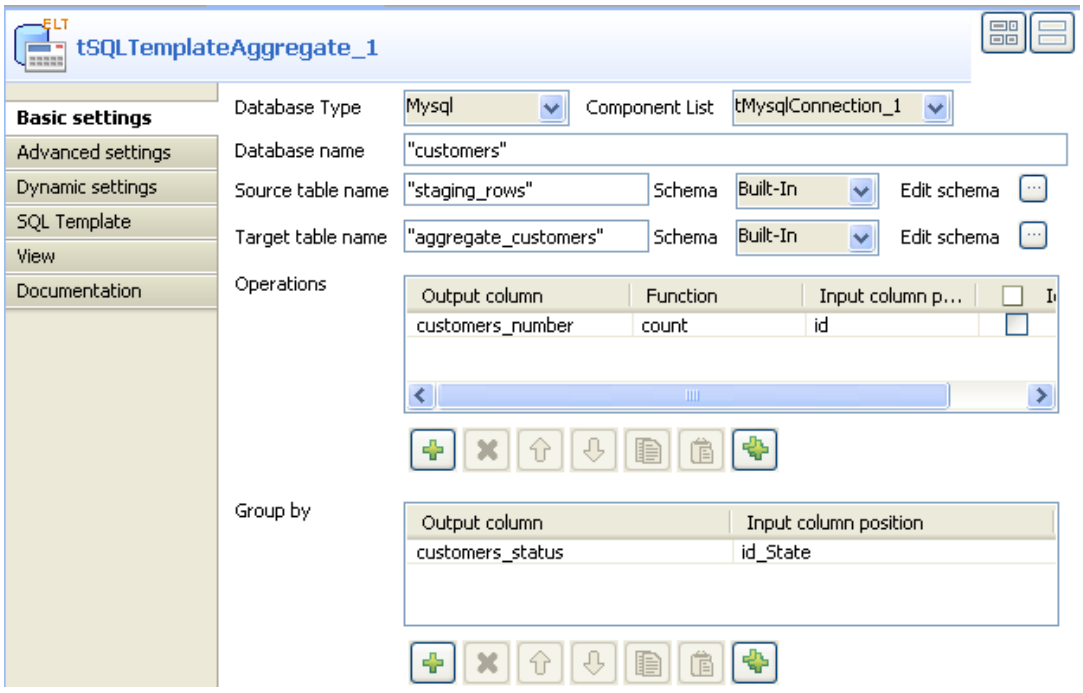
In this scenario, the source table has the three initially instantiated columns: *id*, *First_Name*, and *id_State* and the source table has the same three-column schema.

- In the **Where condition** field, enter a WHERE clause to extract only those records that fulfill the specified criterion.

In this scenario, the **tSQLTemplateFilterRows** component filters the *First_Name* column in the source table to extract only the first names that contain the “a” letter.

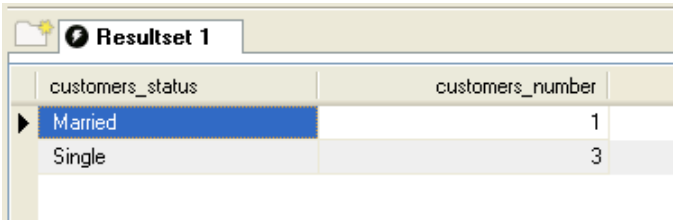
- In the design workspace, select **tSQLTemplateAggregate** and click the **Component** tab to define its basic settings.
- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

The schema for the source table consists of the three columns: *id*, *First_Name*, and *id_State*. The schema for the target table consists of two columns: *customers_status* and *customers_number*. In this scenario, we want to group customers by their marital status and count customer number in each marital group. To do that, we define the **Operations** and **Group by** panels accordingly.



- In the **Operations** panel, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the counted data.
- Click in the **Function** line and select the operation to be carried on.
- In the **Group by** panel, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the aggregated data.
- In the design workspace, select **tSQLTemplateCommit** and click the **Component** tab to define its basic settings.
- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Do the same for **tSQLTemplateRollback**.
- Save your Job and press **F6** to execute it.

A two-column table *aggregate_customers* is created in the database. It groups customers according to their marital status and count customer number in each marital group.



tSQLTemplateCommit

tSQLTemplateCommit properties

This component is closely related to **tSQLTemplateRollback** and to the ELT connection component for the database you work with. **tSQLTemplateCommit**, **tSQLTemplateRollback** and the ELT database connection component are usually used together in a transaction.

Component family	ELT/SQLTemplate	
Function	**tSQLTemplateCommit** validates the data processed in a Job in a specified database.	
Purpose	Using a single connection, this component commits a global action in one go instead of doing so for every row or every batch of rows, separately. This provides a gain in performance.	
Basic settings	*Database Type*	Select the database type you want to connect to from the list.
	Component List	Select the ELT database connection component in the list if more than one connection is required for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
SQL Template	*SQL Template List*	**To add a default system SQL template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by Talend. **Note:** You can create your own SQL template and add them to the **SQL Ttemplate List**. **To create a user-defined SQL template:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code. -Click the **Add** button to add the new created template to the **SQL Template list**. For more information, see *Talend Open Studio for Big Data* User Guide.
Usage	This component is to be used with ELT components, especially with **tSQLTemplateRollback** and the relevant database connection component.	
Limitation	n/a	

Related scenario

This component is closely related to **tSQLTemplateRollback** and to the ELT connection component depending on the database you are working with. It usually does not make much sense to use ELT components without using the relevant ELT database connection component as its purpose is to open a connection for a transaction.

For more information on **tSQLTemplateCommit**, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tSQLTemplateFilterColumns

tSQLTemplateFilterColumns Properties

Component family	ELT/SQLTemplate	
Function	**tSQLTemplateFilterColumns** makes specified changes to the defined schema of the database table based on column name mapping. This component has real-time capabilities since it runs the data filtering on the DBMS itself	
Purpose	Helps homogenize schemas by reorganizing, deleting or adding new columns.	
Basic settings	*Database Type*	Select the type of database you want to work on from the drop-down list.
	Component List	Select the relevant DB connection component in the list if you use more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to filter.
	Target table name	Name of the table you want to write the filtered data in.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Column Filters	In the table, click the **Filter** check box to filter all of the columns. To select specific columns for filtering, select the check box(es) which correspond(s) to the column name(s).
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
SQL Template	*SQL Template List*	**To add a default system SQL Template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by Talend. **Note:** You can create your own SQL templates and add them to the **SQL Template List**. **To create a user-defined SQL list:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL Template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code. -Click the **Add** button to add the new created template to the **SQL Template list**.

		For more information, see *Talend Open Studio for Big Data User Guide*.
Usage	This component is used as an intermediary component with other relevant DB components, especially DB connection components.	
Limitation	n/a	

Related Scenario

For a related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tSQLTemplateFilterRows

tSQLTemplateFilterRows Properties

Component family	ELT/SQLTemplate	
Function	**tSQLTemplateFilterRows** allows you to define a row filter on one table. This component has real-time capabilities since it runs the data filtering on the DBMS itself.	
Purpose	Helps to set row filters for any given data source, based on a WHERE clause.	
Basic settings	*Database Type*	Select the type of database you want to work on from the drop down list.
	Component List	Select the relevant DB connection component in the list if you are using more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to filter.
	Target table name	Name of the table you want to write the filtered data in.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Where condition	Use a WHERE clause to set the criteria that you want the rows to meet. You can use the WHERE clause to select specific rows from the table that match specified criteria or conditions.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
SQL Template	*SQL Template List*	**To add a default system SQL template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by Talend. **Note:** You can create your own SQL template and add them to the **SQL Template List**. **To create a user-defined SQL template:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code.

		-Click the **Add** button to add the new created template to the **SQL Template list**. For more information, see *Talend Open Studio for Big Data User Guide*.
Usage	This component is used as an intermediary component with other DB components, particularly DB connection components.	
Limitation	n/a	

Related Scenario

For a related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

tSQLTemplateMerge

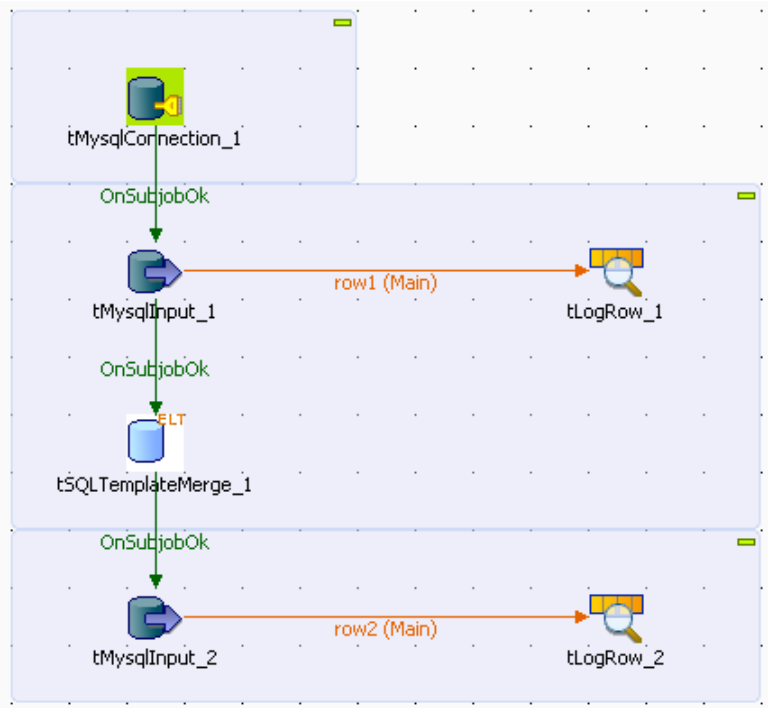
tSQLTemplateMerge properties

Component family	ELT/SQLTemplate	
Function	This component creates an SQL MERGE statement to merge data into a database table.	
Purpose	This component is used to merge data into a database table directly on the DBMS by creating and executing a MERGE statement.	
Basic settings	*Database Type*	Select the type of database you want to work on from the drop-down list.
	Component list	Select the relevant DB connection component from the list if you use more than one connection in the current Job.
	Source table name	Name of the database table holding the data you want to merge into the target table.
	Target table name	Name of the table you want to merge data into.
	Schema and *Edit schema*	This component involves two schemas: source schema and target schema. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to modify the schema. Note that if you make the modification, the schema switches automatically to the **Built-in** mode.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Merge ON	Specify the target and source columns you want to use as the primary keys.
	Use UPDATE (WHEN MATCHED)	Select this check box to update existing records. With the check box selected, the **UPDATE Columns** table appears, allowing you to define the columns in which records are to be updated.
	Specify additional output columns	Select this check box to update records in additional columns other than those listed in the **UPDATE Columns** table. With this check box selected, the **Additional UPDATE Columns** table appears, allowing you to specify additional columns.
	Specify UPDATE WHERE clause	Select this check box and type in a WHERE clause in the **WHERE clause** field to filter data during the update operation. This option may not work with certain database versions, including Oracle 9i.
	Use INSERT (WHEN MATCHED)	Select this check box to insert new records. With the check box selected, the **INSERT Columns** table appears, allowing you to specify the columns to be involved in the insert operation.
	Specify additional output columns	Select this check box to insert records to additional columns other than those listed in the **INSERT Columns** table. With this check box selected, the **Additional INSERT Columns** table appears, allowing you to specify additional columns.
	Specify INSERT WHERE clause	Select this check box and type in a WHERE clause in the **WHERE clause** field to filter data during the insert operation. This option may not work with certain database versions, including Oracle 9i.

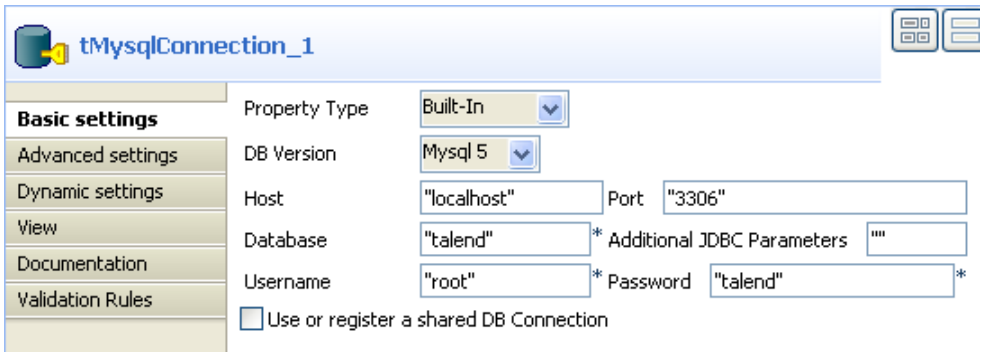
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at component level.
SQL Template	*SQL Template List*	**To add a default system SQL template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by **Talend**. **Note:** You can create your own SQL template and add them to the **SQL Template List**. **To create a user-defined SQL template:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code. -Click the **Add** button to add the new created template to the **SQL Template list**. For more information, see *Talend Open Studio for Big Data User Guide*.
Usage	This component is used as an intermediate component with other relevant DB components, especially the DB connection and commit components.	

Scenario: Merging data directly on the DBMS

This scenario describes a simple Job that opens a connection to a MySQL database, merges data from a source table into a target table according to customer IDs, and displays the contents of the target table before and after the merge action. A WHERE clause is used to filter data during the merge operation.



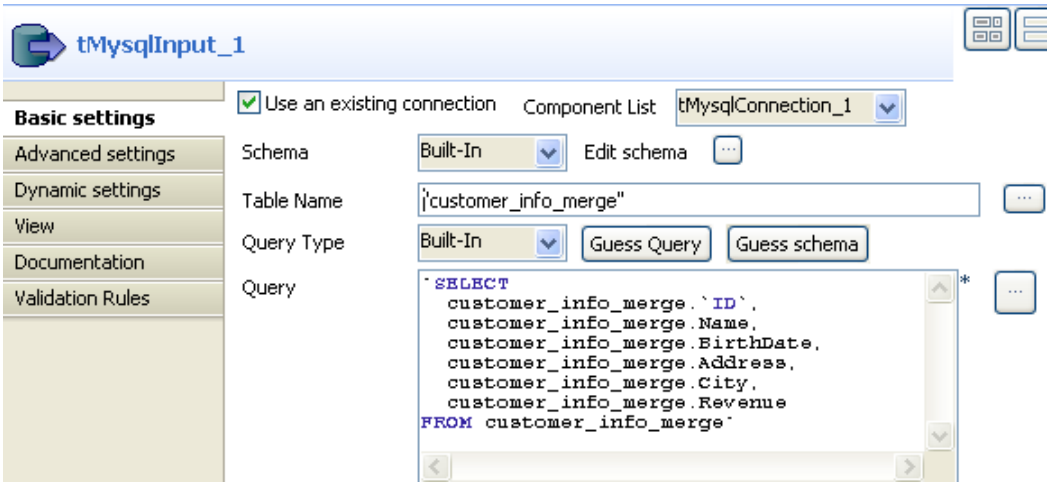
- Drop a **tMysqlConnection** component, a **tSQLTemplateMerge** component, two **tMysqlInput** components and two **tLogRow** components from the **Palette** onto the design workspace.
- Connect the **tMysqlConnection** component to the first **tMysqlInput** component using a **Trigger** > **OnSubjobOK** connection.
- Connect the first **tMysqlInput** component to the first **tLogRow** component using a **Row** > **Main** connection. This row will display the initial contents of the target table on the console.
- Connect the first **tMysqlInput** component to the **tSQLTemplateMerge** component, and the **tSQLTemplateMerge** component to the second **tMysqlInput** component using **Trigger** > **OnSubjobOK** connections.
- Connect the second **tMysqlInput** component to the second **tLogRow** component using a **Row** > **Main** connection. This row will display the merge result on the console.
- Double-click the **tMysqlConnection** component to display its **Basic settings** view.



- Set the database connection details manually or select **Repository** from the **Property Type** list and select your DB connection if it has already been defined and stored in the **Metadata** area of the **Repository** tree view.

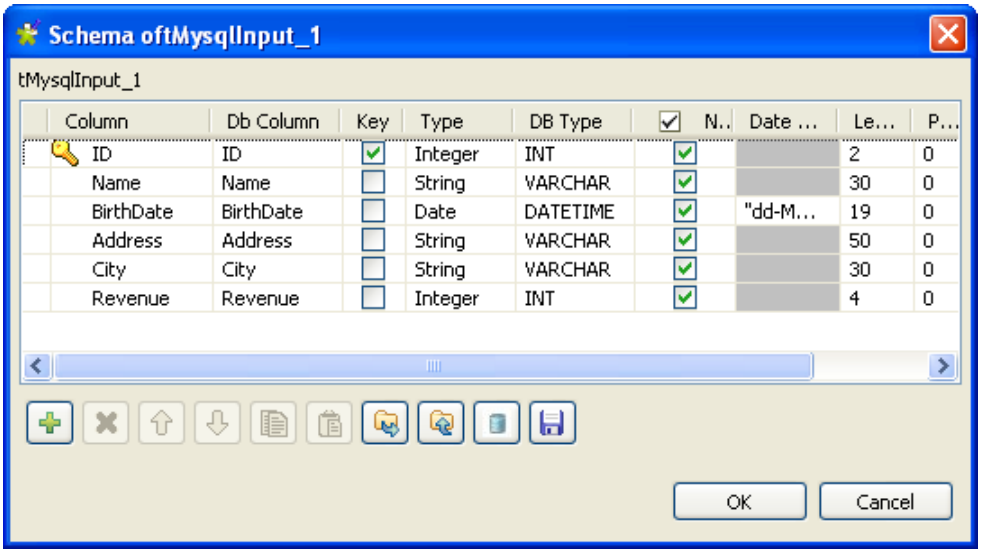
  For more information about Metadata, see *Talend Open Studio for Big Data* **User Guide**.

- Double-click the first **tMysqlInput** component to display its **Basic settings** view.

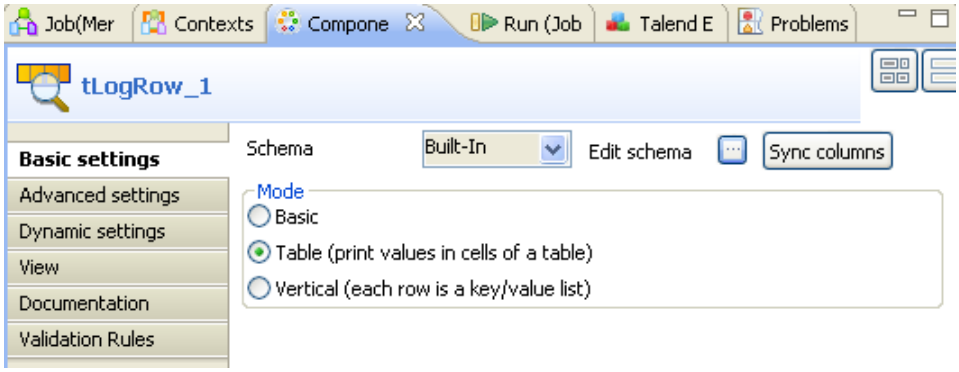


- Select the **Use an existing connection** check box. If you are using more than one DB connection component in your Job, select the component you want to use from the **Component List**.
- Click the three-dot button next to **Edit schema** and define the data structure of the target table, or select **Repository** from the **Schema** list and select the target table if the schema has already been defined and stored in the **Metadata** area of the **Repository** tree view.

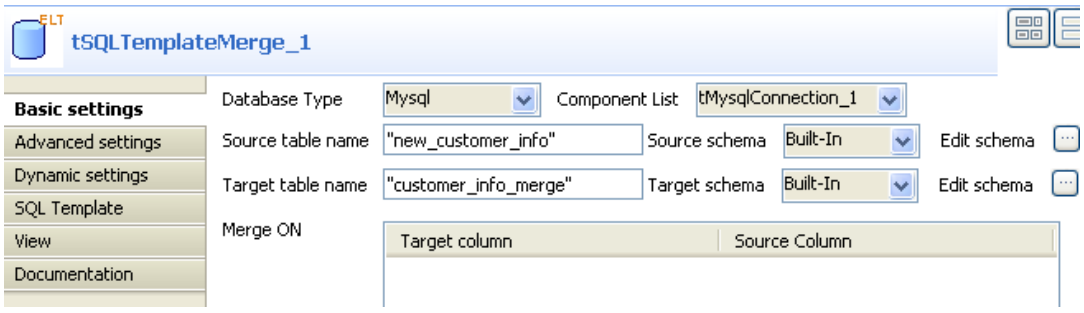
In this scenario, we use built-in schemas.



- Define the columns as shown above, and then click **OK** to propagate the schema structure to the output component and close the schema dialog box.
- Fill the **Table Name** field with the name of the target table, *customer_info_merge* in this scenario.
- Click the **Guess Query** button, or type in `"SELECT * FROM customer_info_merge"` in the **Query** area, to retrieve all the table columns.
- Define the properties of the second **tMysqlInput** component, using exactly the same settings as for the first **tMysqlInput** component.
- In the **Basic settings** view of each **tLogRow** component, select the **Table** option in the **Mode** area so that the contents will be displayed in table cells on the console.



- Double-click the **tSQLTemplateMerge** component to display its **Basic settings** view.



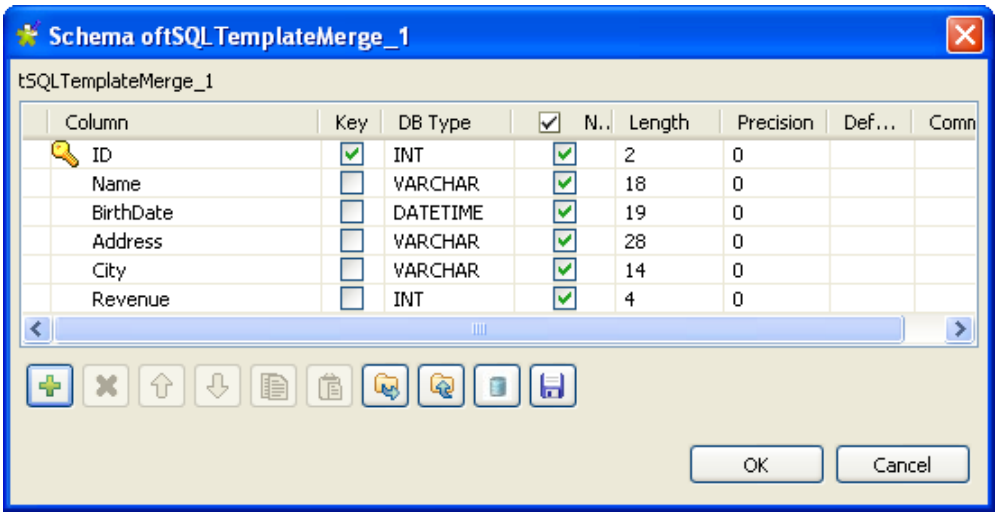
- Type in the names of the source table and the target table in the relevant fields.

  In this scenario, the source table is *new_customer_info*, which contains eight records; the target table is *customer_info_merge*, which contains five records, and both tables have the same data structure.

The source table and the target table may have different schema structures. In this case, however, make sure that the source column and target column specified in each line of the **Merge ON** table, the **UPDATE Columns** table, and the **INSERT Columns** table are identical in data type and the target column length allows the insertion of the data from the corresponding source column.

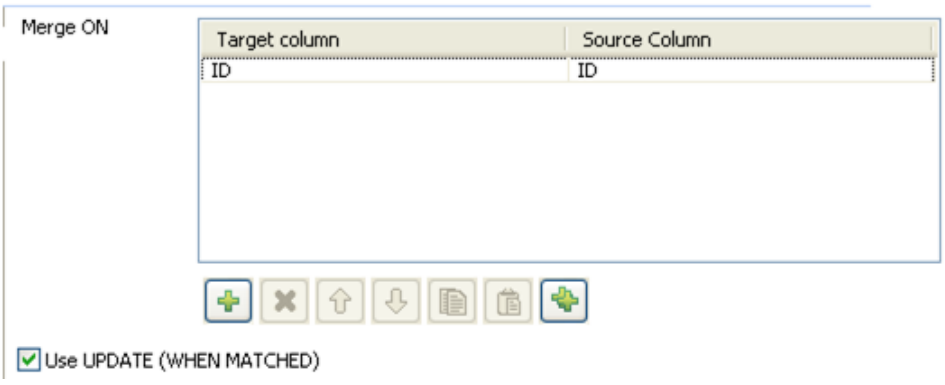
- Define the source schema manually, or select **Repository** from the **Schema** list and select the relevant table if the schema has already been defined and stored in the **Metadata** area of the **Repository** tree view.

  In this scenario, we use built-in schemas.



- Define the columns as shown above and click **OK** to close the schema dialog box, and do the same for the target schema.

- Click the green plus button beneath the **Merge ON** table to add a line, and select the *ID* column as the primary key.

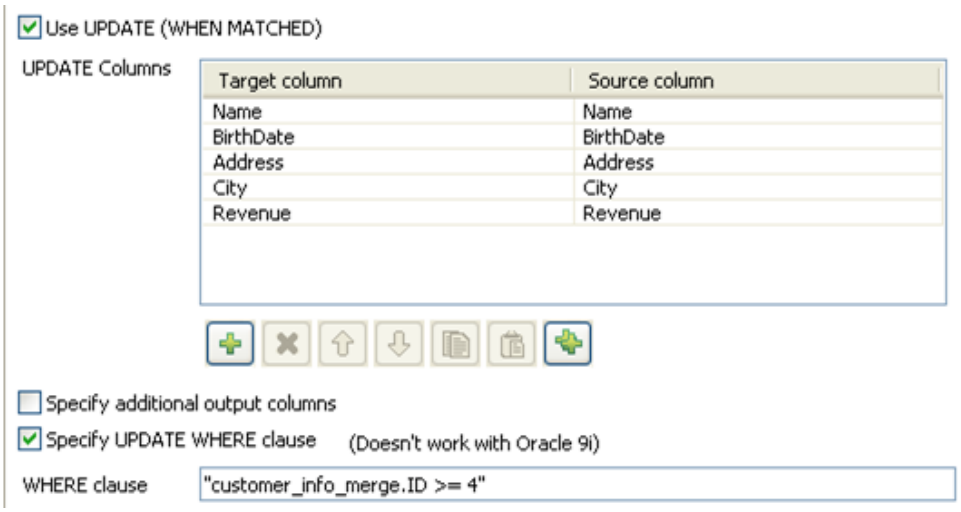


- Select the **Use UPDATE** check box to update existing data during the merge operation, and define the columns to be updated by clicking the green plus button and selecting the desired columns.

  In this scenario, we want to update all the columns according to the customer IDs. Therefore, we select all the columns except the *ID* column.

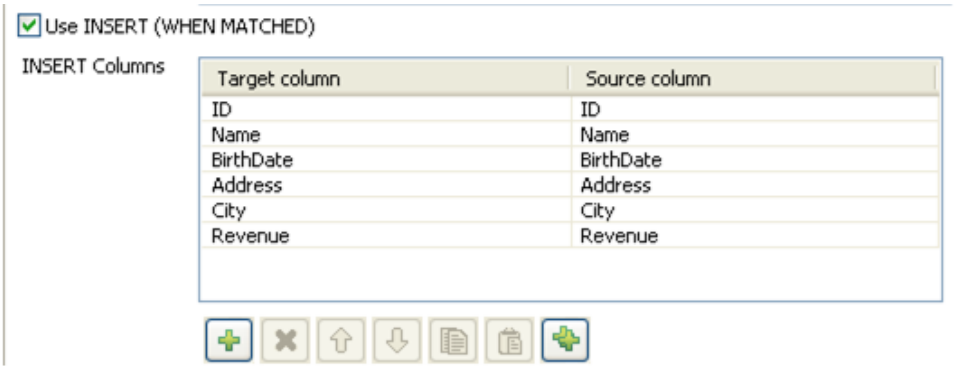
The columns defined as the primary key CANNOT and MUST NOT be made subject to updates.

- Select the **Specify UPDATE WHERE clause** check box and type in `customer_info_merge.ID >= 4` within double quotation marks in the **WHERE clause** field so that only those existing records with an ID equal to or greater than 4 will be updated.



- Select the **Use INSERT** check box and define the columns to take data from and insert data to in the **INSERT Columns** table.

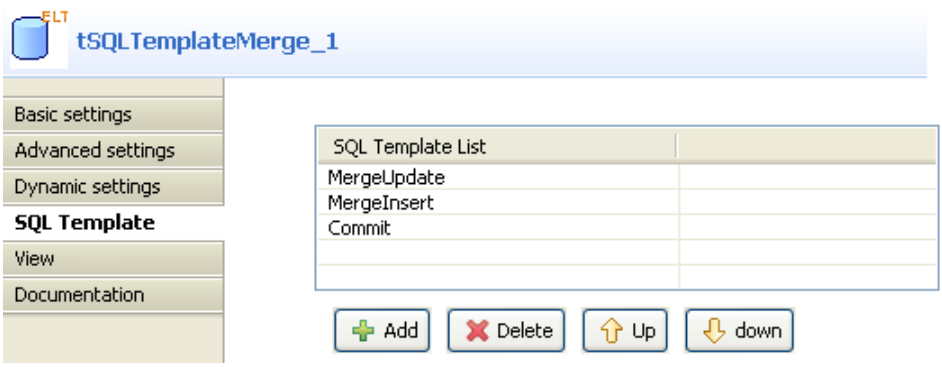
  In this example, we want to insert all the records that do not exist in the target table.



- Select the **SQL Template** view to display and add the SQL templates to be used.

  By default, the **SQLTemplateMerge** component uses two system SQL templates: **MergeUpdate** and **MergeInsert**.

In the **SQL Template** tab, you can add system SQL templates or create your own and use them within your Job to carry out the coded operation. For more information, see section *tSQLTemplateFilterColumns Properties*.

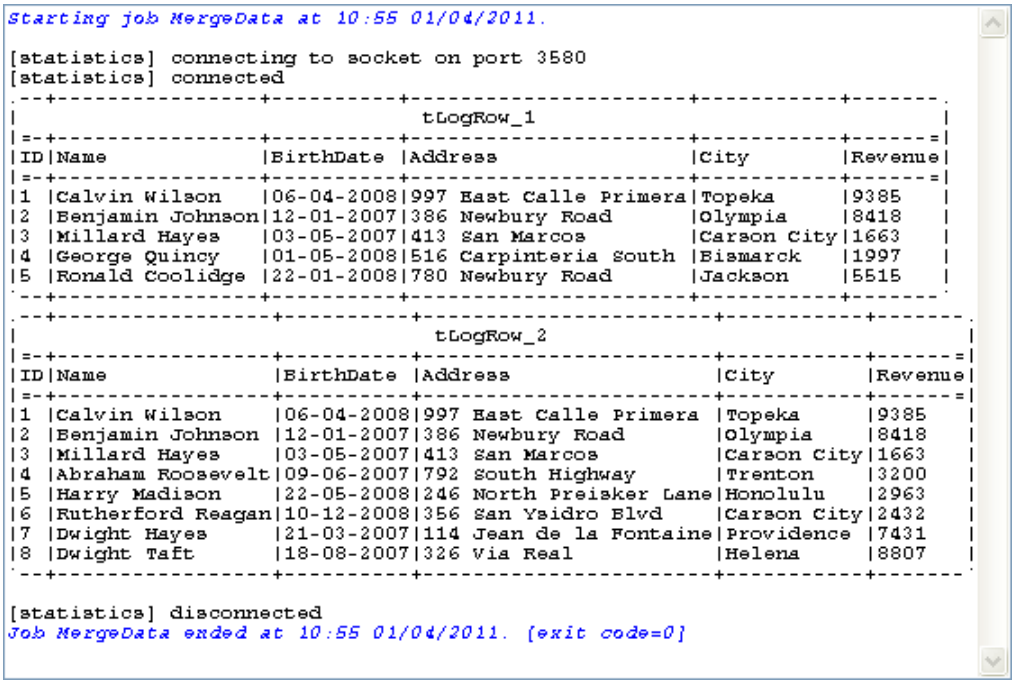


- Click the **Add** button to add a line and select **Commit** from the template list to commit the merge result to your database.

  Alternatively, you can connect the **tSQLTemplateMerge** component to a **tSQLTemplateCommit** or **tMysqlCommit** component using a **Trigger** > **OnSubjobOK** connection to commit the merge result to your database.

- Save your Job and press **F6** to run it.

  Both the original contents of the target table and the merge result are displayed on the console. In the target table, records No. 4 and No. 5 contain the updated information, and records No.6 through No. 8 contain the inserted information.



```
Starting job MergeData at 10:55 01/04/2011.

[statistics] connecting to socket on port 3580
[statistics] connected
.--+----------------+----------+---------------------+-----------+-------.
|                                tLogRow_1                                 |
|=-+----------------+----------+---------------------+-----------+------=|
|ID|Name            |BirthDate |Address              |City       |Revenue|
|=-+----------------+----------+---------------------+-----------+------=|
|1 |Calvin Wilson   |06-04-2008|997 East Calle Primera|Topeka     |9385   |
|2 |Benjamin Johnson|12-01-2007|386 Newbury Road      |Olympia    |8418   |
|3 |Millard Hayes   |03-05-2007|413 San Marcos        |Carson City|1663   |
|4 |George Quincy   |01-05-2008|516 Carpinteria South |Bismarck   |1997   |
|5 |Ronald Coolidge |22-01-2008|780 Newbury Road      |Jackson    |5515   |
'--+----------------+----------+---------------------+-----------+-------'
.--+----------------+----------+---------------------+-----------+-------.
|                                tLogRow_2                                 |
|=-+----------------+----------+---------------------+-----------+------=|
|ID|Name            |BirthDate |Address              |City       |Revenue|
|=-+----------------+----------+---------------------+-----------+------=|
|1 |Calvin Wilson   |06-04-2008|997 East Calle Primera|Topeka     |9385   |
|2 |Benjamin Johnson|12-01-2007|386 Newbury Road      |Olympia    |8418   |
|3 |Millard Hayes   |03-05-2007|413 San Marcos        |Carson City|1663   |
|4 |Abraham Roosevelt|09-06-2007|792 South Highway    |Trenton    |3200   |
|5 |Harry Madison   |22-05-2008|246 North Preisker Lane|Honolulu  |2963   |
|6 |Rutherford Reagan|10-12-2008|356 San Ysidro Blvd  |Carson City|2432   |
|7 |Dwight Hayes    |21-03-2007|114 Jean de la Fontaine|Providence|7431   |
|8 |Dwight Taft     |18-08-2007|326 Via Real         |Helena     |8807   |
'--+----------------+----------+---------------------+-----------+-------'

[statistics] disconnected
Job MergeData ended at 10:55 01/04/2011. [exit code=0]
```

tSQLTemplateRollback

tSQLTemplateRollback properties

This component is closely related to **tSQLTemplateCommit** and to the ELT connection component relative to the database you work with. **tSQLTemplateRollback**, **tSQLTemplateCommit** and the ELT database connection component are usually used together in a transaction.

Component family	ELT/SQLTemplate	
Function	**tSQLTemplateRollback** cancels the transaction committed in the database you connect to.	
Purpose	To avoid committing transactions accidentally.	
Basic settings	*Database Type*	Select the database type you want to connect to from the list.
	Component List	Select the ELT database connection component in the list if more than one connection is planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
SQL Template	*SQL Template List*	**To add a default system SQL template:** Click the **Add** button to add the default system SQL template(s) in the **SQL Template List**. Click in the SQL template field and then click the arrow to display the system SQL template list. Select the desired system SQL template provided by Talend. **Note:** You can create your own SQL template and add them to the **SQL Template List**. **To create a user-defined SQL template:** -Select a system template from the **SQL Template list** and click on its code in the code box. You will be prompted by the system to create a new template. -Click **Yes** to open the SQL template wizard. -Define your new SQL template in the corresponding fields and click **Finish** to close the wizard. An SQL template editor opens where you can enter the template code. -Click the **Add** button to add the new created template to the **SQL Template list**. For more information, see *Talend Open Studio for Big Data* **User Guide**.
Usage	This component is to be used with ELT components, especially with **tSQLTemplateCommit** and the relevant database connection component.	
Limitation	n/a	

Related scenarios

For a **tSQLTemplateRollback** related scenario, see section *Scenario: Filtering and aggregating table columns directly on the DBMS*.

ESB components

This chapter details the main components that you can find in the **ESB** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The ESB component family groups together the components dedicated to ESB related tasks.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tESBConsumer

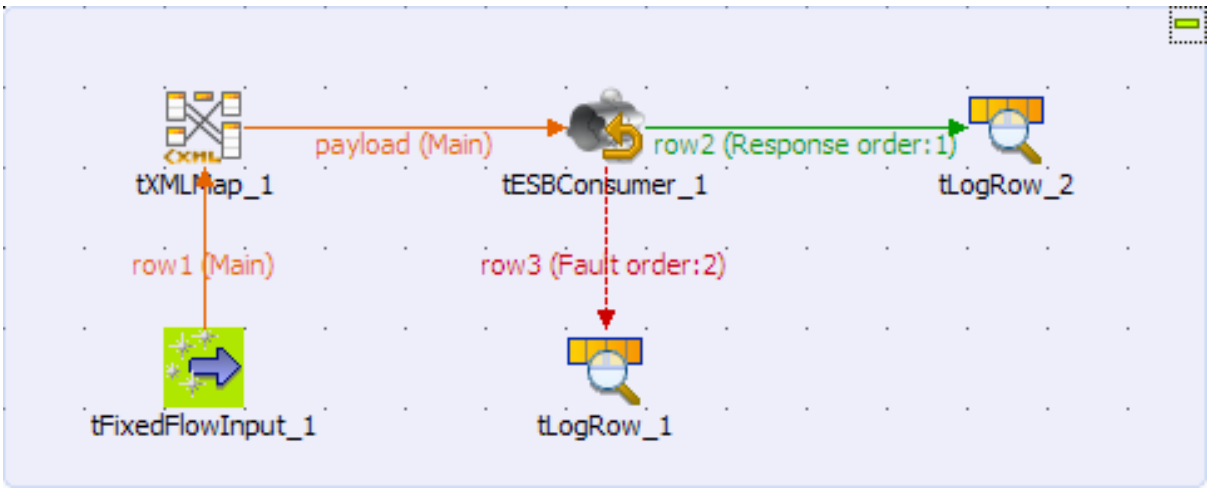
tESBConsumer properties

Component family	ESB/Web Services	
Function	Calls the defined method from the invoked Web service and returns the class as defined, based on the given parameters.	
Purpose	Invokes a Method through a Web service.	
Basic settings	*Property Type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Service configuration	Description of Web service bindings and configuration. The **Endpoint** field gets filled in automatically upon completion of the service configuration.
	Connection time out(second)	Set a value in seconds for Web service connection time out.
	Receive time out(second)	Set a value in seconds for server answer.
	Input Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Response Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Fault Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	ESB Service Settings	**Use Service Locator**: Maintains the availability of the service to help meet demands and service level agreements (SLAs).
		Use Service Activity Monitor: Captures events and stores this information to facilitate in-depth analysis of service activity and track-and-trace of messages throughout a business transaction. This can be used to analyze service response times, identify traffic patterns, perform root cause analysis and more.
		Use Authentication: Select this check box to enable the authentication option. Select from **Basic HTTP**, **SAML Token (ESB runtime only)** and **Username Token**. Enter a username and a password in the corresponding fields as required. Authentication

		with the username token works both from the studio and at runtime. Authentication with the SAML token works at runtime only.
	Use http proxy/Proxy host, *Proxy port*, *Proxy user*, and *Proxy password*	Select this check box if you are using a proxy server and fill in the necessary information.
	Trust server with SSL/ TrustStore file and *TrustStore password*	Select this check box to validate the server certificate to the client via an SSL protocol and fill in the corresponding fields: **TrustStore file**: Enter the path (including filename) to the certificate TrustStore file that contains the list of certificates that the client trusts. **TrustStore password**: Enter the password used to check the integrity of the TrustStore data.
	Mapping links display as	**Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Curves**: Links between the schema and the Web service parameters are in the form of curves. **Lines (fast)**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Die on error	Select this check box to kill the Job when an error occurs.
Advanced settings	*Service Locator Custom Properties*	This table appears when **Use Service Locator** is selected. You can add as many lines as needed in the table to customize the relevant properties. Enter the name and the value of each property between double quotation marks in the **Property Name** field and the **Property Value** field respectively.
	Service Activity Custom Properties	This table appears when **Use Service Activity Monitor** is selected. You can add as many lines as needed in the table to customize the relevant properties. Enter the name and the value of each property between double quotation marks in the **Property Name** field and the **Property Value** field respectively.
	Temporary folder (for wsdl2java)	Set or browse to a temporary folder that you configured in order to store the WSDL files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as an intermediate component. It requires to be linked to an output component.	
Limitation	A JDK is required for this component to operate.	

Scenario: Returning valid email

This scenario describes a Job that uses a **tESBConsumer** component to retrieve the valid email.



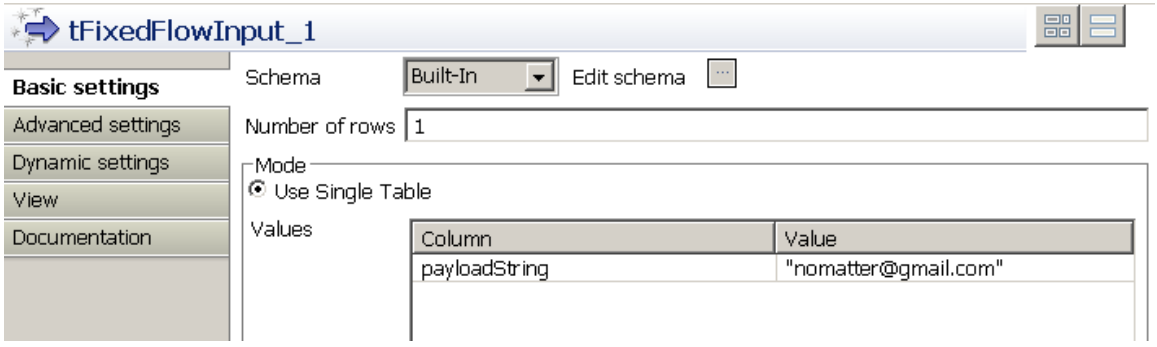
Dropping and linking the components

1. Drop the following components from the **Palette** onto the design workspace: a **tFixedFlowInput**, a **tXMLMap**, a **tESBConsumer**, and two **tLogRow** components.
2. Right-click the **tFixedFlowInput** component, select **Row** > **Main** from the contextual menu and click the **tXMLMap** component.
3. Right-click the **tXMLMap** component, select **Row** > ***New Output* (Main)** from the contextual menu and click the **tESBConsumer** component. Enter *payload* in the popup dialog box to name this row and accept the propagation that prompts you to get the schema from the **tESBConsumer** component.
4. Right-click the **tESBConsumer** component, select **Row** > **Response** from the contextual menu and click one of the **tLogRow** component.
5. Right-click the **tESBConsumer** component again, select **Row** > **Fault** from the contextual menu and click the other **tLogRow** component.

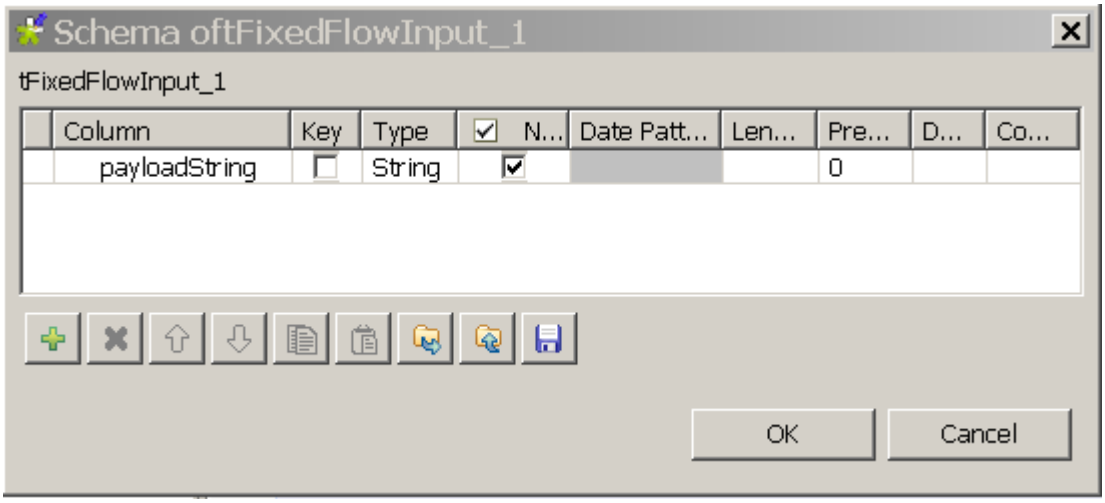
Configuring the components

Configuring the tFixedFlowInput component

1. Double-click the **tFixedFlowInput** component to open its **Basic settings** view in the **Component** tab.



2. Click the three-dot button next to **Edit Schema**. In the schema dialog box, click the plus button to add a new line of **String** type and name it *payloadString*. Click **OK** to close the dialog box.



3. In the **Number of rows** field, set the number of rows as *1*.

4. In the **Mode** area, select **Use Single Table** and input the following request in double quotation marks into the **Value** field:

   ```
   nomatter@gmail.com
   ```

Configuring the tXMLMap component

1. In the design workspace, double-click the **tXMLMap** component to open the **Map Editor**.

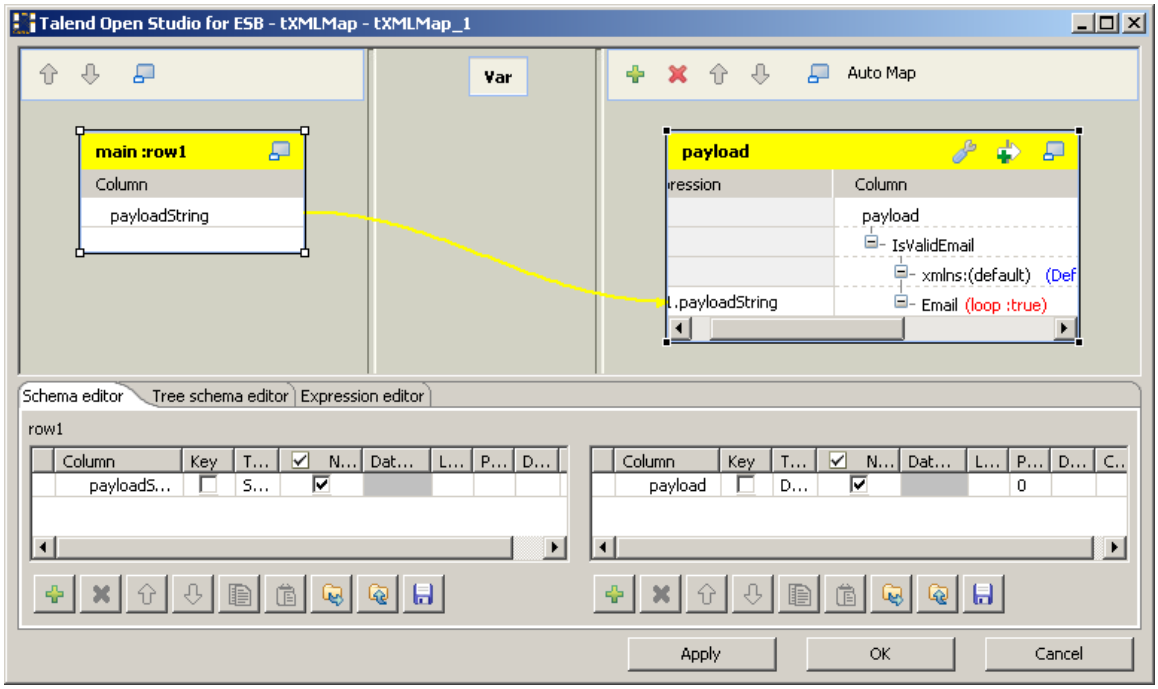
2. In the output table, right-click the root node and select **Rename** from the contextual menu. Enter *IsValidEmail* in the dialog box that appears.

3. Right-click the *IsValidEmail* node and select **Set A Namespace** from the contextual menu. Enter *http://www.webservicex.net* in the dialog box that appears.

4. Right-click the *IsValidEmail* node again and select **Create Sub-Element** from the contextual menu. Enter *Email* in the dialog box that appears.

5. Right-click the *Email* node and select **As loop element** from the contextual menu.

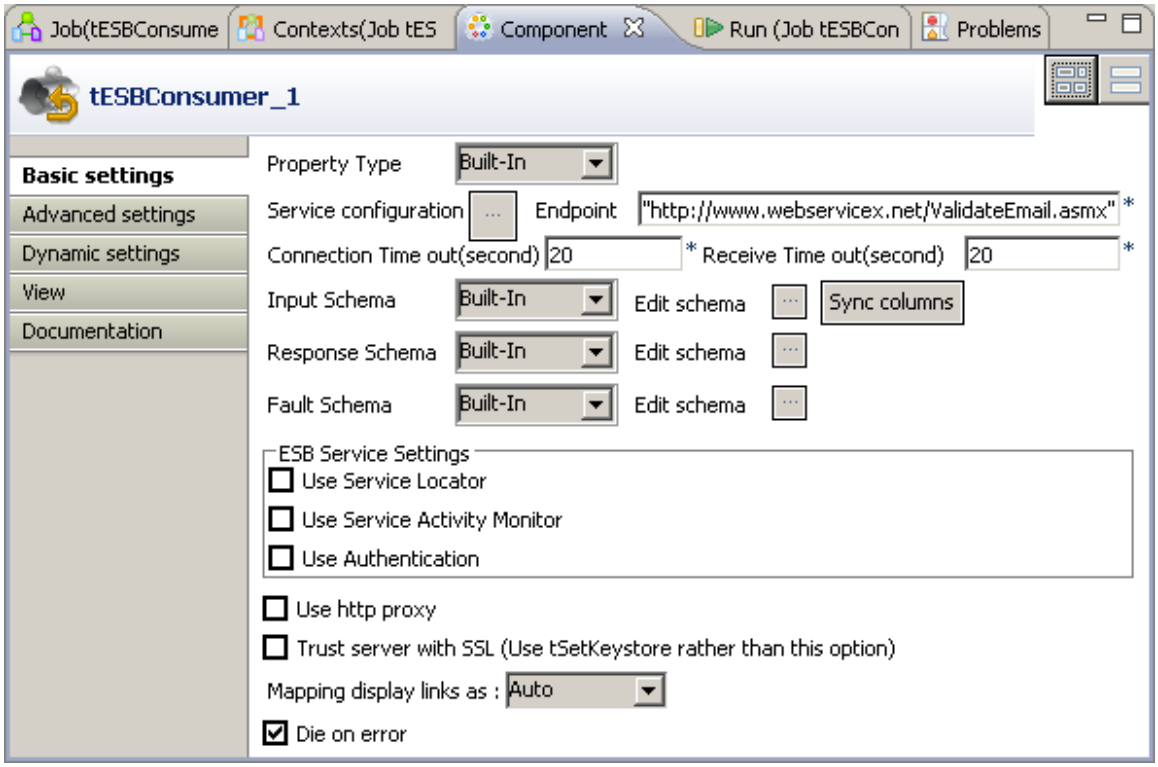
6. Click the *payloadString* node in the input table and drop it to the **Expression** column in the row of the *Email* node in the output table.



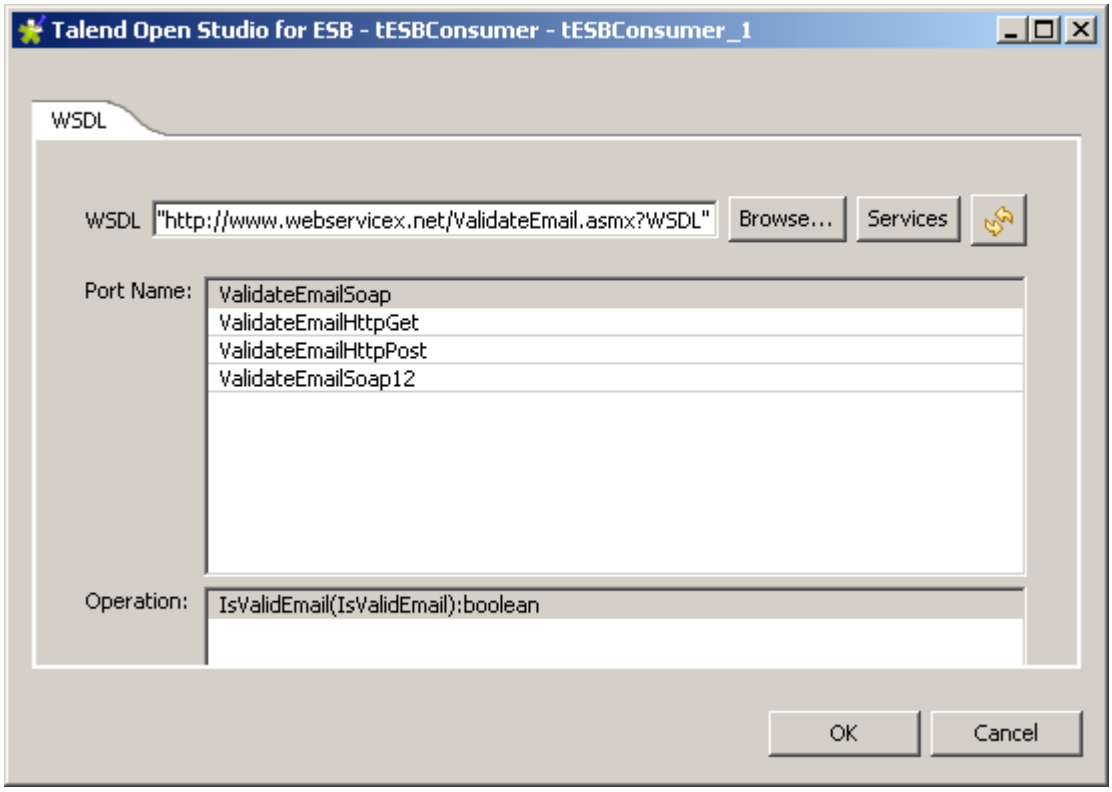
7. Click **OK** to validate the mapping and close the **Map Editor**.

Configuring the tESBConsumer component

1. In the design workspace, double-click the **tESBConsumer** component to open its **Basic settings** view in the **Component** tab.

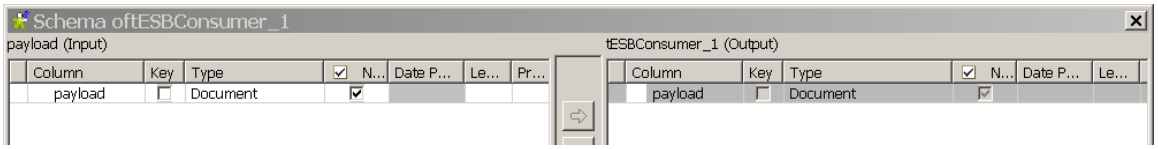


2. Click the three-dot button next to **Service configuration**.

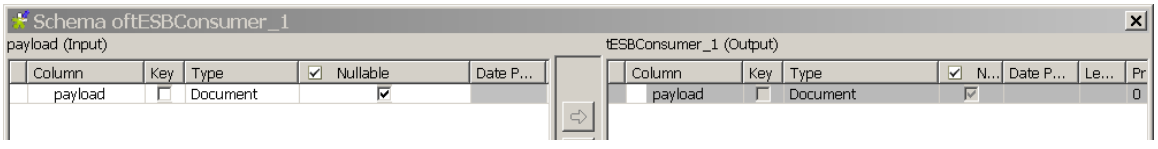


3. In the dialog box that appears, type in: *http://www.webservicex.net/ValidateEmail.asmx?WSDL* in the **WSDL** field and click the refresh button to retrieve port name and operation name. In the **Port Name** list, select the port you want to use, *ValidateEmailSoap* in this example. Click **OK** to validate your settings and close the dialog box.

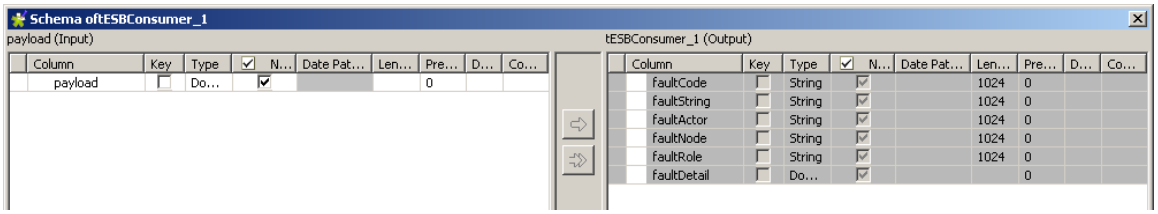
4. Set the **Input Schema** as follows:



5. Set the **Response Schema** as follows:

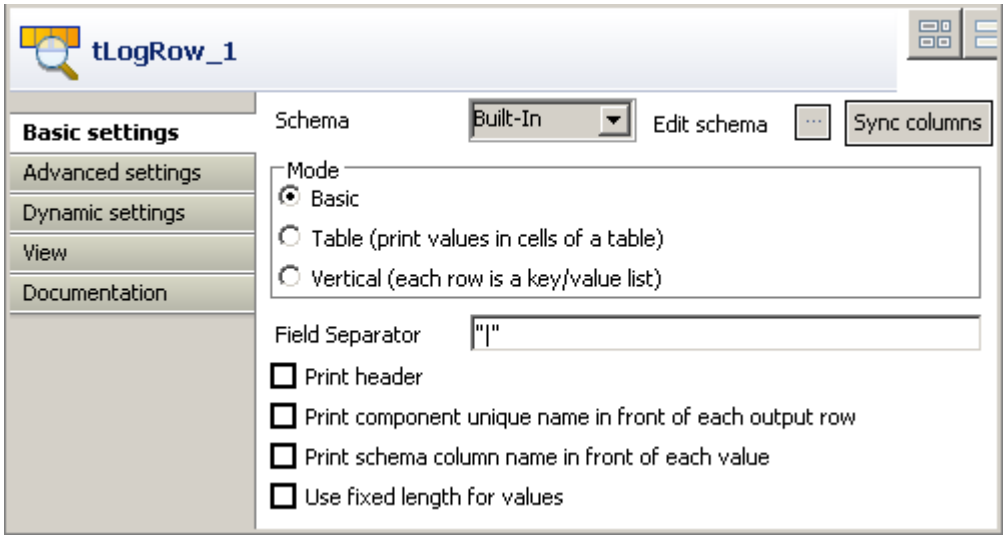


6. Set the **Fault Schema** as follows:

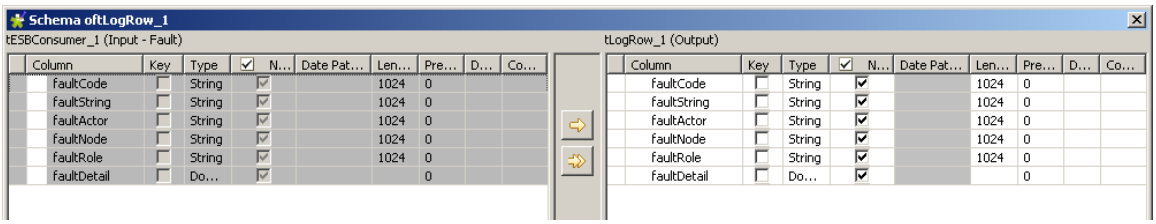


Configuring the tLogRow components

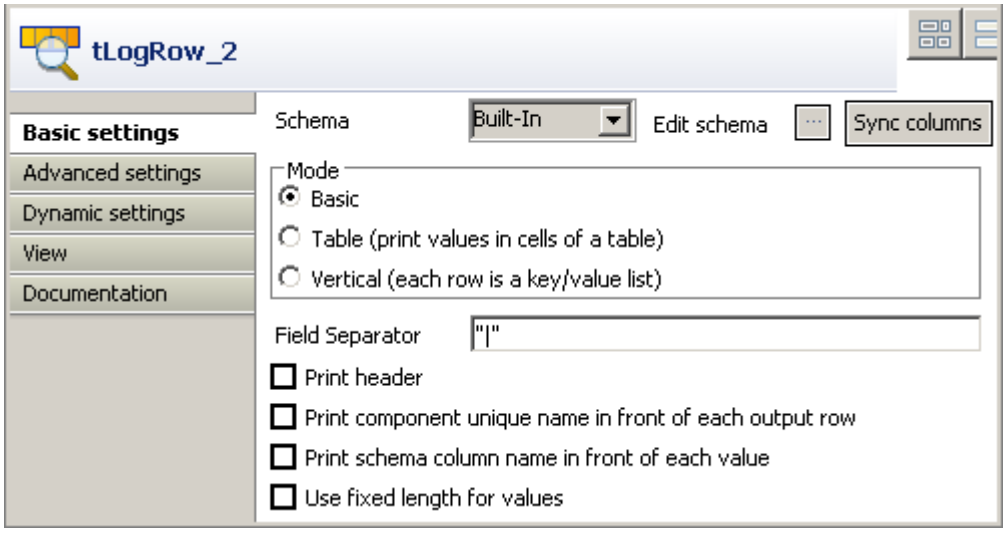
1. In the design workspace, double-click the **tLogRow** component that monitors the fault message to display its **Basic settings** view in the **Component** tab.



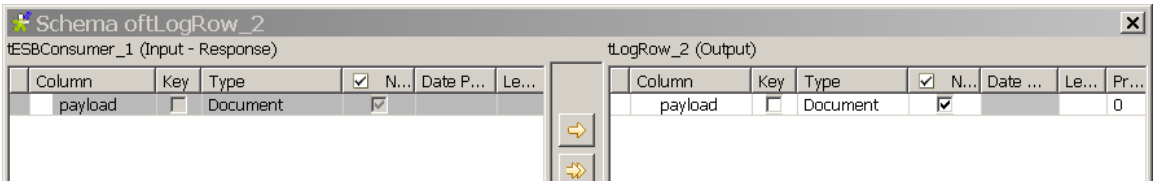
2. Click the three-dot button next to **Edit Schema** and define the schema as follows:



3. In the design workspace, double-click the **tLogRow** component that monitors the response message to display its **Basic settings** view in the **Component** tab.



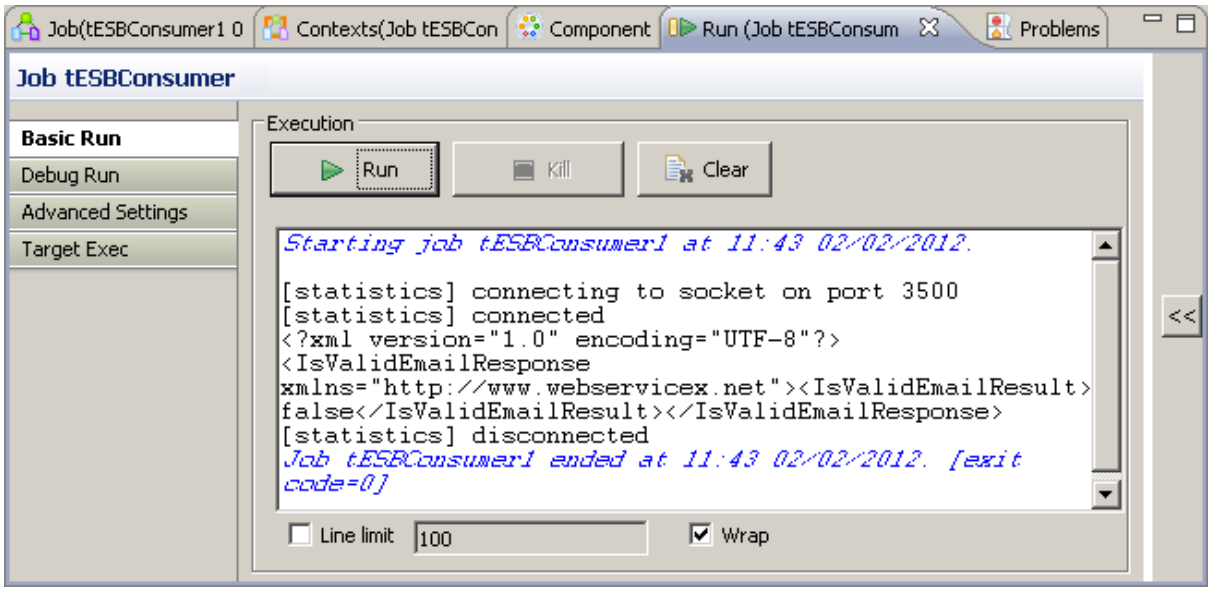
4. Click the three-dot button next to **Edit Schema** and define the schema as follows:



5. Press **Ctrl+S** to save your Job.

Executing the Job

Click the **Run** view to display it and click the **Run** button to launch the execution of your Job. You can also press **F6** to execute it. In the execution log you will see:



tRESTClient

tRESTClient properties

Component family	ESB/REST	
Function	The **tRESTClient** component sends HTTP and HTTPS requests to a REpresentational State Transfer (REST) Web service provider and gets the corresponding responses. This component integrates well with **Talend** Runtime to get HTTPS support, with more QoS features to be supported in time.	
Purpose	The **tRESTClient** component is used to interact with RESTful Web service providers by sending HTTP and HTTPS requests using CXF (JAX-RS).	
Basic settings	*URL*	Type in the URL address of the REST server to be invoked.
	HTTP Method	From this list, select an HTTP method that describes the desired action. The specific meanings of the HTTP methods are subject to definitions of your Web service provider. Listed below are the generally accepted HTTP method definitions: - **GET**: retrieves data from the server end based on the given parameters. - **POST**: uploads data to the server end based on the given parameters. - **PUT**: updates data based on the given parameters, or if the data does not exist, creates it. - **DELETE**: removes data based on the given parameters.
	Content Type	Select **XML**, **JSON**, or **FORM** according to the media type of the content to be uploaded to the server end. This list appears only when you select the **POST** or **PUT** HTTP method.
	Accept Type	Select the media type the client end is prepared to accept for the response from the server end. Available options are **XML**, **JSON**, and **ANY**. When **ANY** is selected, the response message can be of any type and will be transformed into a string.
	Query parameters	Specify the URI query parameters in the form of name-value pairs. This option is mostly used with the GET method.
	Use Authentication	Select this check box if authentication is required on the REST server end. Once selected, you need to provide your username and password. Currently only basic HTTP authentication is supported and more options will be provided.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. This component uses three built-in, read-only schemas. Click **Edit Schema** to view the schema structure.

		Changing the schema type may result in loss of the schema structure and therefore failure of the component.
	Input Schema	Schema for the input data. This schema contains two columns: - **body**: stores the content of structured input data - **string**: stores the input content when it is, or is handled as, a string.
	Response Schema	Schema for server response. This schema is passed onto the next component via a **Row** > **Response** link, and it contains three columns: - **statusCode**: stores the HTTP status code from the server end. - **body**: stores the content of a structured response from the server end. - **string**: stores the response content from the server end when it is, or is handled as, a string.
	Error Schema	Schema for error information. This schema is passed onto the next component via a **Row** > **Error** link, and it contains two columns: - **errorCode**: stores the HTTP status code from the server end when an error occurs during the invocation process. The specific meanings of the errors codes are subject to definitions of your Web service provider. For reference information, visit en.wikipedia.org/wiki/List_of_HTTP_status_codes. - **errorMessage**: stores the error message corresponding the error code.
	Die on error	This check box is selected to kill the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Connection timeout*	Set the amount of time, in seconds, that the client will attempt to establish a connection before it times out. If set to *0*, the client will continue to attempt to open a connection indefinitely. (default: 30)
	Receive timeout	Set the amount of time, in seconds, that the client will wait for a response before it times out. If set to *0*, the client will wait indefinitely. (default: 60)
	HTTP Headers	Type in the name-value pair(s) for HTTP headers to define the parameters of the requested HTTP operation. For the specific definitions of HTTP headers, consult your REST Web service provider. For reference information, visit en.wikipedia.org/wiki/List_of_HTTP_headers.
	Use HTTP proxy	Select this check box if you are using a proxy server. Once selected, you need to provide the connection details: host, port, username and password.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is used as a RESTful Web service client to communicate with a RESTful service provider, with the ability to input a request to a service into a Job and return the Job result as a service response. Depending on the actions to perform, it usually works as a start or middle component in a Job or subjob.	
Connections	Outgoing links: **Row**: Response; Error. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links:	

	Row: Main; Reject. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a

Scenario: Sending and retrieving data by interacting with a RESTful service

This scenario describes a Job composed of two subjobs: a four-component subjob that adds a customer record to a RESTful Web service, and a two-component subjob that retrieves the customer information from the server end. When executed, the Job displays relevant information on the **Run** console.

In this scenario, the Web service used as an example has the following XML tree structure, where only the content of *name* nodes is user updatable:

```
<Customers>
    <Customer>
        <id>id1</id>
        <name>Name1</name>
    </Customer>
    <Customer>
        <id>id2</id>
        <name>Name2</name>
    </Customer>
    ...
</Customers>
```

In the first subjob, a **tXMLMap** component is used to adapt the input data structure to the tree structure of the RESTful Web service.

Setting up the Job

1. Drop the following components from the **Palette** onto the design workspace: a **tFixedFlowInput**, a **tXMLMap**, two **tRESTClient** components and two **tLogRow** components.

2. Connect the **tFixedFlowInput** to the **tXMLMap** component using a **Row** > **Main** connection.

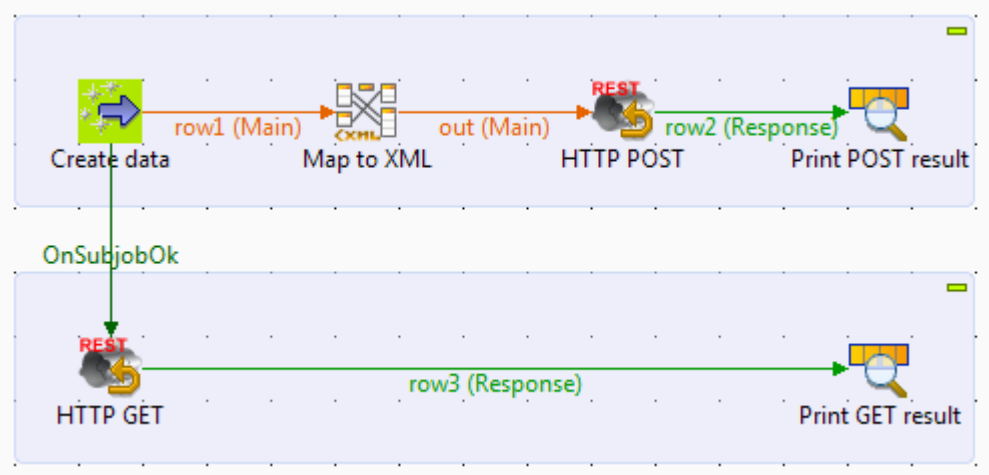
3. Connect the **tXMLMap** component to the first **tRESTClient** component using a **Row** > **Main** connection, and give it a name, *out* in this example.

4. Connect the first **tRESTClient** to first **tLogRow** using a **Row** > **Response** connection to complete the first subjob, which will upload a customer record to the server end and display the response information from the server end.

5. Connect the second **tRESTClient** to the second **tLogRow** using a **Row** > **Response** connection to complete the second subjob, which will retrieve and display customer information from the server end.

6. Connect the two subjobs using a **Trigger** > **OnSubjobOK** connection.

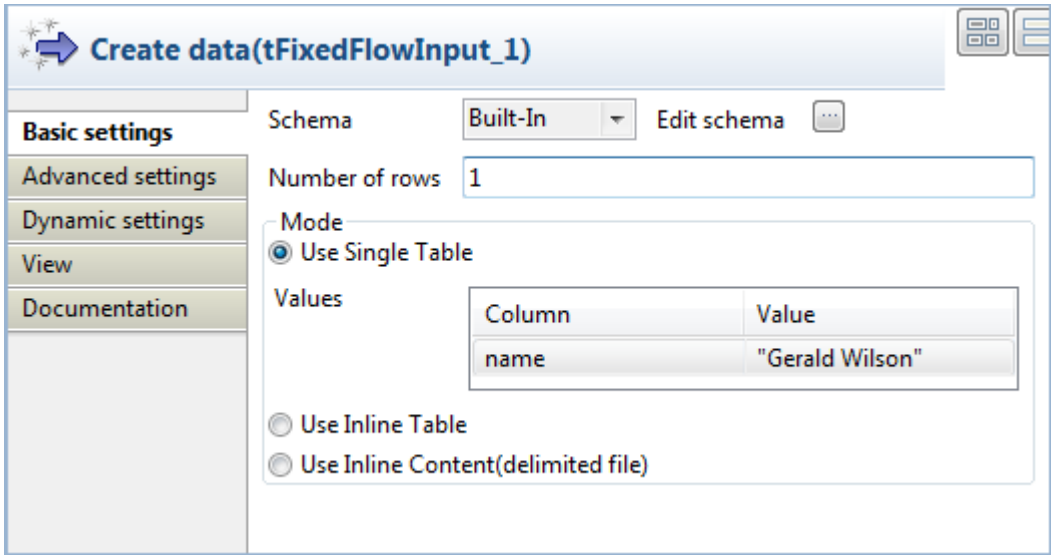
7. Label the components to best describe the actions to perform.



Configuring the components

Setting up data input and structure mapping

1. Double-click the **tFixedFlowInput** component to open its **Basic settings** view.



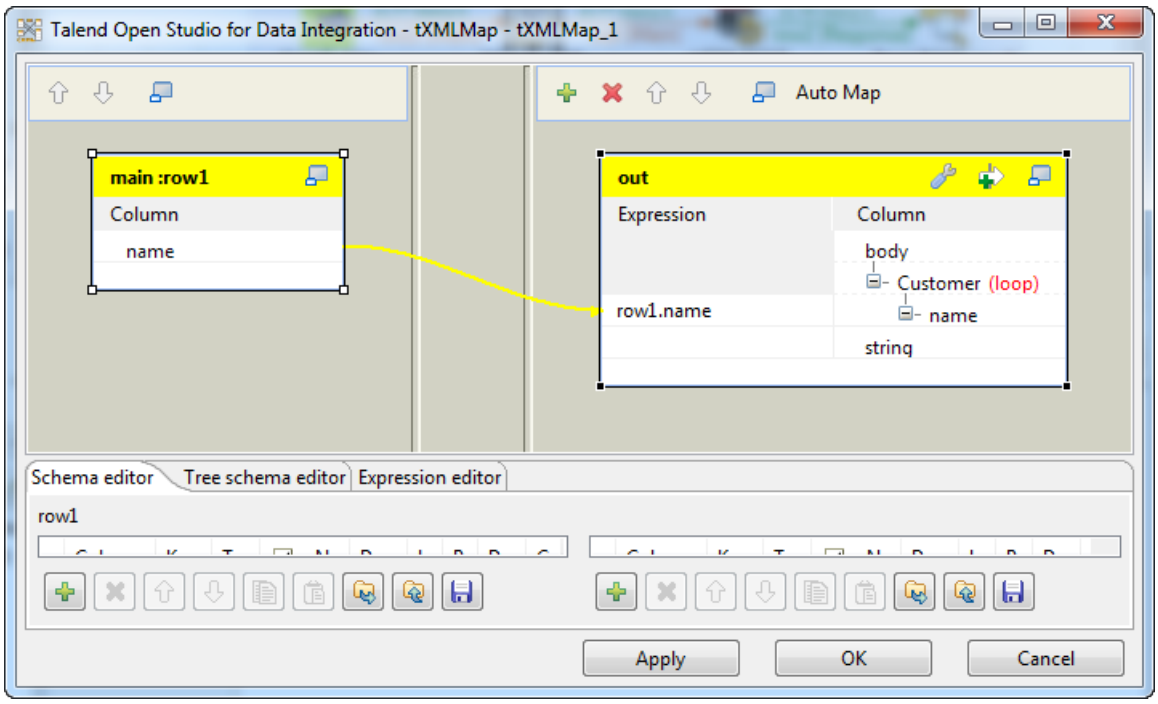
2. Click the **[...]** button next to **Edit schema** and then set up the schema of the input data in the **[Schema]** dialog box, and click **OK** to close the **[Schema]** dialog box.

   In this example, the input schema has only one column, *name*.

3. In the **Values** table under the **Use Single Table** option, fill the **Value** field with a customer name, *Gerald Wilson* for example, between double quotation marks.

4. Fill the **Number of rows** field with *1*.

5. Double-click the **tXMLMap** component to open the Map Editor.



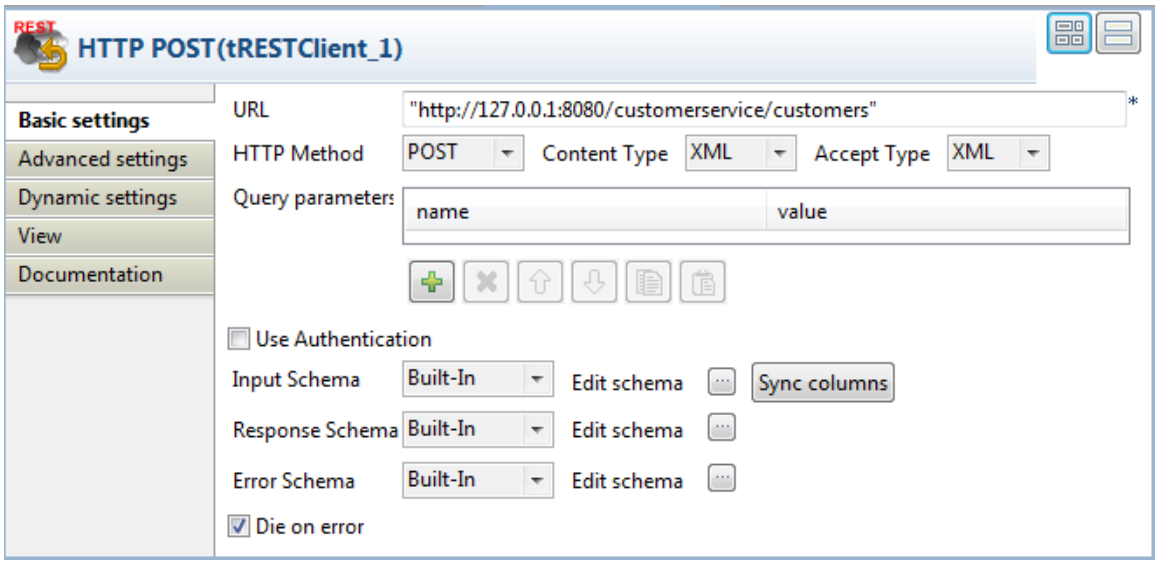
6. In the output table, right-click the default *root* node of the *body* column, select **Rename** from the contextual menu, and rename it to *Customer*. Make sure *Customer* is the loop element because the XML structure of the Web service to be invoked is looped on this element.

7. Right-click the *Customer* node, select **Create Sub-Element** from the contextual menu, and create sub-element named *name*.

8. Drop the *name* column in the input table to the *name* node in the output table, and then click **OK** to validate the mapping and close the Map Editor.

Setting up REST actions and console display

1. Double-click the first **tRESTClient** component to open its **Basic settings** view.



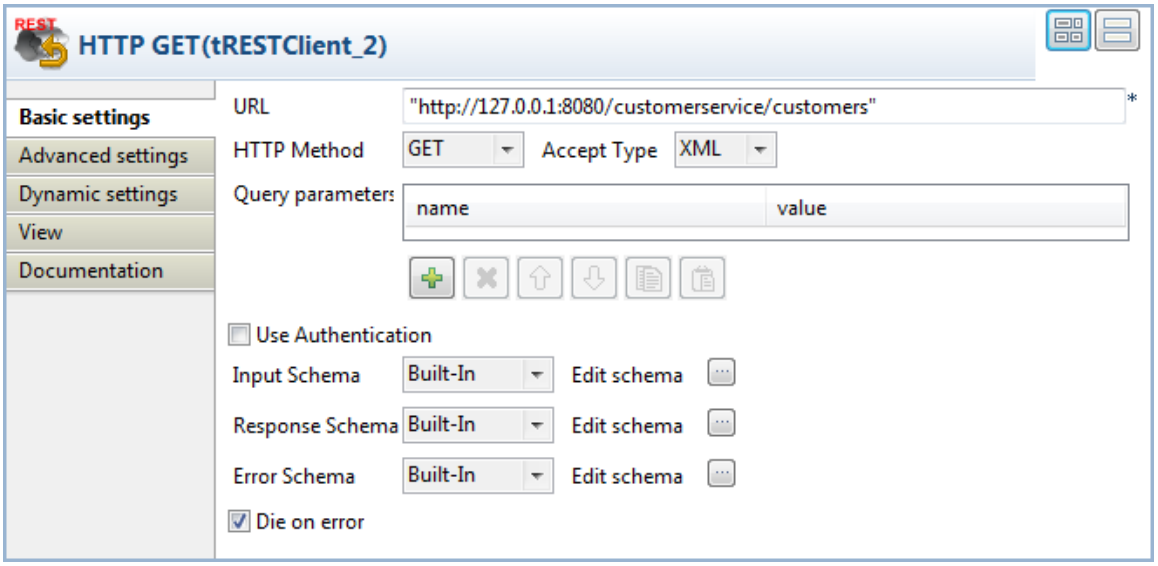
2. Fill the **URL** field with the URL of the Web service you are going to invoke. Note that the URL provided in this use case is for demonstration purpose only and is not a live address.

3. From the **HTTP Method** list, select **POST** to send an HTTP request for creating a new record.

4. From the **Content Type** list, select the type of the content to be uploaded to the server end, **XML** in this example.

5. From the **Accept Type** list, select the type the client end is prepared to accept for the response from the server end, **XML** in this example. Leave the rest of the settings as they are.

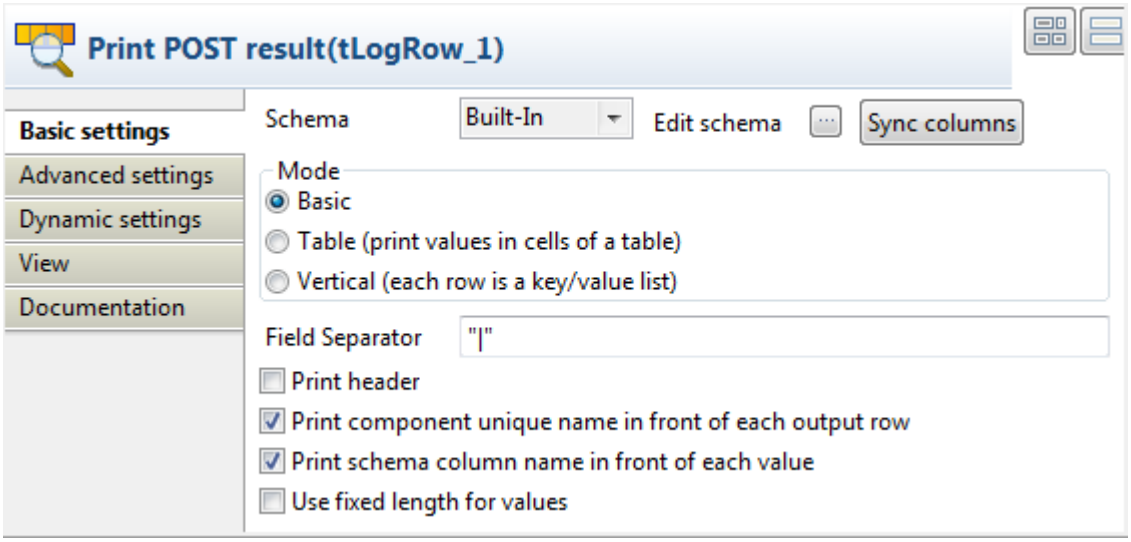
6. Double-click the second **tRESTClient** component to open its **Basic settings** view.



7. Fill the **URL** field with the same URL as in the first **tRESTClient** component.

8. From the **HTTP Method** list, select **GET** to send an HTTP request for retrieving the existing records, and select **XML** from the **Accept Type** list. Leave the rest of the settings as they are.

9. In the **Basic settings** view of each **tLogRow**, select the **Print component unique name in front of each output row** and **Print schema column name in front of each value** check boxes for better identification of the output flows.

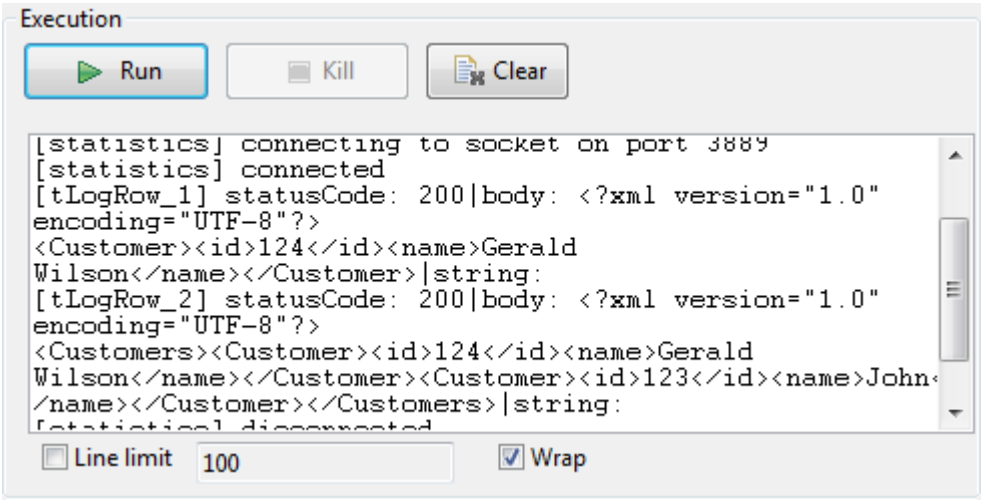


Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** console to launch the Job.

The console shows that the first **tRESTClient** component sends an HTTP request to the server end to create a new customer named *Gerald Wilson*, and the second **tRESTClient** component successfully reads data from the server end, which includes the information of the new customer you just created.



tRouteFault

tRouteFault properties

This component is available to ESB Studio users only.

Component family	ESB/Route	
Function	**tRouteFault** sends messages from a Job to a Route and mark the message as fault.	
Purpose	**tRouteFault** allows you to send messages from a Job to a Route and mark the message as fault.	
Basic settings	*Output Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: No property data stored centrally.
	Camel	Enter the message parameters corresponding to the columns you defined in the schema dialog box via the **Edit schema** button.
	Type	Select between: **Body**: Used to set the body of a Route, equals the code `exchange.getOut().setBody(…)`. **Property**: Used to set the property of a Route, equals the code `exchange.setProperty(name, value)`. **System**: Used to set the system property, equals the code `System.setProperty(name, value)`. **Header**: Used the set the output header, equals the code `exchange.getOut().setHeader(name, value)`.
	Name	This column is used to set the name for the parameter when its type is either **Property**, **System** or **Header**. This column is ignored when the type is **Body**.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as the end component and thus requires an input component.	
Limitation	The Job using a **tRouteFault** must be launched only within a Route using the **cTalendJob** component.	

Scenario: Getting messages from tRouteFault

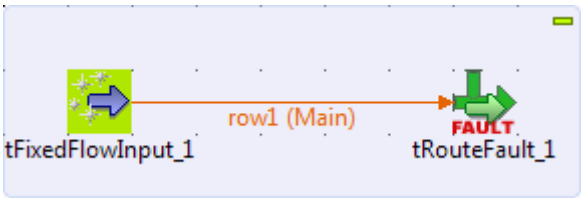
This scenario describes how to send messages from a Job to a Route using the **tRouteFault** component. To do this, we need to build a Job to send the message and then create a Route using the **cTalendJob** component to call the Job and receive the message.

For more information about how to use the Mediation components, see *Reference Guide*.

Creating an Data Integration Job

In this section, we will build a Job named *RouteCommunication* to send the message to a Route.

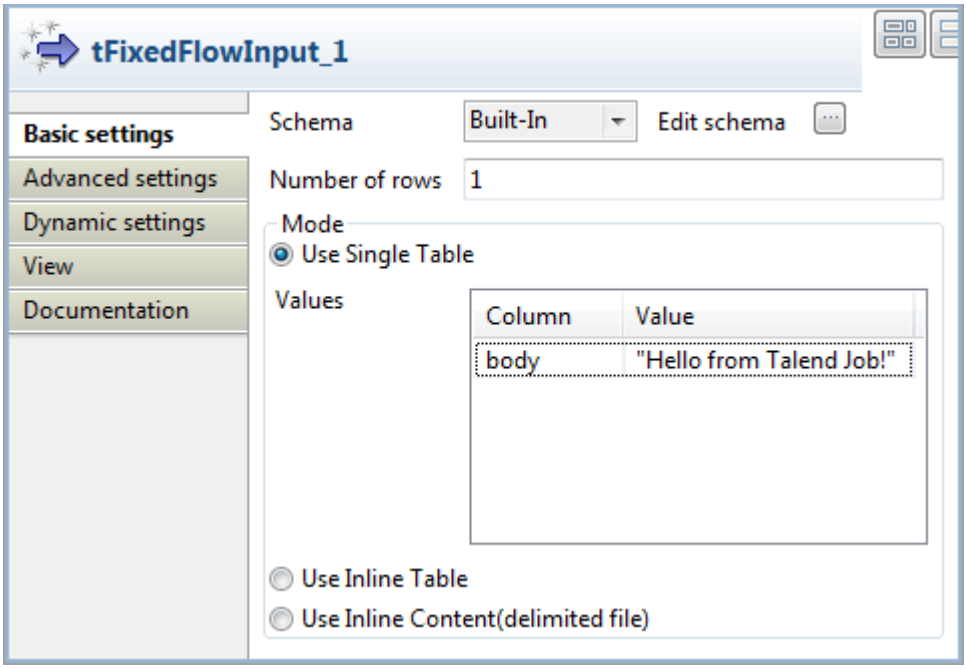
Dropping and linking the components



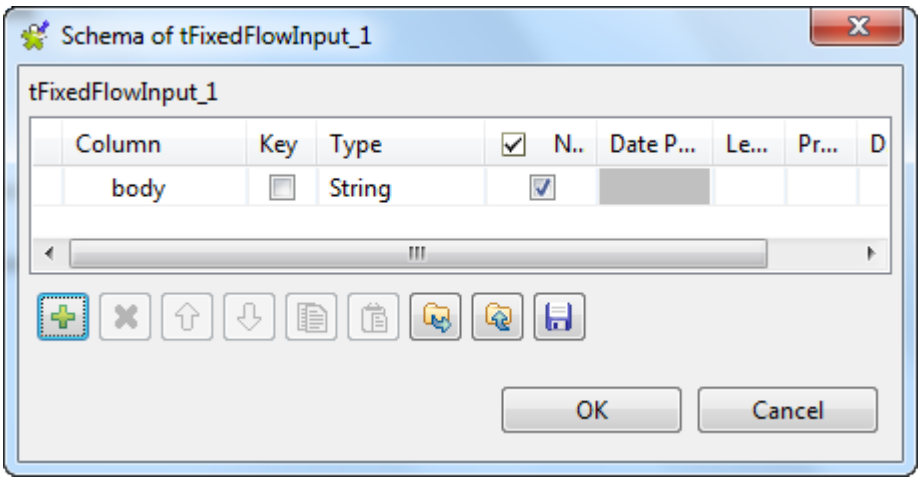
1. Drag and drop a **tFixedFlowInput** and a **tRouteFault** from the **Palette** onto the design workspace.
2. Right-click the **tFixedFlowInput** component, select **Row** > **Main** from the contextual menu and click the **tRouteFault** component.

Configuring the components

1. Double-click the **tFixedFlowInput** component to open its **Basic settings** view in the **Component** tab.



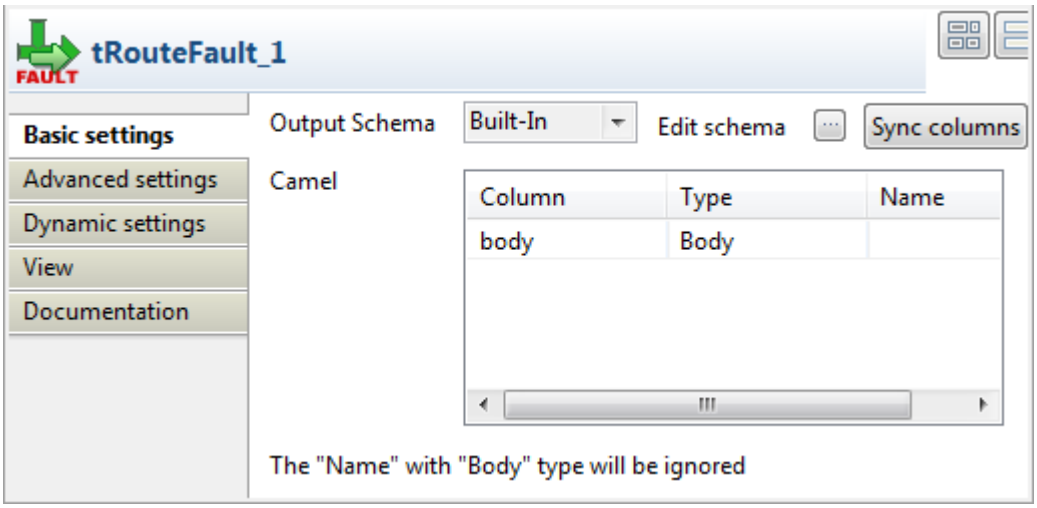
2. Click the three-dot button next to **Edit Schema**. In the schema dialog box, click the plus button to add a new line of **String** type and name it *body*. Click **OK** to close the dialog box.



3. In the **Number of rows** field, set the number of rows as *1*.
4. In the **Mode** area, select **Use Single Table** and input the following request in double quotation marks into the **Value** field:

   ```
   Hello from Talend Job!
   ```

5. Double-click the **tRouteFault** component to display its **Basic settings** view in the **Component** tab.

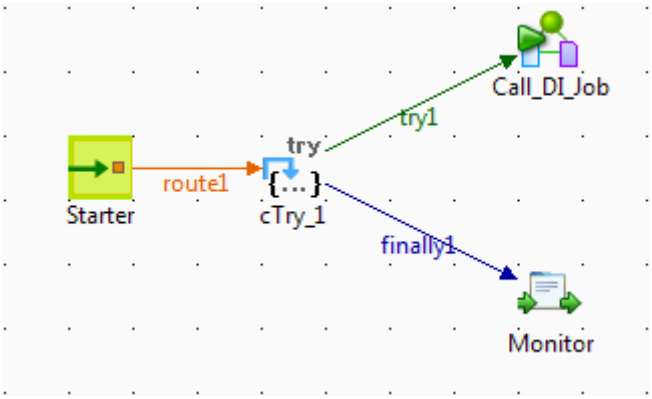


6. Make sure that **tRouteFault** has one element *body*. Set its type to **Body**.
7. Press **Ctrl+S** to save your Job.

Creating a Mediation Route

In this section, we will create a Route to receive the message sent from the Job.

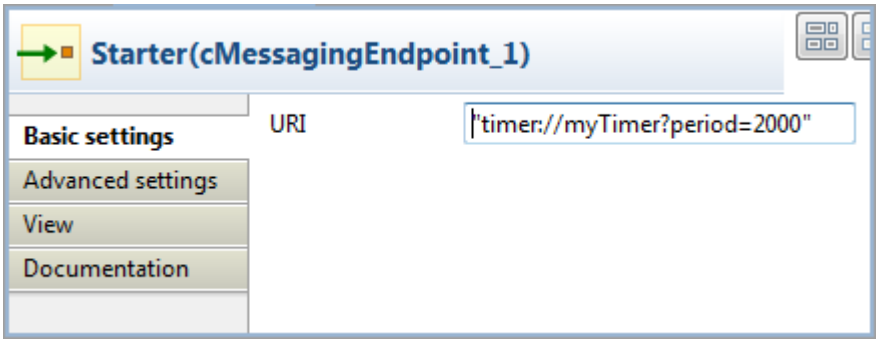
Dropping and linking the components



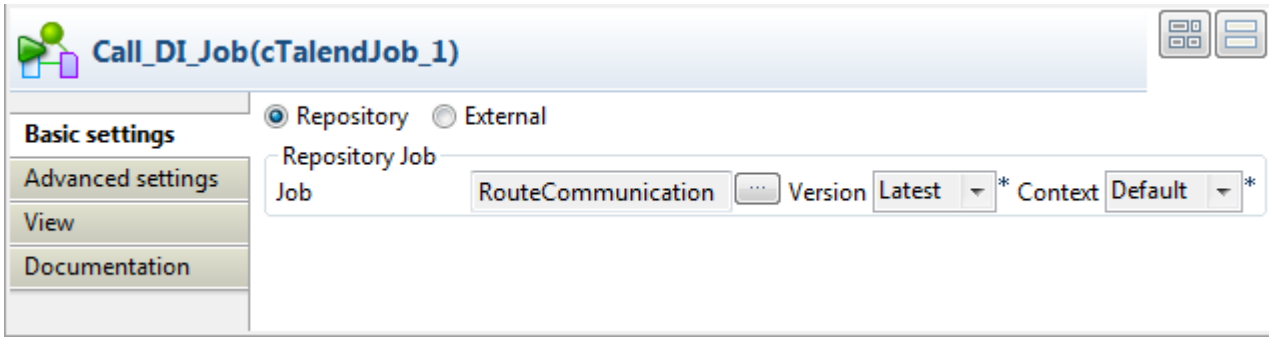
1. Drag and drop a **cMessagingEndpoint**, a **cTry**, a **cTalendJob**, and a **cProcessor** component from the **Palette** onto the design workspace.
2. Link the components with the **Row** > **Route** connection as shown above.
3. Label the components for better identification of their roles.

Configuring the components

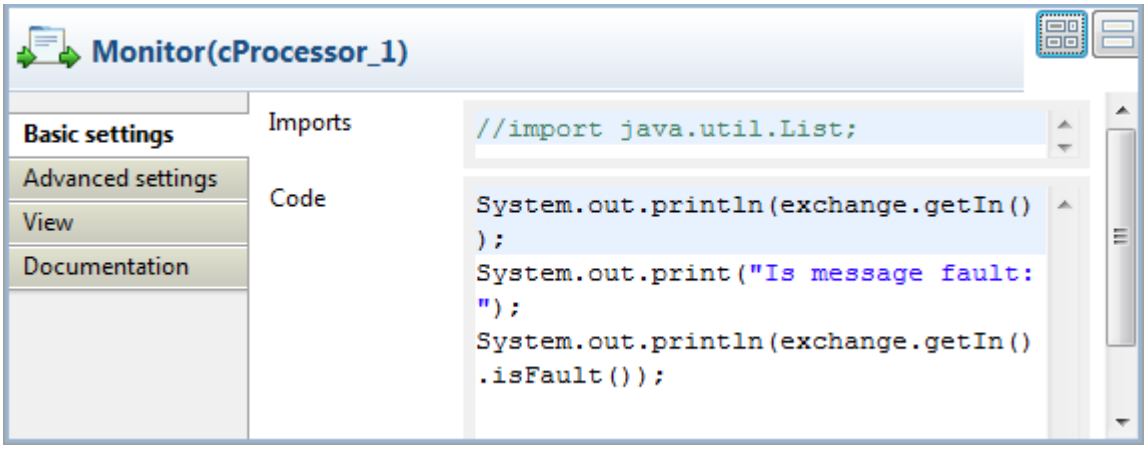
1. Double-click the **cMessagingEndpoint** component to open its **Basic settings** view in the **Component** tab.



2. In the **URI** field, enter `"timer://myTimer?period=2000"` to start message exchanges every 2000 milliseconds.
3. Double-click the **cTalendJob** component to display its **Basic settings** view in the **Component** tab.



4. Select Job *RouteCommunication* from the repository.
5. Double click the **cProcessor** component to display its **Basic settings** view in the **Component** tab.



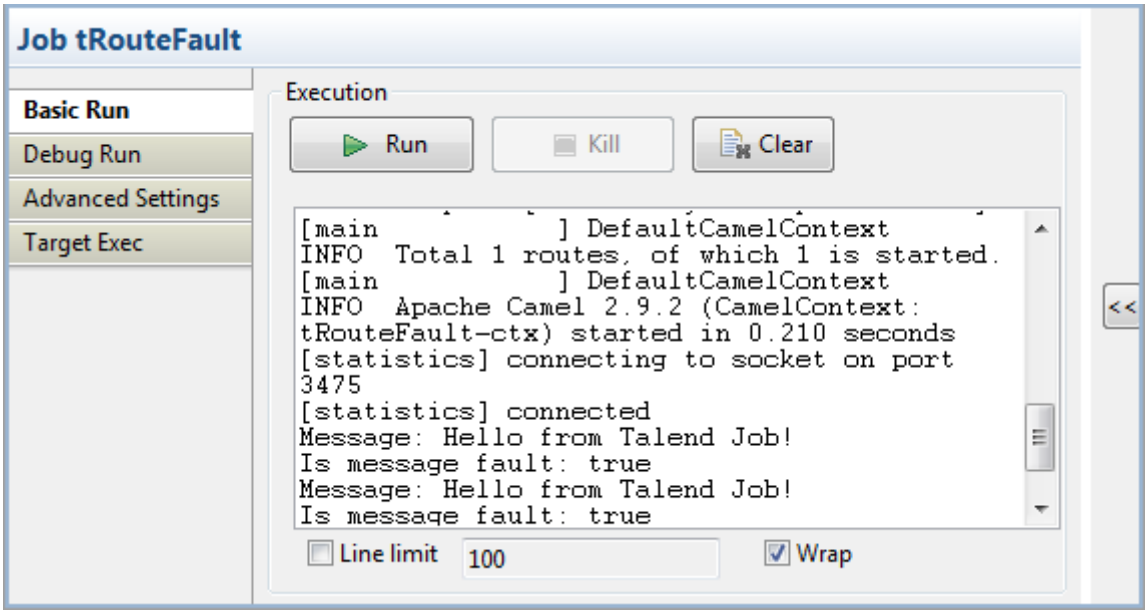
6. Customize the code in the **Code** area to show the message body and check whether the message is marked as fault or not:

```
System.out.println(exchange.getIn());
System.out.print("Is message fault: ");
System.out.println(exchange.getIn().isFault());
```

7. Press **Ctrl+S** to save your Route.

Executing the Route

Click the **Run** view to display it and click the **Run** button to launch the execution of your Route. You can also press **F6** to execute it. In the execution log you will see:



tRouteInput

tRouteInput properties

This component is available to ESB Studio users only.

Component family	ESB/Route	
Function	**tRouteInput** accepts messages from a Route.	
Purpose	**tRouteInput** allows you to accept messages from a Route.	
Basic settings	*Input Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: No property data stored centrally.
	Camel	Enter the message parameters corresponding to the columns you defined in the schema dialog box via the **Edit schema** button using the Simple Expression Language. For more information about the Simple Expression Language, see the site http://camel.apache.org/simple.html.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as starter component and has an output.	
Limitation	The Job using a **tRouteInput** must be launched only within a Route using the **cTalendJob** component.	

Scenario: Getting messages from a Route

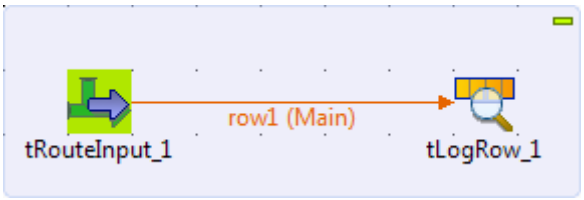
This scenario describes how to accept messages from a Route using the **tRouteInput** component. To do this, we need to build a Job to accept the message and a Route to send the message and call the Job using the **cTalendJob** component.

For more information about how to use the Mediation components, see *Reference Guide*.

Creating an Data Integration Job

In this section, we will build a Job named *RouteCommunication* to accept the message from a Route.

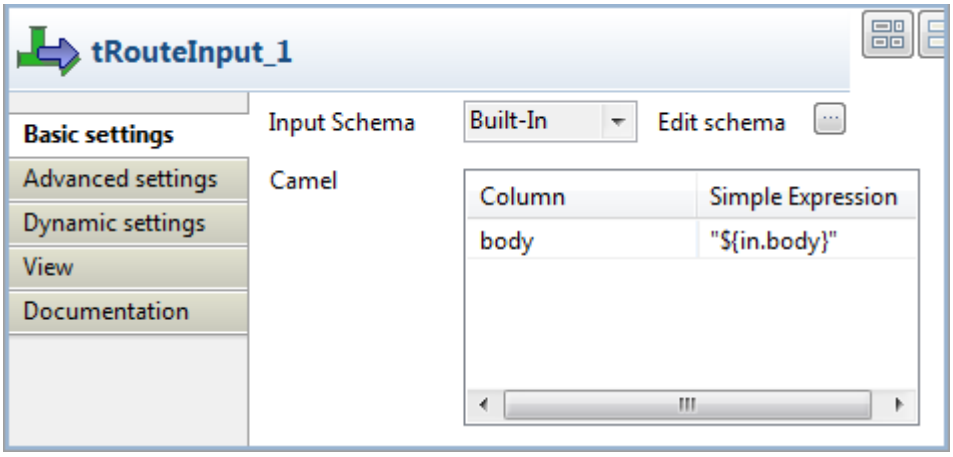
Dropping and linking the components



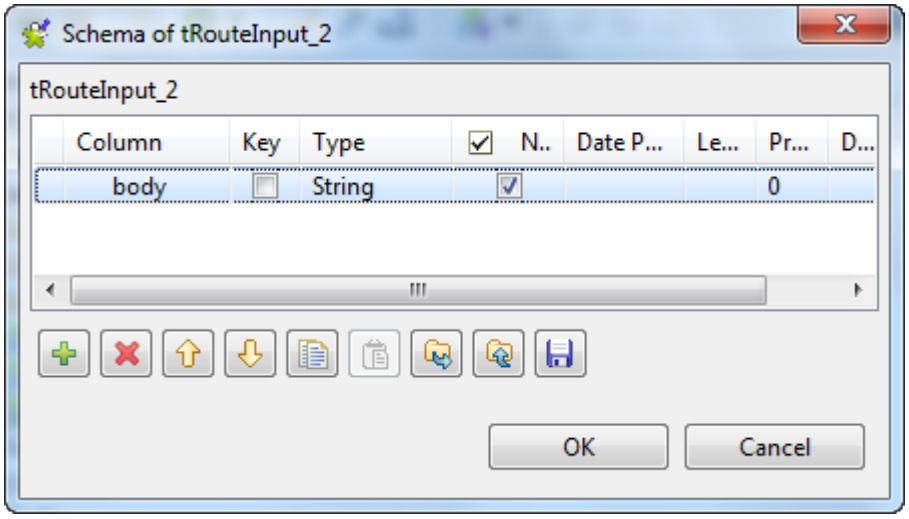
1. Drag and drop a **tRouteInput** and a **tLogRow**from the **Palette** onto the design workspace.
2. Right-click the **tRouteInput** component, select **Row** > **Main** from the contextual menu and click the **tLogRow** component.

Configuring the components

1. Double-click the **tRouteInput** component to open its **Basic settings** view in the **Component** tab.



2. Click the three-dot button next to **Edit Schema**. In the schema dialog box, click the plus button to add a new line of **String** type and name it *body*. Click **OK** to close the dialog box.



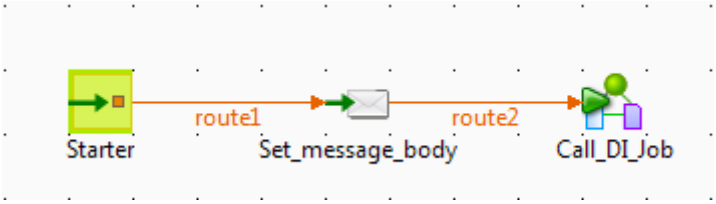
3. In the **Simple Expression** field for the *body* element, enter `"${in.body}"` to get the body of the input message from the Route.
4. The **tLogRow** component will monitor the message exchanges and does not need any configuration.

5. Press **Ctrl+S** to save your Job.

Creating a Mediation Route

In this section, we will create a Route to send the message to the Job.

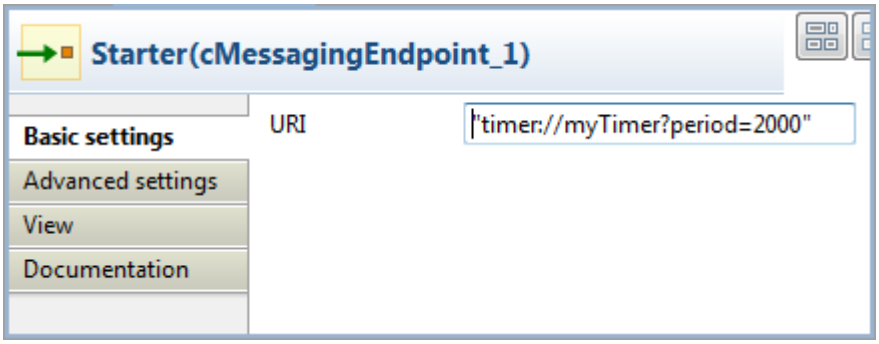
Dropping and linking the components



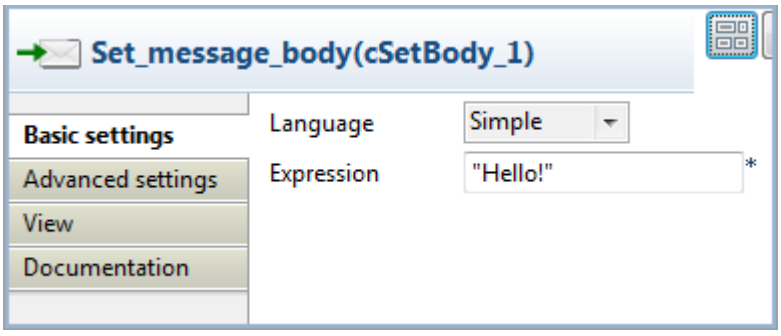
1. Drag and drop a **cMessagingEndpoint**, a **cSetBody**, and a **cTalendJob** from the **Palette** onto the design workspace.
2. Link the components with the **Row** > **Route** connection as shown above.
3. Label the components for better identification of their roles.

Configuring the components

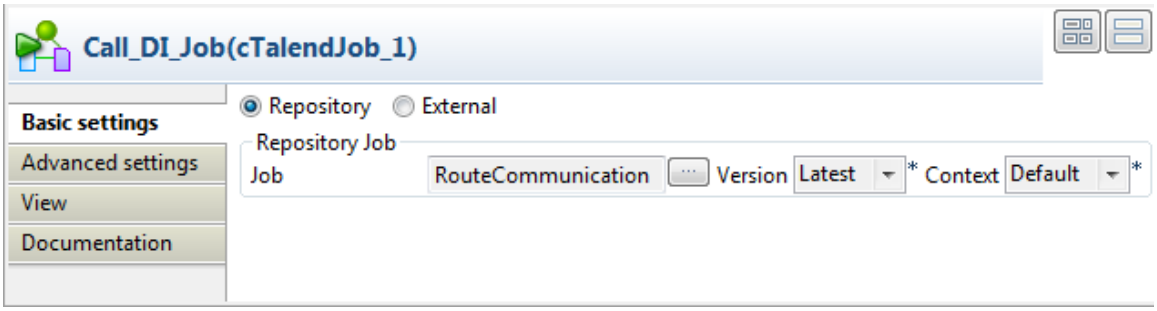
1. Double-click the **cMessagingEndpoint** component to open its **Basic settings** view in the **Component** tab.



2. In the **URI** field, enter `"timer://myTimer?period=2000"` to start message exchanges every 2000 milliseconds.
3. Double click the **cSetBody** component to display its **Basic settings** view in the **Component** tab.



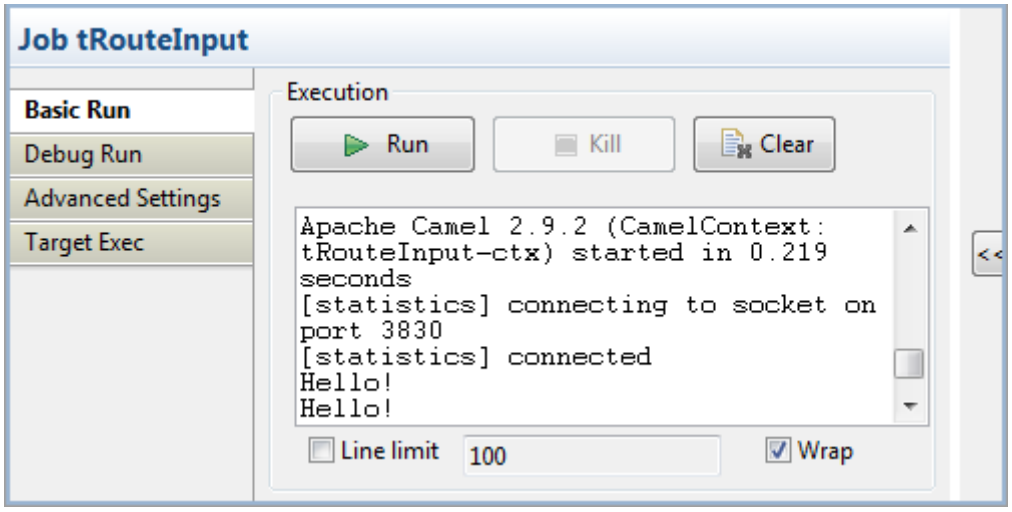
4. Select **Simple** from the **Language** list and enter `Hello!` in the **Expression** field.
5. Double-click the **cTalendJob** component to display its **Basic settings** view in the **Component** tab.



6. Select Job *RouteCommunication* from the repository.
7. Press **Ctrl+S** to save your Route.

Executing the Route

Click the **Run** view to display it and click the **Run** button to launch the execution of your Route. You can also press **F6** to execute it. In the execution log you will see:



tRouteOutput

tRouteOutput properties

This component is available to ESB Studio users only.

Component family	ESB/Route	
Function	**tRouteOutput** sends messages from a Job to a Route.	
Purpose	**tRouteOutput** allows you to send messages from a Job to a Route.	
Basic settings	*Output Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: No property data stored centrally.
	Camel	Enter the message parameters corresponding to the columns you defined in the schema dialog box via the **Edit schema** button.
	Type	Select between: **Body**: Used to set the body of a Route, equals the code `exchange.getOut().setBody(…)`. **Property**: Used to set the property of a Route, equals the code `exchange.setProperty(name, value)`. **System**: Used to set the system property, equals the code `System.setProperty(name, value)`. **Header**: Used the set the output header, equals the code `exchange.getOut().setHeader(name, value)`.
	Name	This column is used to set the name for the parameter when its type is either **Property**, **System** or **Header**. This column is ignored when the type is **Body**.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as the end component and thus requires an input component.	
Limitation	The Job using a **tRouteOutput** must be launched only within a Route using the **cTalendJob** component.	

Scenario: Getting messages from tRouteOutput

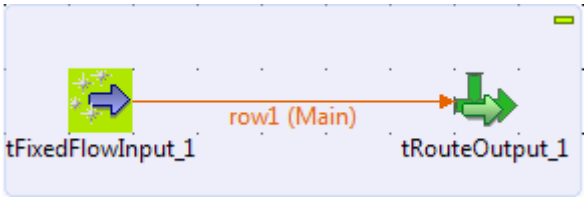
This scenario describes how to send messages from a Job to a Route using the **tRouteOutput** component. To do this, we need to build a Job to send the message and then create a Route using the **cTalendJob** component to call the Job and receive the message.

For more information about how to use the Mediation components, see *Reference Guide*.

Creating an Data Integration Job

In this section, we will build a Job named *RouteCommunication* to send the message to a Route.

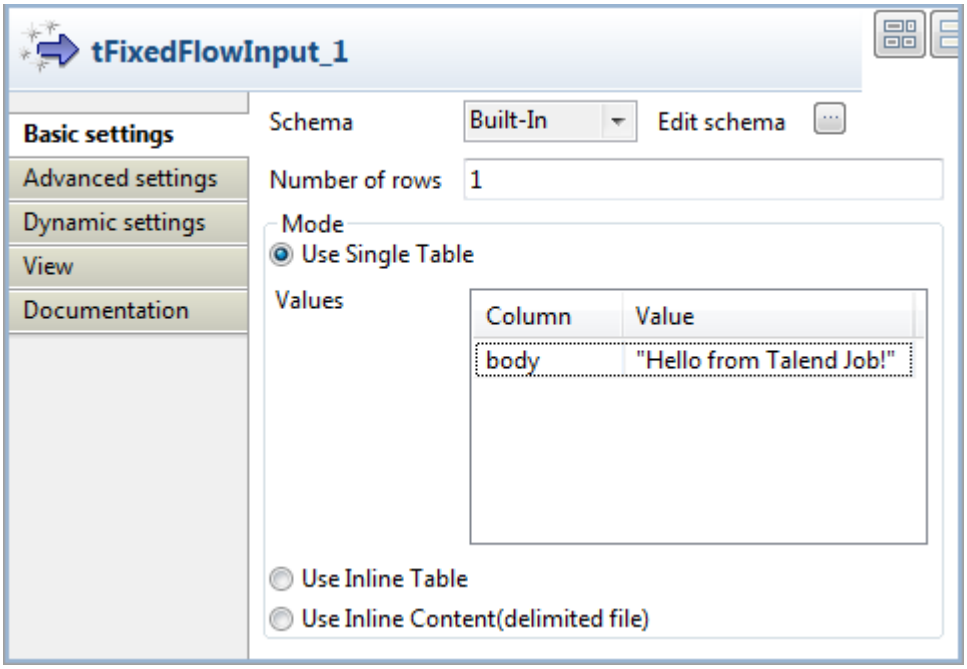
Dropping and linking the components



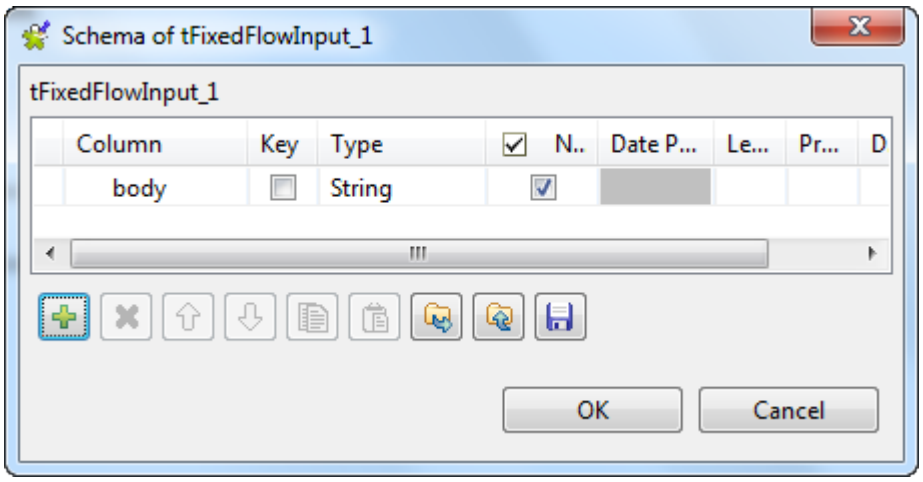
1. Drag and drop a **tFixedFlowInput** and a **tRouteOutput** from the **Palette** onto the design workspace.
2. Right-click the **tFixedFlowInput** component, select **Row** > **Main** from the contextual menu and click the **tRouteOutput** component.

Configuring the components

1. Double-click the **tFixedFlowInput** component to open its **Basic settings** view in the **Component** tab.



2. Click the three-dot button next to **Edit Schema**. In the schema dialog box, click the plus button to add a new line of **String** type and name it *body*. Click **OK** to close the dialog box.

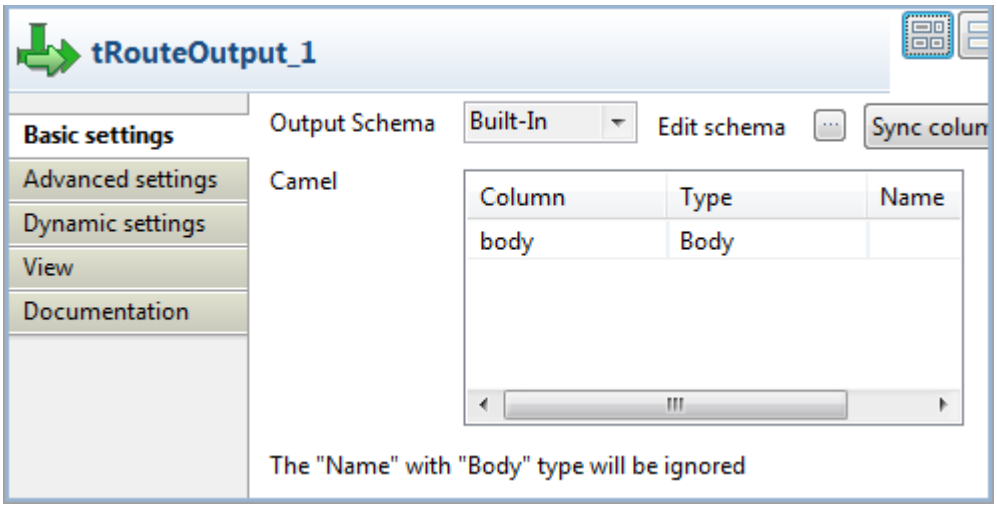


3. In the **Number of rows** field, set the number of rows as *1*.

4. In the **Mode** area, select **Use Single Table** and input the following request in double quotation marks into the **Value** field:

   ```
   Hello from Talend Job!
   ```

5. Double-click the **tRouteOutput** component to display its **Basic settings** view in the **Component** tab.



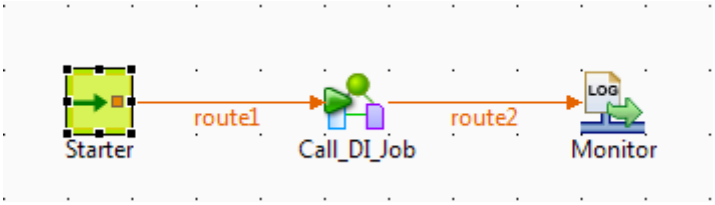
6. Make sure that **tRouteOutput** has one element *body*. Set its type to **Body**.

7. Press **Ctrl+S** to save your Job.

Creating a Mediation Route

In this section, we will create a Route to receive the message sent from the Job.

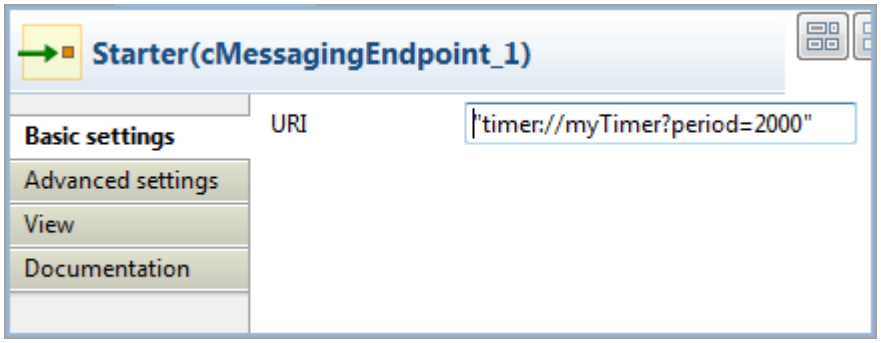
Dropping and linking the components



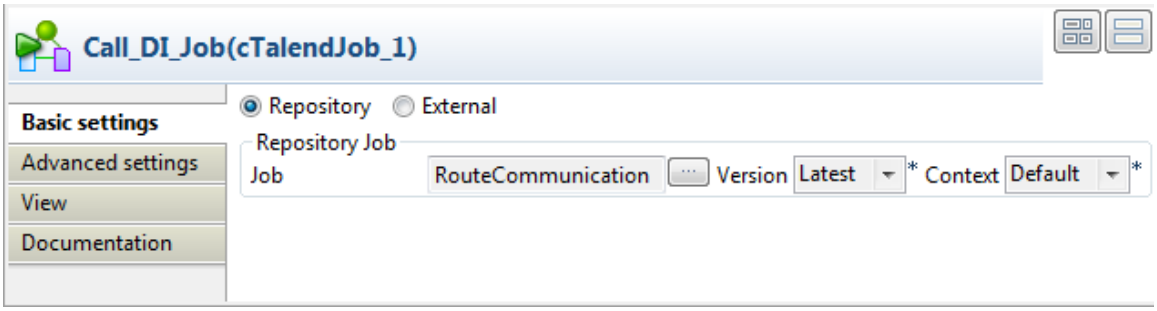
1. Drag and drop a **cMessagingEndpoint**, a **cTalendJob**, and a **cLog** component from the **Palette** onto the design workspace.
2. Link the components with the **Row** > **Route** connection as shown above.
3. Label the components for better identification of their roles.

Configuring the components

1. Double-click the **cMessagingEndpoint** component to open its **Basic settings** view in the **Component** tab.



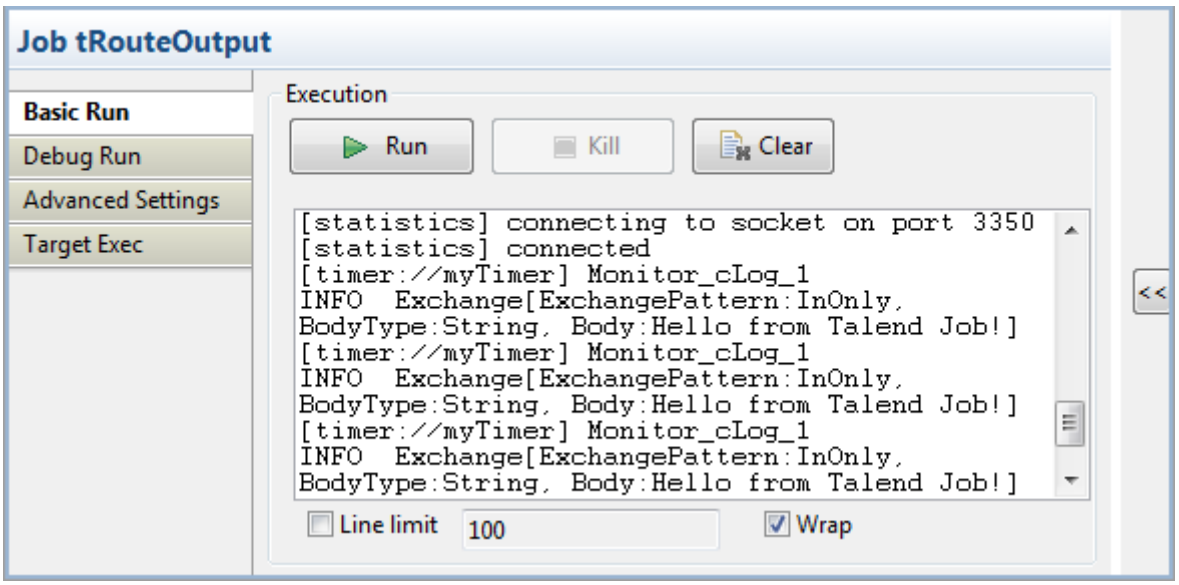
2. In the **URI** field, enter `"timer://myTimer?period=2000"` to start message exchanges every 2000 milliseconds.
3. Double-click the **cTalendJob** component to display its **Basic settings** view in the **Component** tab.



4. Select Job *RouteCommunication* from the repository.
5. The **cLog** component will monitor the message exchanges and does not need any configuration.
6. Press **Ctrl+S** to save your Route.

Executing the Route

Click the **Run** view to display it and click the **Run** button to launch the execution of your Route. You can also press **F6** to execute it. In the execution log you will see:


Job tRouteOutput
Basic Run
Debug Run
Advanced Settings
Target Exec
Execution
Run
Kill
Clear
[statistics] connecting to socket on port 3350
[statistics] connected
[timer://myTimer] Monitor_cLog_1
INFO Exchange[ExchangePattern:InOnly,
BodyType:String, Body:Hello from Talend Job!]
[timer://myTimer] Monitor_cLog_1
INFO Exchange[ExchangePattern:InOnly,
BodyType:String, Body:Hello from Talend Job!]
[timer://myTimer] Monitor_cLog_1
INFO Exchange[ExchangePattern:InOnly,
BodyType:String, Body:Hello from Talend Job!]
Line limit
100
Wrap
<<

File components

This chapter details the main components that you can find in **File** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The File family groups together components that read and write data in all types of files, from the most popular to the most specific format (in the Input and Output subfamilies). In addition, the Management subfamily groups together File-dedicated components that perform various tasks on files, including unarchiving, deleting, copying, comparing files and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAdvancedFileOutputXML

tAdvancedFileOutputXML belongs to two component families: File and XML. For more information on **tAdvancedFileOutputXML,** see section *tAdvancedFileOutputXML*.

tApacheLogInput

tApacheLogInput properties

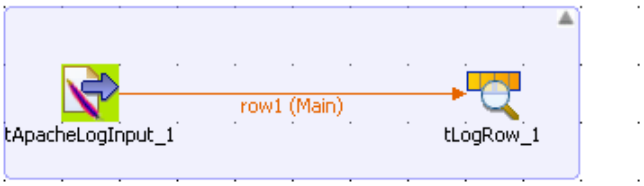
Component family	File/Input	
Function	**tApacheLogInput** reads the access-log file for an Apache HTTP server.	
Purpose	**tApachLogInput** helps to effectively manage the Apache HTTP Server,. It is necessary to get feedback about the activity and performance of the server as well as any problems that may be occurring.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In the context of **tApacheLogInput** usage, the schema is read-only**.**
		Built-in: You can create the schema and store it locally for this component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File Name	Name of the file and/or the variable to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row > Reject** link.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	**tApacheLogInput** can be used with other components or as a standalone component. It allows you to create a data flow using a **Row > Main** connection, or to create a reject flow to filter specified data using a **Row > Reject** connection. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	
Limitation	n/a	

Scenario: Reading an Apache access-log file

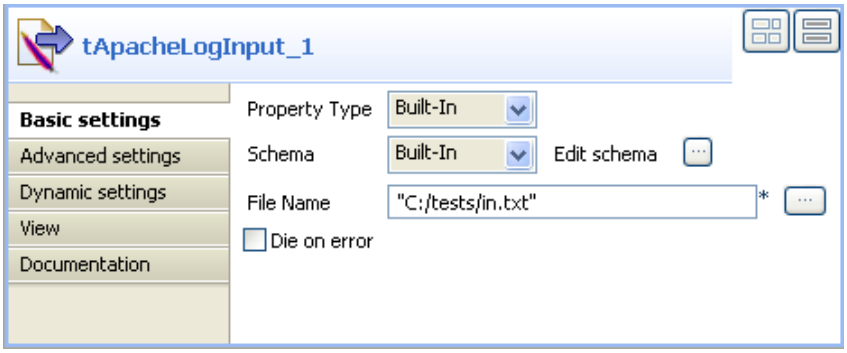
The following scenario creates a two-component Job, which aims at reading the access-log file for an Apache HTTP server and displaying the output in the **Run** log console.

1. Drop a **tApacheLogInput** component and a **tLogRow** component from the **Palette** onto the design workspace.

2. Right-click on the **tApacheLogInput** component and connect it to the **tLogRow** component using a **Main Row** link.



3. In the design workspace, select **tApacheLogInput**.
4. Click the **Component** tab to define the basic settings for **tApacheLogInput**.



5. If desired, click the **Edit schema** button to see the read-only columns.
6. In the **File Name** field, enter the file path or browse to the access-log file you want to read.
7. In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*
8. Press **F6** to execute the Job.

```
Starting job apach at 12:11 01/09/2008.
194.84.172.4|-|-|27/May/2007|03:06:12|+0200|GET|/forum/index.php|HTTP/
1.0|200|19024|http://www.talendforge.org/forum/index.php|Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
88.154.28.65|-|-|27/May/2007|05:44:21|+0200|GET|/bugs/view.php?id=205|
HTTP/1.0|200|19456|http://le-gall.net/pierrick/blog/|Mozilla/4.0
(compatible; MSIE 5.0; Windows 98)
72.9.105.42|-|-|27/May/2007|08:53:01|+0200|GET|/forum/register.php|HT
TP/1.0|200|21669|http://www.talendforge.org/register.php|Mozilla/4.0
(compatible; MSIE 7.0b; Windows NT 6.0)
76.171.68.99|-|-|27/May/2007|19:12:33|+0200|GET|/forum/login.php|HTTP/
1.0|200|17445|http://talendforge.org/login.php|Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]
Job apach ended at 12:11 01/09/2008. [exit code=0]
```

The log lines of the defined file are displayed on the console.

tCreateTemporaryFile

tCreateTemporaryFile properties

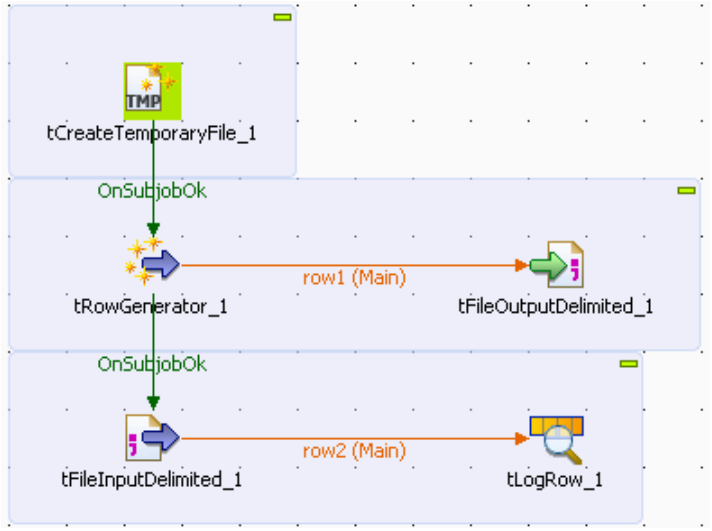
Component family	File/Management	
Function	**tCreateTemporaryFile** creates and manages temporary files.	
Purpose	**tCreateTemporaryFile** helps to create a temporary file and puts it in a defined directory. This component allows you to either keep the temporary file or delete it after Job execution.	
Basic settings	*Remove file when execution is over*	Select this check box to delete the temporary file after Job execution.
	Use default temporary system directory	Select this check box to create the file in the system's default temporary directory.
	Directory	Select this check box to create the temporary file .
	Template	Enter a name to the temporary file respecting the template.
	Suffix	Enter the filename extension to indicate the file format you want to give to the temporary file.
Usage	**tCreateTemporaryFile** provides the possibility to manage temporary files so that the memory can be freed for other ends and thus optimizes system performance.	
Global Variables		**Filepath**: Retrieves the path to where the file was created. This is available as an **After** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Creating a temporary file and writing data in it

The following scenario describes a simple Job that creates an empty temporary file in a defined directory, writes data in it and deletes it after Job execution.

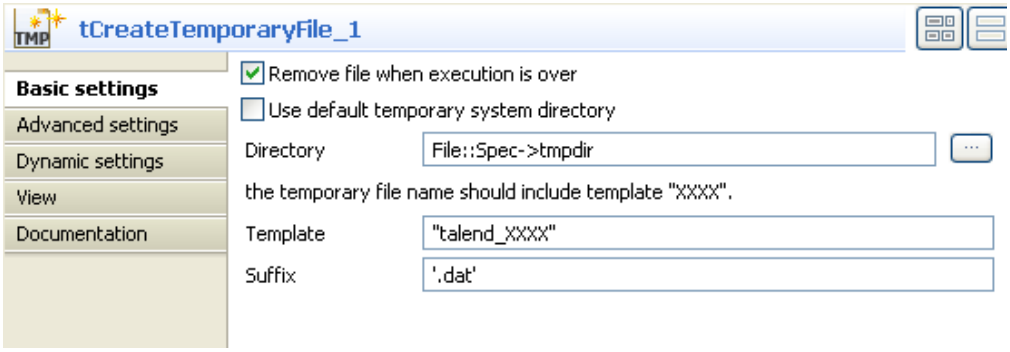
Dropping and linking components

1. Drop the following components from the **Palette** onto the design workspace: **tCreate temporaryFile**, **tRowGenerator**, **tFileOutputDelimited**, **tFileInputDelimited** and **tLogRow**.
2. Connect **tCreateTemporaryFile** to **tRowGenerator** using a **SubjobOk** link.
3. Connect **tRowGenerator** to **tFileOutputDelimited** using a **Row Main** link.
4. Connect **tRowGenerator** to **tFileInputDelimited** using a **SubjobOk** link.
5. Connect **tFileInputDelimited** to **tLogRow** using a **Row Main** link.

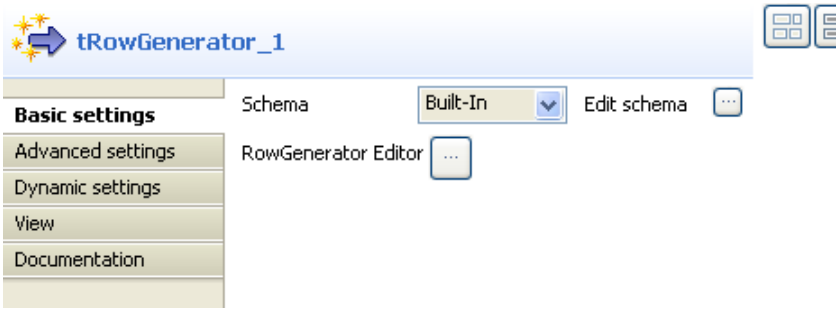


Configuring the components

1. In the design workspace, select **tCreateTemporaryFile**.
2. Click the **Component** tab to define the basic settings for **tCreateTemporaryFile**.

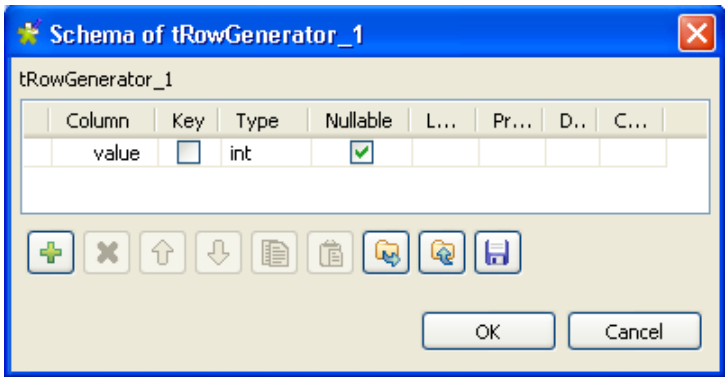


3. Select the **Remove file when execution is over** check box to delete the created temporary file when Job execution is over.
4. Click the three-dot button next to the **Directory** field to browse to the directory where temporary files will be stored, or enter the path manually.
5. In the **Template** field, enter a name for the temporary file respecting the template format.
6. In the **Suffix** field, enter a filename extension to indicate the file format you want to give to the temporary file.
7. In the design workspace, select **tRowGenerator** and click the **Component** tab to define its basic settings.

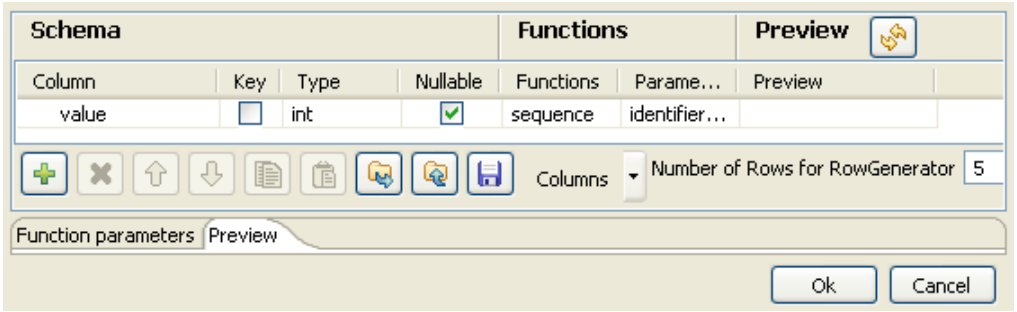


8. Set the **Schema** to **Built-In**.
9. Click the **Edit schema** three-dot button to define the data to pass on to the **tFileOutputDelimited** component, one column in this scenario, *value*.

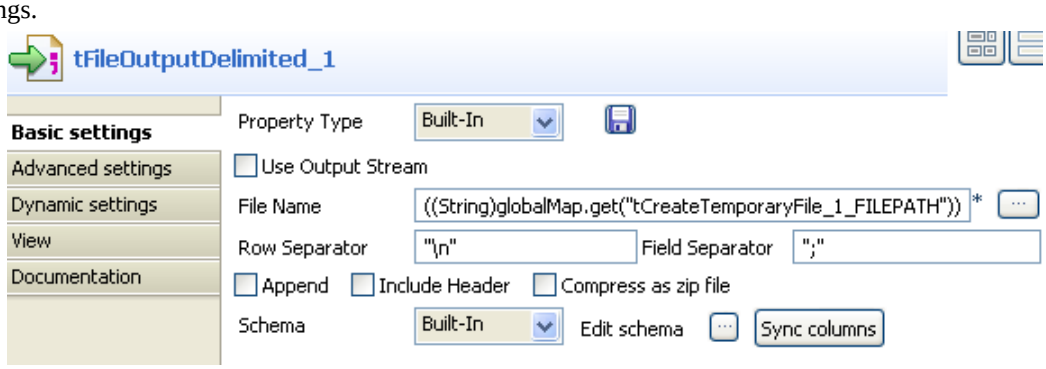
   Click **OK** to close the dialog box.



10. Click the **RowGenerator Editor** three-dot button to open the editor dialog box.



11. In the **Number of Rows for Rowgenerator** field, enter *5* to generate five rows and click **Ok** to close the dialog box.
12. In the design workspace, select **tFileOutputDelimited** and click the **Component** tab to define its basic settings.



13. Set **Property Type** to **Built-In**.

14. Click in the **File Name** field and use the **Ctrl+Space bar** combination to access the variable completion list. To output data in the created temporary file, select `tCreateTemporaryFile_1.FILEPATH` on the global variable list.

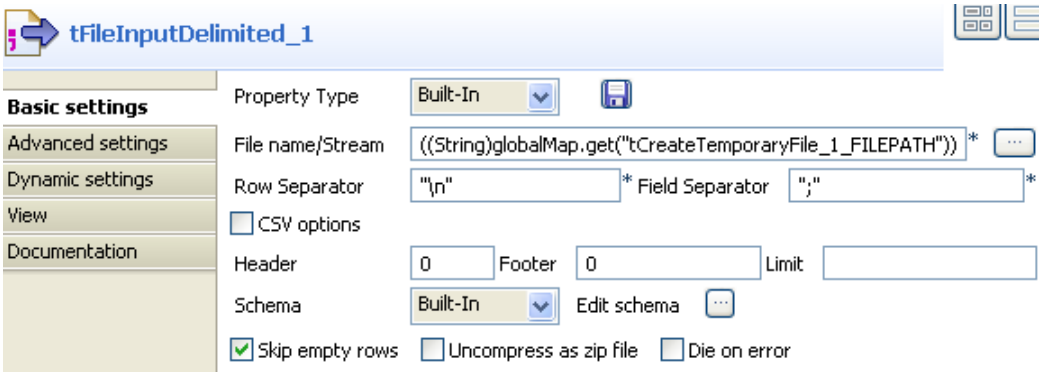
15. Set the row and field separators in their corresponding fields as needed.

16. Set **Schema** to **Built-In** and click **Sync columns** to synchronize input and output columns. Note that the row connection feeds automatically the output schema.

    For more information about schema types, see *Talend Open Studio for Big Data User Guide*.

17. In the design workspace, select the **tFileInputDelimited** component.

18. Click the **Component** tab to define the basic settings of **tFileInputDelimited**.



19. Click in the **File Name** field and use the **Ctrl+Space bar** combination to access the variable completion list. To read data in the created temporary file, select `tCreateTemporaryFile_1.FILEPATH` on the global variable list.

20. Set the row and field separators in their corresponding fields as needed.

21. Set **Schema** to **Built in** and click **Edit schema** to define the data to pass on to the **tLogRow** component. The schema consists of one column here, *value*.

Saving and executing the Job

1. Press **Ctrl+S** to save the Job.

2. Press **F6** to execute the Job or click the **Run** button of the **Run** tab.

The temporary file is created in the defined directory during Job execution and the five generated rows are written in it. The temporary file is deleted when Job execution is over.

tChangeFileEncoding

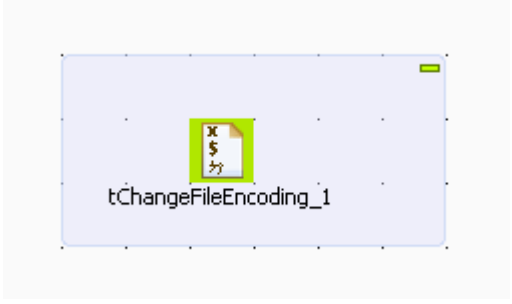
tChangeFileEncoding Properties

Component family	File/Management	
Function	**tChangeFileEncoding** changes the encoding of a given file.	
Purpose	**tChangeFileEncoding** transforms the character encoding of a given file and generates a new file with the transformed character encoding.	
Basic settings	*Use Custom Input Encoding*	Select this check box to customize input encoding type. When it is selected, a list of input encoding types appears, allowing you to select an input encoding type or specify an input encoding type by selecting **CUSTOM**.
	Encoding	From this list of character encoding types, you can select one of the offered options or customize the character encoding by selecting **CUSTOM** and specifying a character encoding type.
	Input File Name	Path of the input file.
	Output File Name	Path of the output file.
Usage	This component can be used as standalone component.	
Limitation	n/a	

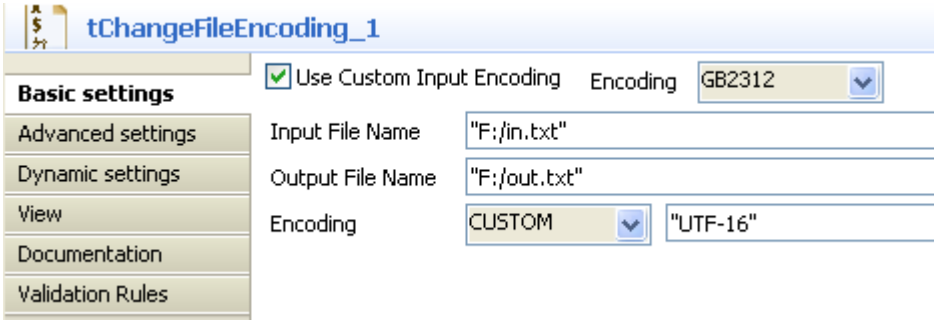
Scenario: Transforming the character encoding of a file

This Java scenario describes a very simple Job that transforms the character encoding of a text file and generates a new file with the new character encoding.

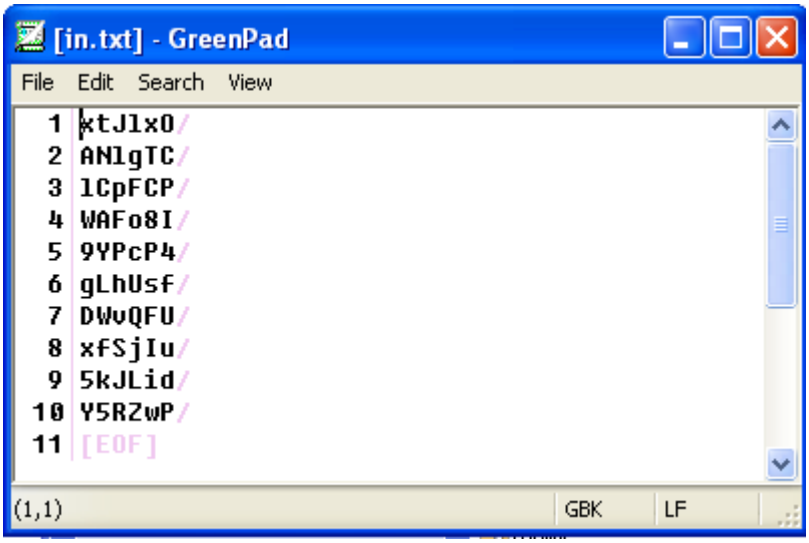
1. Drop a **tChangeFileEncoding** component onto the design workspace.



2. Double-click the **tChangeFileEncoding** component to display its **Basic settings** view.



3. Select **Use Custom Input Encoding** check box. Set the **Encoding** type to **GB2312**.
4. In the **Input File Name** field, enter the file path or browse to the input file.
5. In the **Output File Name** field, enter the file path or browse to the output file.
6. Select **CUSTOM** from the second **Encoding** list and enter *UTF-16* in the text field.
7. Press **F6** to execute the Job.



The encoding type of the file *in.txt* is transformed and *out.txt* is generated with the UTF-16 encoding type.

tFileArchive

tFileArchive properties

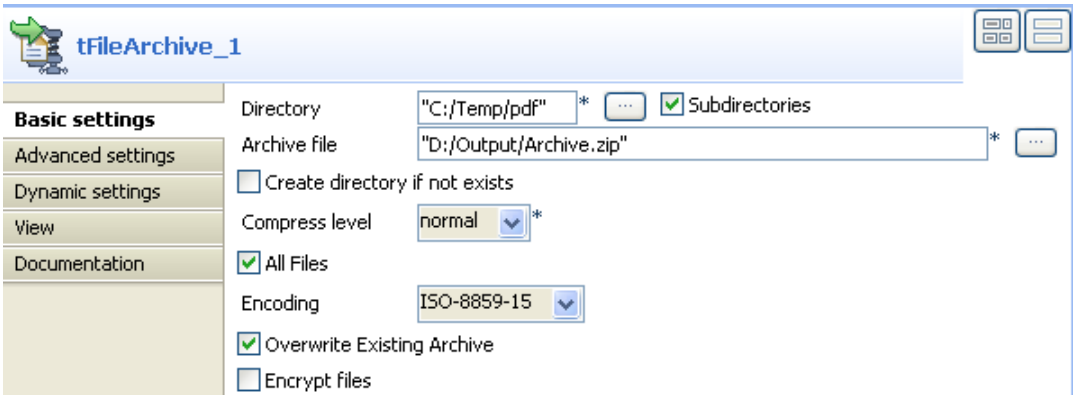
Component Family	File/Management	
Function	The **tFileArchive** zips one or several files according to the parameters defined and places the archive created in the directory selected.	
Purpose	This component zips one or several files for processing.	
Basic settings	*Directory*	Path where the zipped file will be created. **Subdirectories**: Select this check box if the selected directory contains subfolders.
	Archive file	Destination path and name of the archive file.
	Compress level	Select the compression level you want to apply. **Best**: the compression quality will be optimum, but the compression time will be long. **Normal**: compression quality and time will be average. **Fast**: compression will be fast, but quality will be lower.
	All files	Select this check box if you want all files in the directory to be zipped. Clear it to specify the file(s) you want to zip in the **Files** table. **Filemask**: type in a file name or a file mask using a special character or a regular expression.
	Create directory if not exists	This check box is selected by default. It creates a destination folder for the output table if it does not already exist.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Overwrite Existing Archive	This check box is selected by default. This allows you to save an archive by replacing the existing one. But if you clear the check box, an error is reported, the replacement fails and the new archive cannot be saved. When the replacement fails, the Job runs.
	Encrypt files	Select this check box if you want your archive to be password protected. The **Enter Password** text box appears to let you enter your password.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component must be used as a standalone component.	
Global Variables		**Archive File Path**: Retrieves the path to the archive file. This is available as an **After** variable. Returns a string. **Archive File Name**: Retrieves the name of the archive file. This is available as an **After** variable. Returns a string.

		For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

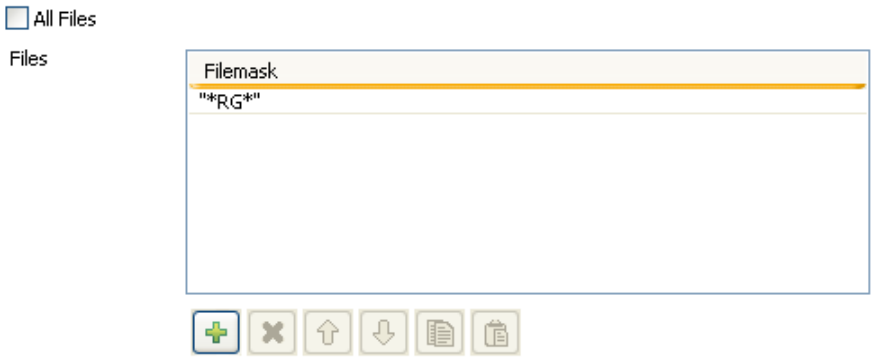
Scenario: Zip files using a tFileArchive

This scenario creates a Job with a unique component. It aims at zipping files and recording them in the selected directory.

1. Drop the **tFileArchive** component from the **Palette** onto the workspace.
2. Double-click it to display its **Component** view.



3. In the **Directory** field, click the **[...]** button, browse your directory and select the directory or the file you want to compress.
4. Select the **Subdirectories** check box if you want to include the subfolders and their files in the archive.
5. Then, set the **Archive file** field, by filling the destination path and the name of your archive file.
6. Select the **Create directory if not exists** check box if you do not have a destination directory yet and you want to create it.
7. In the **Compress level** list, select the compression level you want to apply to your archive. In this example, we use the **normal** level.
8. Clear the **All Files** check box if you only want to zip specific files.



9. Add a row in the table by clicking the **[+]** button and click the name which appears. Between two star symbols (ie. *RG*), type part of the name of the file that you want to compress.

10. Press **F6** to execute your Job.

The **tFileArchive** has compressed the selected file(s) and created the folder in the selected directory.

tFileCompare

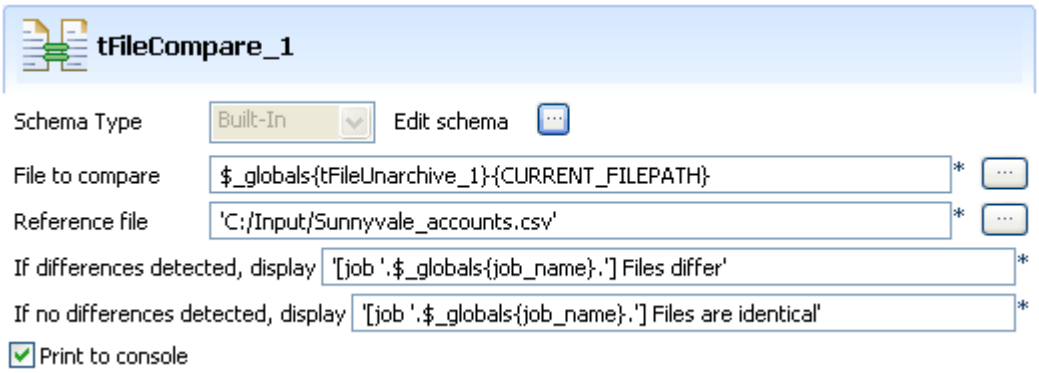
tFileCompare properties

Component family	File/Management	
Function	Compares two files and provides comparison data (based on a read-only schema)	
Purpose	Helps at controlling the data quality of files being processed.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
	File to compare	Filepath to the file to be checked.
	Reference file	Filepath to the file, the comparison is based on.
	If differences are detected, display and *If no difference detected, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the message.
Advanced settings	*Encoding*	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Global Variables		**Difference**: Checks whether two files are identical or not. This is available as a **Flow** variable. Returns a boolean value: - true if the two files are identical. - false if there is a difference between them. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

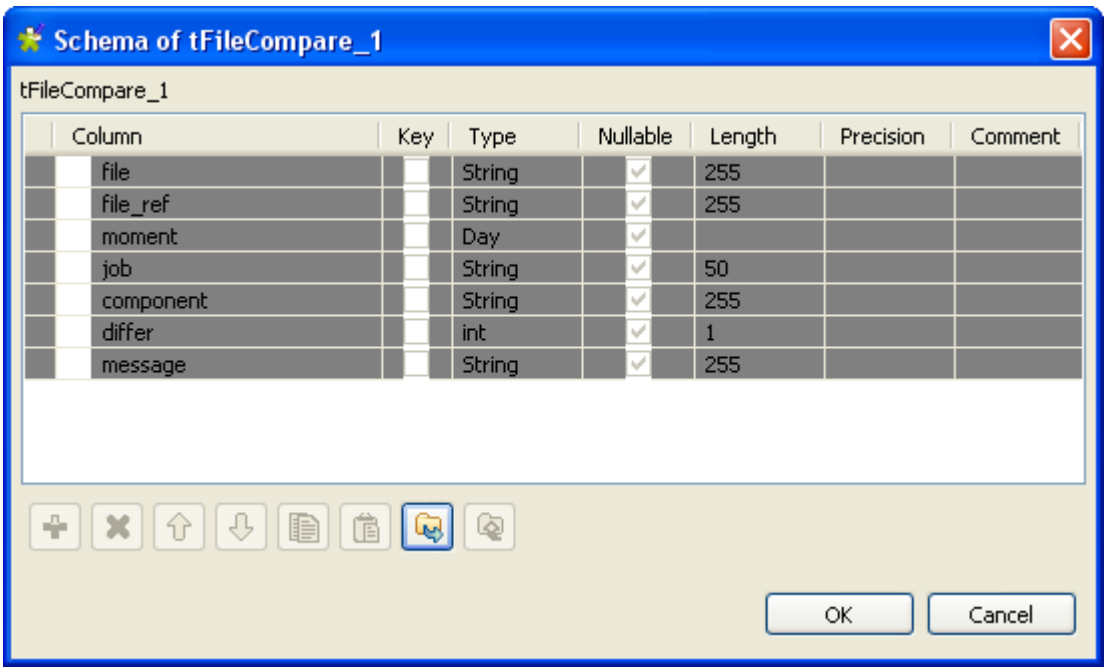
Scenario: Comparing unzipped files

This scenario describes a Job unarchiving a file and comparing it to a reference file to make sure it did not change. The output of the comparison is stored into a delimited file and a message displays in the console.

1. Drag and drop the following components: **tFileUnarchive**, **tFileCompare**, and **tFileOutputDelimited**.
2. Link the **tFileUnarchive** to the **tFileCompare** with **Iterate** connection.
3. Connect the **tFileCompare** to the output component, using a **Main** row link.
4. In the **tFileUnarchive** component **Basic settings**, fill in the path to the archive to unzip.
5. In the **Extraction Directory** field, fill in the destination folder for the unarchived file.
6. In the **tFileCompare Basic settings**, set the **File to compare**. Press *Ctrl+Space bar* to display the list of global variables. Select *$_globals{tFileUnarchive_1}{CURRENT_FILEPATH}* or *"((String)globalMap.get("tFileUnarchive_1_CURRENT_FILEPATH"))"* according to the language you work with, to fetch the file path from the **tFileUnarchive** component.



7. And set the **Reference file** to base the comparison on it.
8. In the messages fields, set the messages you want to see if the files differ or if the files are identical, for example: *"[job " + JobName + "] Files differ"*.
9. Select the **Print to Console** check box, for the message defined to display at the end of the execution.
10. The schema is read-only and contains standard information data. Click **Edit schema** to have a look to it.



11. Then set the output component as usual with semi-colon as data separators.

12. Save your Job and press **F6** to run it.

```
Starting job CompareFiles at 14:11 19/06/2007.
[job CompareFiles] Files differ
Job CompareFiles ended at 14:11 19/06/2007. [exit code=0]
```

The message set is displayed to the console and the output shows the schema information data.

```
file;file_ref;moment;job;component;differ;message
C:\Input\Accounts\Sunnyvale_accounts_new.xls;C:/Input/Sunnyvale_accounts.csv;
2007-06-19 14:11:59;CompareFiles;tFileCompare_1;1;[job CompareFiles] Files
differ
```

tFileCopy

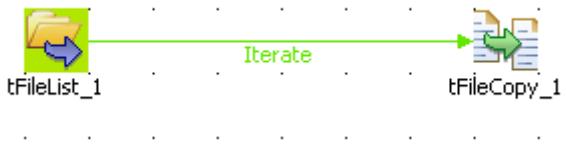
tFileCopy Properties

Component family	File/Management	
Function	Copies a source file into a target directory and can remove the source file if required.	
Purpose	Helps to streamline processes by automating recurrent and tedious tasks such as copy.	
Basic settings	*File Name*	Path to the file to be copied or moved
	Destination	Path to the directory where the file is copied/moved to.
	Remove source file	Select this check box to move the file to the destination.
	Replace existing file	Select this check box to overwrite any existing file with the newly copied file.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as standalone component.	
Global Variables		**Destination File Name**: Retrieves the name of the destination file. This is available as an **After** variable. Returns a string. **Destination File Path**: Retrieves the path to the destination file. This is available as an **After** variable. Returns a string. **Source Directory**:.Retrieves the path to the source directory. This is available as an **After** variable. Returns a string. **Destination Directory**: Retrieves the path to the destination directory. This is available as an **After** variable. Returns a stirng. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize.

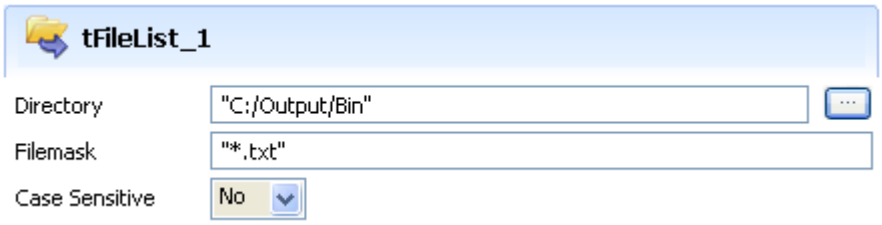
		For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Restoring files from bin

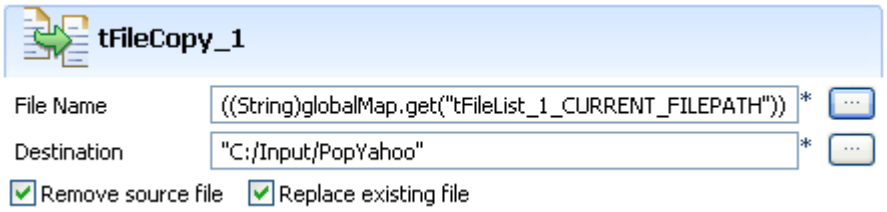
This scenario describes a Job that iterates on a list of files, copies each file from the defined source directory to a target directory. It then removes the copied files from the source directory.



1. Drop a **tFileList** and a **tFileCopy** from the **Palette** to the design workspace.
2. Link both components using an **Iterate** link.
3. In the **tFileList Basic settings**, set the directory for the iteration loop.



4. Set the **Filemask** to *"*.txt"* to catch all files with this extension. For this use case, the case is not sensitive.
5. Then select the **tFileCopy** to set its **Basic settings**.



6. In the **File Name** field, press **Ctrl+Space** bar to access the list of variables.
7. Select the global variable *((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))*. All files from the source directory can be processed.
8. Select the **Remove Source file** check box to get rid of the file that have been copied.
9. Select the **Replace existing file** check box to overwrite any file possibly present in the destination directory.
10. Save your Job and press **F6**.

The files are copied onto the destination folder and are removed from the source folder.

tFileDelete

tFileDelete Properties

Component family	File/Management	
Function	Suppresses a file from a defined directory.	
Purpose	Helps to streamline processes by automating recurrent and tedious tasks such as delete.	
Basic settings	*File Name*	Path to the file to be deleted. This field is hidden when you select the **Delete folder** check box or the **Delete file or folder** check box.
	Directory	Path to the folder to be deleted. This field is available only when you select the **Delete folder** check box.
	File or directory to delete	Enter the path to the file or to the folder you want to delete. This field is available only when you select the **Delete file or folder** check box.
	Fail on error	Select this check box to prevent the main Job from being executed if an error occurs, for example, if the file to be deleted does not exist.
	Delete Folder	Select this check box to display the **Directory** field, where you can indicate the path the folder to be deleted.
	Delete file or folder	Select this check box to display the **File or directory to delete** field, where you can indicate the path to the file or to the folder you want to delete.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as standalone component.	
Global Variables		**Delete path**: Returns the path to the location from which the item was deleted. This is available as an **After** variable. Returns a string. **Current Status**: Indicates whether an item has been deleted or not. This is available as a **Flow** variable. Returns a string and the delete command label. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.

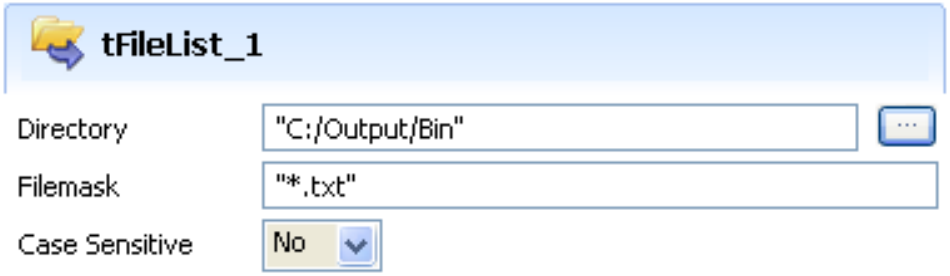
Limitation	n/a

Scenario: Deleting files

This very simple scenario describes a Job deleting files from a given directory.

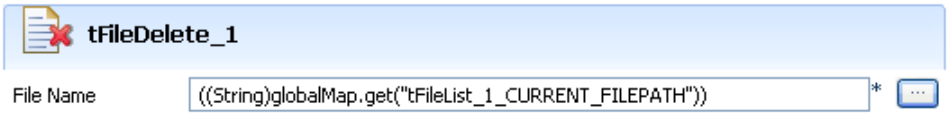
1. Drop the following components: **tFileList**, **tFileDelete**, **tJava** from the **Palette** to the design workspace.

2. In the **tFileList Basic settings**, set the directory to loop on in the **Directory** field.



3. The filemask is *"*.txt"* and no case check is to carry out.

4. In the **tFileDelete Basic settings** panel, set the **File Name** field in order for the current file in selection in the **tFileList** component be deleted. This delete all files contained in the directory, as specified earlier.



5. press **Ctrl+Space bar** to access the list of global variables. In Java, the relevant variable to collect the current file is: ((String)globalMap.get("tFileList_1_CURRENT_FILEPATH")).

6. Then in the **tJava** component, define the message to be displayed in the standard output (Run console). In this Java use case, type in the Code field, the following script: System.out.println(((String)globalMap.get("tFileList_1_CURRENT_FILE"))

   + " has been deleted!");

7. Then save your Job and press **F6** to run it.

```
Starting job FileDel at 18:29 20/06/2007.
16.txt has been deleted!
15.txt has been deleted!
14.txt has been deleted!
13.txt has been deleted!
12.txt has been deleted!
11.txt has been deleted!
10.txt has been deleted!
09.txt has been deleted!
08.txt has been deleted!
07.txt has been deleted!
06.txt has been deleted!
05.txt has been deleted!
04.txt has been deleted!
03.txt has been deleted!
02.txt has been deleted!
01.txt has been deleted!
Job FileDel ended at 18:29 20/06/2007. [exit code=0]
```

The message set in the **tJava** component displays in the log, for each file that has been deleted through the **tFileDelete** component.

tFileExist

tFileExist Properties

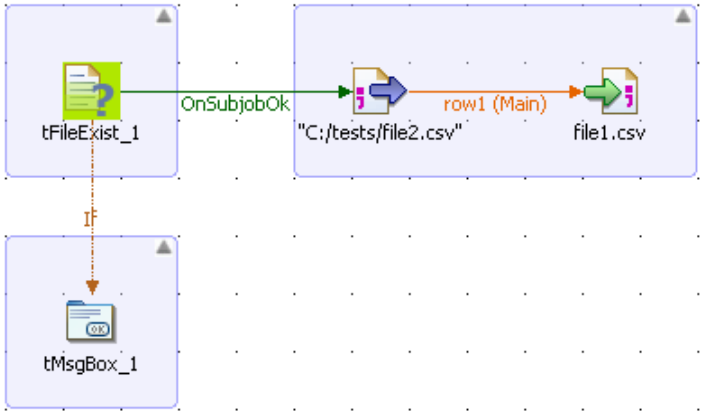
Component family	File/Management	
Function	**tFileExist** checks if a file exists or not.	
Purpose	**tFileExists** helps to streamline processes by automating recurrent and tedious tasks such as checking if a file exists.	
Basic settings	*File Name*	Path to the file you want to check if it exists or not.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as standalone component.	
Global Variables		**Exists**: Indicates whether a specified file exists or not. This is available as a **Flow** variable Returns a boolean value: - true if the file exists. - false if the file does not exist. **File Name**: Retrieves the name and path to a file. This is available as an **After** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Checking for the presence of a file and creating it if it does not exist

This scenario describes a simple Job that: checks if a given file exists, displays a graphical message to confirm that the file does not exist, reads the input data in another given file and writes it in an output delimited file.

Dropping and linking the components

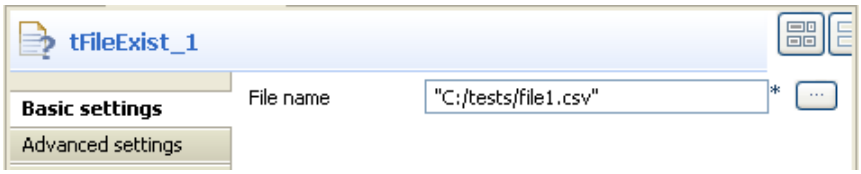
1. Drop the following components from the **Palette** onto the design workspace: **tFileExist**, **tFileInputDelimited**, **tFileOutputDelimited**, and **tMsgBox**.
2. Connect **tFileExist** to **tFile InputDelimited** using an **OnSubjobOk** and to **tMsgBox** using a **Run If** link.



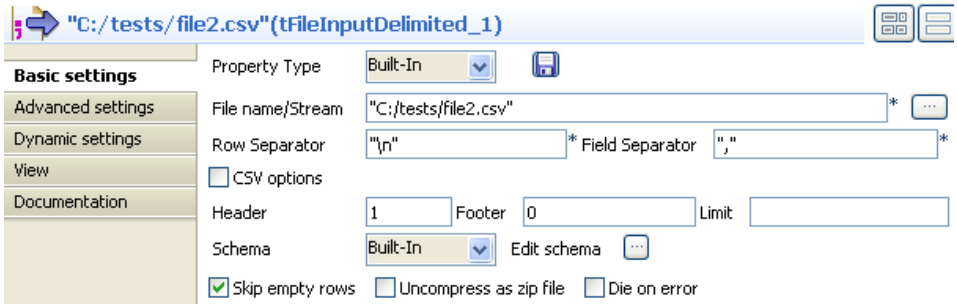
3. Connect **tFileInputDelimited** to **tFileOutputDelimite** using a **Row Main** link.

Configuring the components

1. In the design workspace, select **tFileExist** and click the **Component** tab to define its basic settings.



2. In the **File name** field, enter the file path or browse to the file you want to check if it exists or not.
3. In the design workspace, select **tFileInputDelimited** and click the **Component** tab to define its basic settings.

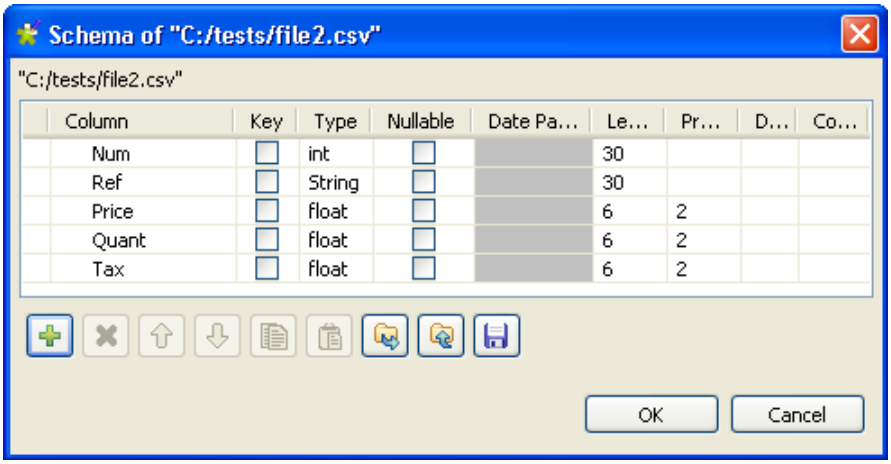


4. Browse to the input file you want to read to fill out the **File Name** field.

> *If the path of the file contains some accented characters, you will get an error message when executing your Job. For more information regarding the procedures to follow when the support of accented characters is missing, see the Installation Guide of the **Talend** solution you are using.*

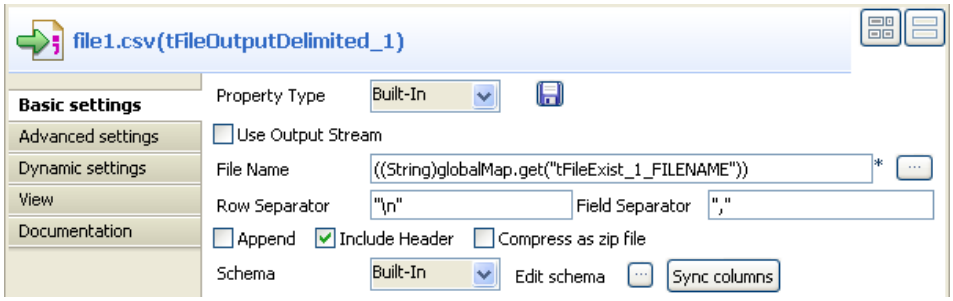
5. Set the row and field separators in their corresponding fields.
6. Set the header, footer and number of processed rows as needed. In this scenario, there is one header in our table.
7. Set **Schema** to **Built-in** and click the **Edit schema** button to define the data to pass on to the **tFileOutputDelimited** component. Define the data present in the file to read, file2 in this scenario.

For more information about schema types, see *Talend Open Studio for Big Data User Guide.*

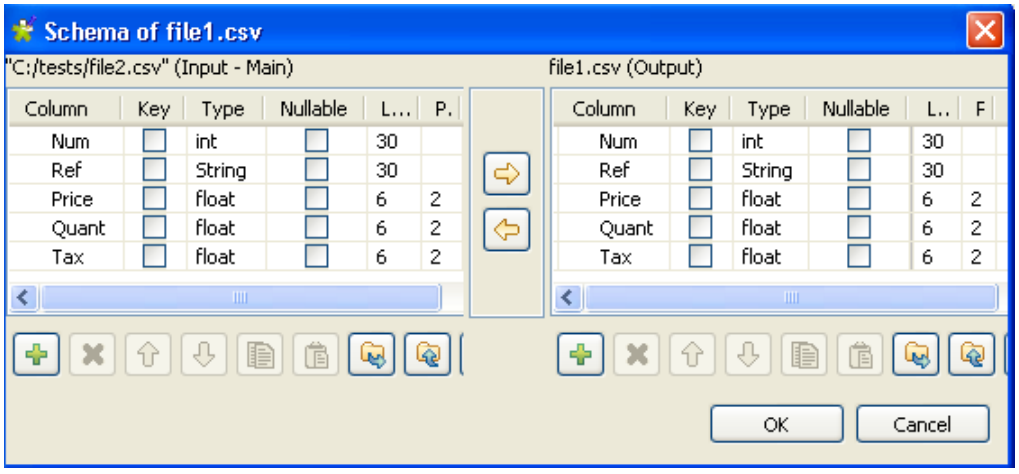


The schema in file2 consists of five columns: *Num*, *Ref*, *Price*, *Quant*, and *tax*.

8. In the design workspace, select the **tFileOutputDelimited** component.
9. Click the **Component** tab to define the basic settings of **tFileOutputDelimited**.

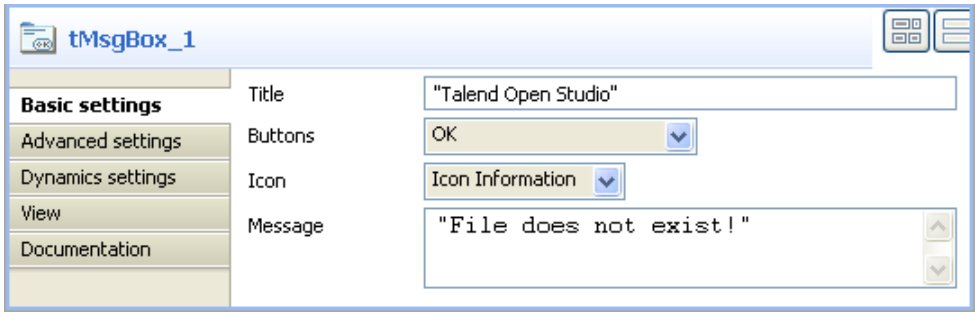


10. Set property type to **Built-in**.
11. In the **File name** field, press **Ctrl+Space** to access the variable list and select the global variable FILENAME.
12. Set the row and field separators in their corresponding fields.
13. Select the **Include Header** check box as file2 in this scenario includes a header.
14. Set **Schema** to **Built-in** and click **Sync columns** to synchronize the output file schema (file1) with the input file schema (file2).



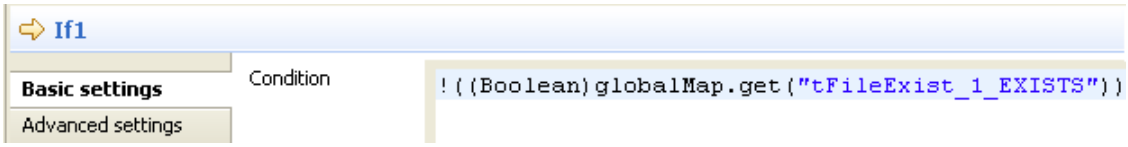
15. In the design workspace, select the **tMsgBox** component.

16. Click the **Component** tab to define the basic settings of **tMsgBox**.



17. Click the **If** link to display its properties in the **Basic settings** view.

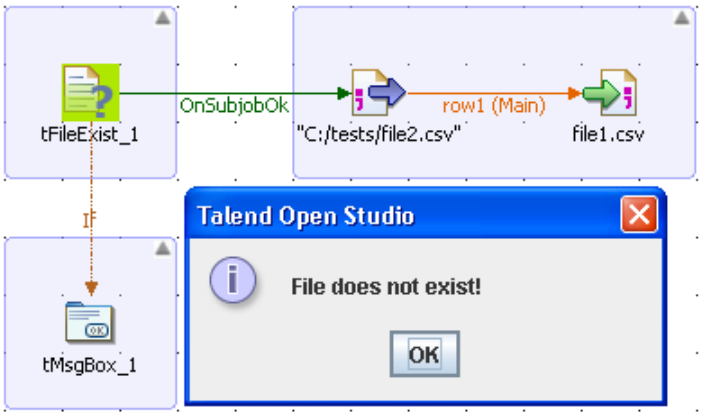
18. In the **Condition** panel, press **Ctrl+Space** to access the variable list and select the global variable EXISTS. Type an exclamation mark before the variable to negate the meaning of the variable.



Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click the **Run** button in the **Run** tab to execute it.



A dialog box appears to confirm that the file does not exists.

Click **OK** to close the dialog box and continue the Job execution process. The missing file, file1 in this scenario, got written in a delimited file in the defined place.

tFileInputARFF

tFileInputARFF properties

Component Family	File/Input	
Function	**tFileInputARFF** reads a ARFF file row by row, with simple separated fields.	
Purpose	This component opens a file and reads it row by row, in order to divide it in fields and to send these fields to the next component, as defined in the schema, through a Row connection.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name and path of the ARFF file and/or variable to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to read a file and separate the fields with the specified separator.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

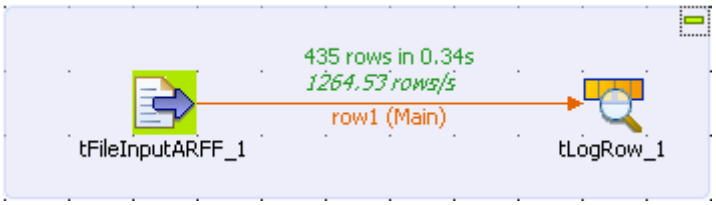
Scenario: Display the content of a ARFF file

This scenario describes a two-component Job in which the rows of an ARFF file are read, the delimited data is selected and the output is displayed in the **Run** view.

An ARFF file looks like the following:

```
@relation vote
@attribute 'handicapped-infants' { 'n', 'y'}
@attribute 'water-project-cost-sharing' { 'n', 'y'}
@attribute 'adoption-of-the-budget-resolution' { 'n', 'y'}
@attribute 'physician-fee-freeze' { 'n', 'y'}
@attribute 'el-salvador-aid' { 'n', 'y'}
@attribute 'religious-groups-in-schools' { 'n', 'y'}
@attribute 'anti-satellite-test-ban' { 'n', 'y'}
@attribute 'aid-to-nicaraguan-contras' { 'n', 'y'}
@attribute 'mx-missile' { 'n', 'y'}
@attribute 'immigration' { 'n', 'y'}
@attribute 'synfuels-corporation-cutback' { 'n', 'y'}
@attribute 'education-spending' { 'n', 'y'}
@attribute 'superfund-right-to-sue' { 'n', 'y'}
@attribute 'crime' { 'n', 'y'}
@attribute 'duty-free-exports' { 'n', 'y'}
@attribute 'export-administration-act-south-africa' { 'n', 'y'}
@attribute 'Class' { 'democrat', 'republican'}
@data
'n','y','n','y','y','y','n','n','n','y',?,'y','y','y','n','y','republican'
'n','y','n','y','y','y','n','n','n','n','n','y','y','y','n',?,'republican'
?,'y','y',?,'y','y','n','n','n','n','y','n','y','y','n','n','democrat'
```

It is generally made of two parts. The first part describes the data structure, that is to say the rows which begin by `@attribute` and the second part comprises the raw data, which follows the expression `@data`.

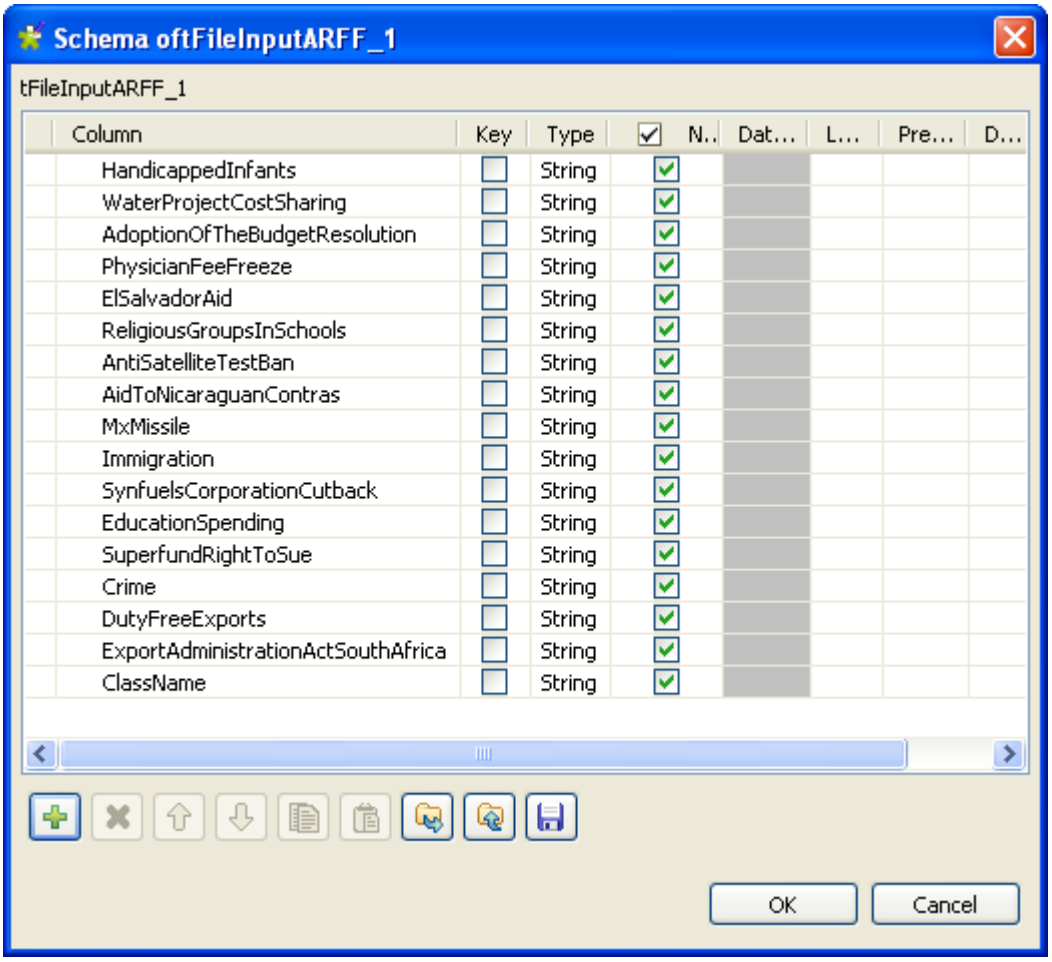


Dropping and linking components

1. Drop the **tFileInputARFF** component from the **Palette** onto the workspace.
2. In the same way, drop the **tLogRow** component.
3. Right-click the **tFileInputARFF** and select **Row > Main** in the menu. Then, drag the link to the **tLogRow**, and click it. The link is created and appears.

Configuring the components

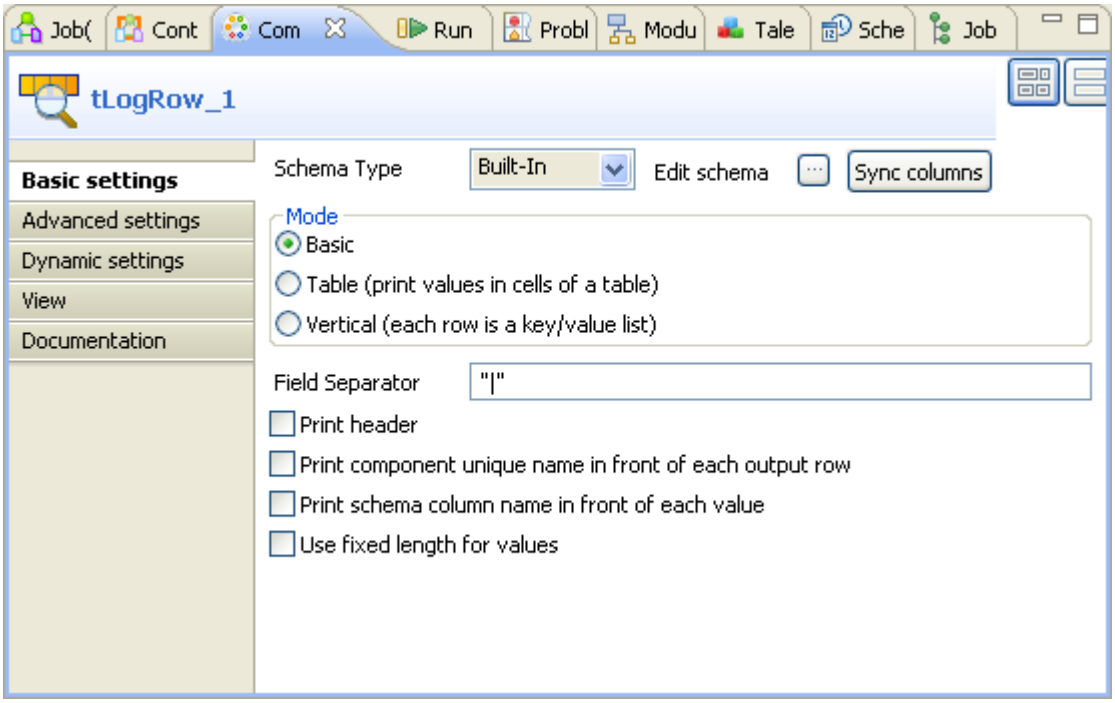
1. Double-click the **tFileInputARFF**.
2. In the **Component** view, in the **File Name** field, browse your directory in order to select your .arff file.
3. In the **Schema** field, select **Built-In**.
4. Click the **[...]** button next to **Edit schema** to add column descriptions corresponding to the file to be read.
5. Click on the button as many times as required to create the number of columns required, according to the source file. Name the columns as follows.



6. For every column, the **Nullable** check box is selected by default. Leave the check boxes selected, for all of the columns.

7. Click **OK**.

8. In the workspace, double-click the **tLogRow** to display its **Component** view.



9. Click the **[...]** button next to **Edit schema** to check that the schema has been propagated. If not, click the **Sync columns** button.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** to execute your Job.

```
n|n|n|y|y|y|y|y|n|y|n|y|y|y|n|y|republican
n|n|n|n|n|n|y|y|y|y|n|n|y|n|y|y|democrat
y|n|y|n|n|n|y|y|y|y|n|y|n|n|y|y|democrat
n|n|y|y|y|y|n|n|y|y|n|y|y|y|n|y|republican
n|n|y|n|n|n|y|y|y|y|n|n|n|n|n|y|democrat
n|n|n|y|y|y|n|n|n|n|y|y|y|y|n|y|republican
n|n|n|y|y|y|n|n|n|n|n|y|y|y|n|y|republican
n|y|n|y|y|y|n|n|n|y|n|y|y|y|n|n|republican
[statistics] disconnected
Job Test ended at 13:36 16/11/2009. [exit code=0]
```

The console displays the data contained in the ARFF file, delimited using a vertical line (the default separator).

tFileInputDelimited

tFileInputDelimited properties

Component family	File/Input	
Function	**tFileInputDelimited** reads a given file row by row with simple separated fields.	
Purpose	Opens a file and reads it row by row to split them up into fields then sends fields as defined in the Schema to the next Job component, via a Row link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name/Stream	**File name**: Name and path of the file to be processed. **Stream:** The data flow to be processed. The data must be added to the flow in order for **tFileInputDelimited** to fetch these data via the corresponding representative variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl+Space**) to fill the current field on condition that this variable has been properly defined. Related topic to the available variables: see *Talend Open Studio for Big Data User Guide*
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	CSV options	Select this check box to include CSV specific parameters such as **Escape char** and **Text enclosure**.
	Header	Number of rows to be skipped in the beginning of file.
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Skip empty rows	Select this check box to skip empty rows.
	Uncompress as zip file	Select this check box to uncompress the input file.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row** > **Reject** link.

		To catch the `FileNotFoundException`, you also need to select this check box.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Extract lines at random	Select this check box to set the number of lines to be extracted randomly.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all column	Select this check box to remove leading and trailing whitespace from all columns.
	Check each row structure against schema	Select this check box to synchronize every row against the input schema.
	Check date	Select this check box to check the date format strictly against the input schema.
	Check columns to trim	Select the check box next to the column name you want to remove leading and trailing whitespace from.
	Split row before field	Select this check box to split rows before splitting fields.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to read a file and separate fields contained in this file using a defined separator. It allows you to create a data flow using a **Row > Main** link or via a **Row > Reject** link in which case the data is filtered by data that does not correspond to the type defined. For further information, please see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Delimited file content display

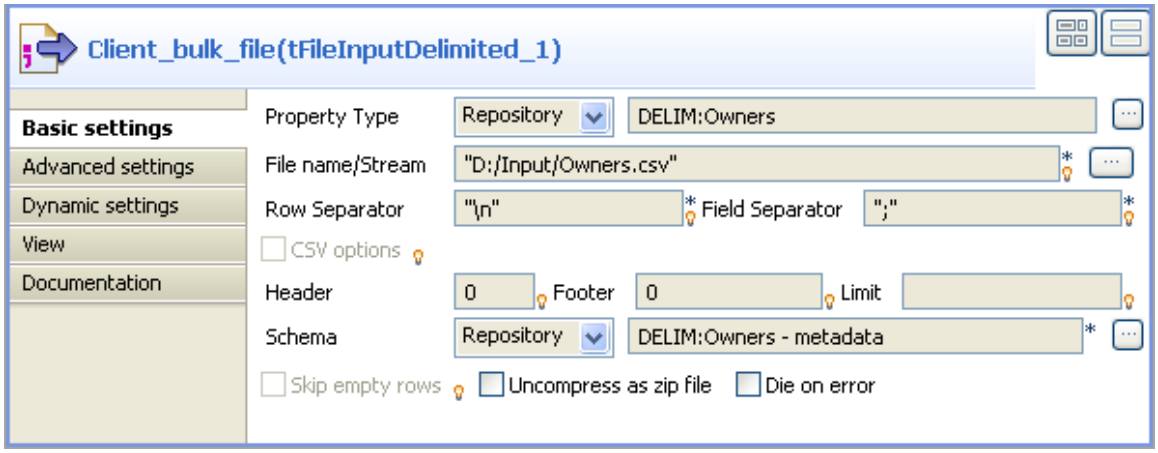
The following scenario creates a two-component Job, which aims at reading each row of a file, selecting delimited data and displaying the output in the **Run** log console.

Dropping and linking components

1. Drop a **tFileInputDelimited** component and a **tLogRow** component from the **Palette** to the design workspace.
2. Right-click on the **tFileInputDelimited** component and select **Row** > **Main**. Then drag it onto the **tLogRow** component and release when the plug symbol shows up.

Configuring the components

1. Select the **tFileInputDelimited** component again, and define its Basic settings:



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Fill in a path to the file in the **File Name** field. This field is mandatory.

   If the path of the file contains some accented characters, you will get an error message when executing your Job. For more information regarding the procedures to follow when the support of accented characters is missing, see the Installation Guide *of the **Talend** Solution you are using.*

3. Define the **Row separator** allowing to identify the end of a row. Then define the **Field separator** used to delimit fields in a row.

4. In this scenario, the header and footer limits are not set. And the **Limit** number of processed rows is set on 50.

5. Edit the schema according to the structure of your input file via the **Edit Schema** function to define the data to pass on to the **tLogRow** component.

   Related topics: see *Talend Open Studio for Big Data User Guide*.

6. Enter the encoding standard the input file is encoded in. This setting is meant to ensure encoding consistency throughout all input and output files.

7. Select the **tLogRow** and define the **Field separator** to use for the output display. Related topic: section *tLogRow*.

8. Select the **Print schema column name in front of each value** check box to retrieve the column labels in the output displayed.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Go to **Run** tab, and click on **Run** to execute the Job.

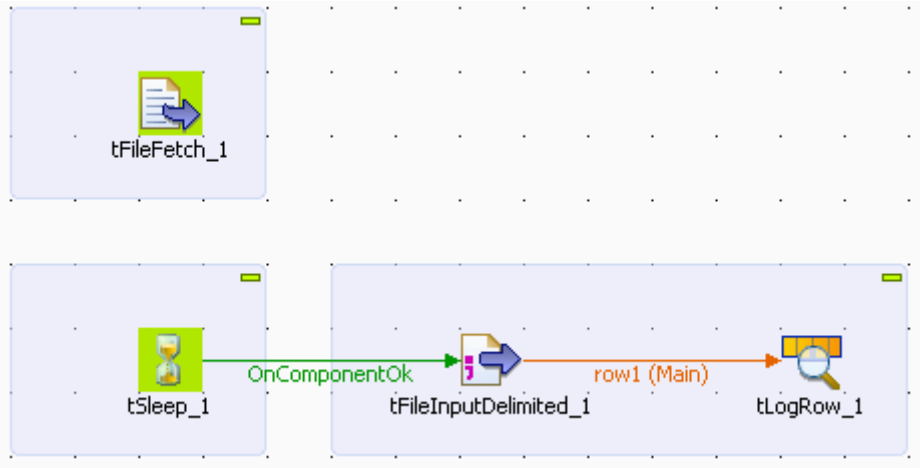
   The file is read row by row and the extracted fields are displayed on the **Run** log as defined in both components **Basic settings**.

```
Starting job FileDel at 10:45 12/04/2007.
ID_Owner: 2|Name: lebouh|ID_Insurance: PKI2906
ID_Owner: 3|Name: bouhnan|ID_Insurance: BNU9147
ID_Owner: 4|Name: hirtle|ID_Insurance: TEV8360
ID_Owner: 5|Name: bobouh|ID_Insurance: WPM3151
ID_Owner: 6|Name: carbone|ID_Insurance: IFY9885
```

   The Log sums up all parameters in a header followed by the result of the Job.

Scenario 2: Reading data from a remote file in streaming mode

This scenario describes a four component Job used to fetch data from a voluminous file almost as soon as it has been read. The data is displayed in the **Run** view. The advantage of this technique is that you do not have to wait for the entire file to be downloaded, before viewing the data.

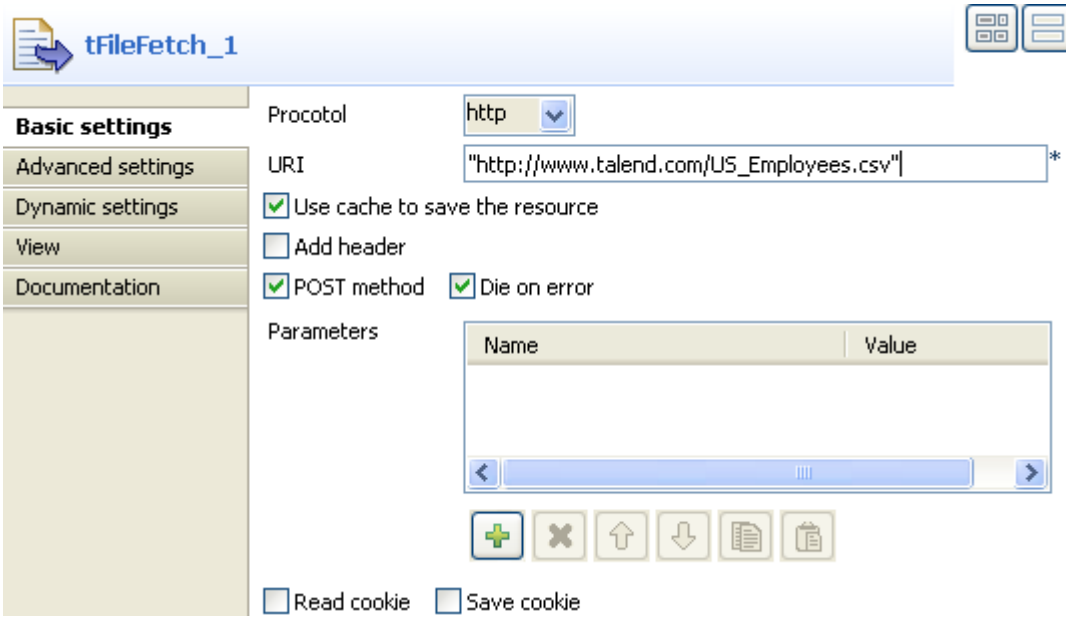


Dropping and linking components

1. Drop the following components onto the workspace: **tFileFetch**, **tSleep**, **tFileInputDelimited**, and **tLogRow**.
2. Connect **tSleep** and **tFileInputDelimited** using a **Trigger** > **OnComponentOk** link and connect **tFileInputDelimited** to **tLogRow** using a **Row** > **Main** link.

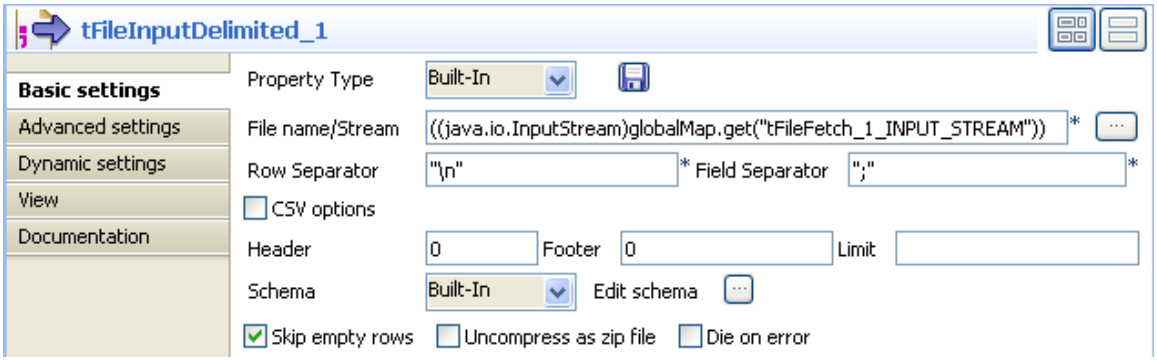
Configuring the components

1. Double-click **tFileFetch** to display the **Basic settings** tab in the **Component** view and set the properties.



2. From the **Protocol** list, select the appropriate protocol to access the server on which your data is stored.
3. In the **URI** field, enter the URI required to access the server on which your file is stored.
4. Select the **Use cache to save the resource** check box to add your file data to the cache memory. This option allows you to use the streaming mode to transfer the data.
5. In the workspace, click **tSleep** to display the **Basic settings** tab in the **Component** view and set the properties.

   By default, **tSleep**'s **Pause** field is set to *1* second. Do not change this setting. It pauses the second Job in order to give the first Job, containing **tFileFetch**, the time to read the file data.
6. In the workspace, double-click **tFileInputDelimited** to display its **Basic settings** tab in the **Component** view and set the properties.



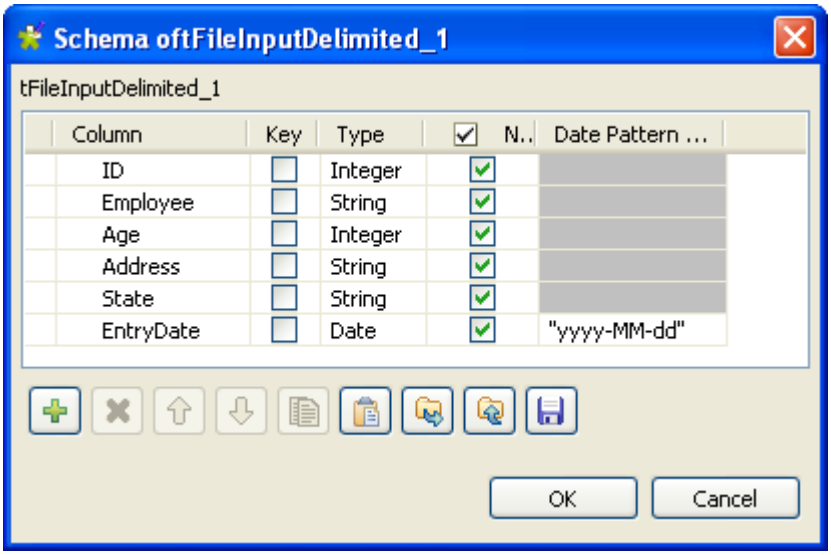
7. In the **File name/Stream** field:

   - Delete the default content.

   - Press **Ctrl+Space** to view the variables available for this component.

   - Select *tFileFetch_1_INPUT_STREAM* from the auto-completion list, to add the following variable to the **Filename** field: `((java.io.InputStream)globalMap.get("tFileFetch_1_INPUT_STREAM"))`.
8. From the **Schema** list, select **Built-in** and click **[...]** next to the **Edit schema** field to describe the structure of the file that you want to fetch. The *US_Employees* file is composed of six columns: *ID*, *Employee*, *Age*, *Address*, *State*, *EntryDate*.

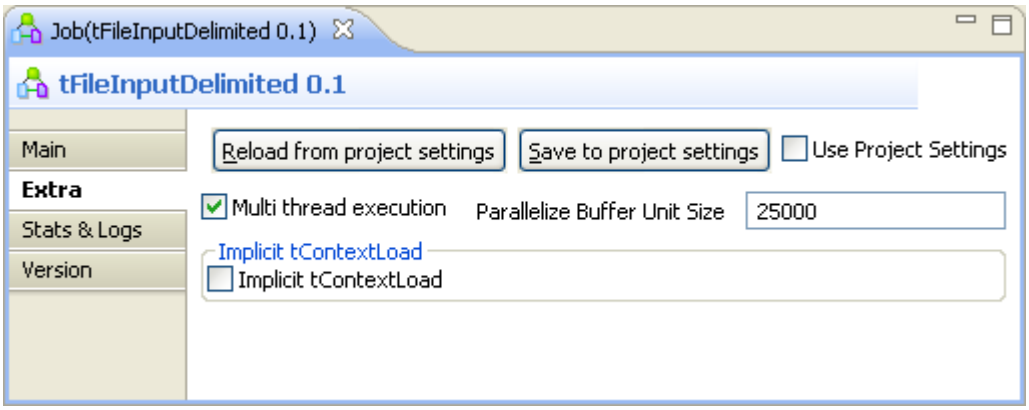
   Click **[+]** to add the six columns and set them as indicated in the above screenshot. Click **OK**.



9. In the workspace, double-click **tLogRow** to display its **Basic settings** in the **Component** view and click **Sync Columns** to ensure that the schema structure is properly retrieved from the preceding component.

Configuring Job execution and executing the Job

1. Click the **Job** tab and then on the **Extra** view.



2. Select the **Multi thread execution** check box in order to run the two Jobs at the same time. Bear in mind that the second Job has a one second delay according to the properties set in **tSleep**. This option allows you to fetch the data almost as soon as it is read by **tFileFetch**, thanks to the **tFileDelimited** component.

3. Save the Job and press **F6** to run it.

```
105730|Harry QUINCY|32|Woodson Rd., Pierre|FLORIDA|2008-03-09
105731|Ronald EISENHOWER|26|East Calle Primera, Saint
Paul|TEXAS|2008-07-03
105732|Ulysses BUCHANAN|32|North Ventu Park Road,
Lansing|ARKANSAS|2008-02-16
105733|Thomas TAFT|45|Moreno Drive, Boise|WEST VIRGINIA|2008-08-08
105734|John EISENHOWER|45|Steele Lane, Indianapolis|ALASKA|2007-07-17
105735|Andrew QUINCY|33|El Camino Real, Boston|WISCONSIN|2008-08-25
105736|Lyndon ADAMS|49|Harbor Dr, Frankfort|MARYLAND|2008-12-28
105737|George EISENHOWER|38|Monroe Street, Sacramento|UTAH|2008-09-04
105738|Grover EISENHOWER|42|Steele Lane, Pierre|NEVADA|2008-01-17
105739|Chester CARTER|39|Fontaine Road, Augusta|WEST VIRGINIA|2007-12-20
105740|Abraham NIXON|30|Monroe Street, Columbia|NEBRASKA|2008-07-29
Job tFileInputDelimited ended at 17:58 09/02/2010. [exit code=0]
```

The data is displayed in the console as almost as soon as it is read.

tFileInputExcel

tFileInputExcel properties

Component family	File/Input	
Function	**tFileInputExcel** reads an Excel file (.xls or .xlsx) and extracts data line by line.	
Purpose	**tFileInputExcel** opens a file and reads it row by row to split data up into fields using regular expressions. Then sends fields as defined in the schema to the next component in the Job via a **Row** link.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Read excel2007 file format (xlsx)	Select this check box to read the .xlsx file of Excel 2007.
	File Name/Stream	**File name**: Name of the file and/or the variable to be processed. **Stream**: Data flow to be processed. The data must be added to the flow in order to be collected by **tFileInputExcel** via the *INPUT_STREAM* variable in the auto-completion list (**Ctrl+Space**). Related topic: see *Talend Open Studio for Big Data User Guide*.
	All sheets	Select this check box to process all sheets of the Excel file.
	Sheet list	Click the plus button to add as many lines as needed to the list of the excel sheets to be processed: **Sheet (name or position)**: enter the name or position of the excel sheet to be processed. **Use Regex**: select this check box if you want to use a regular expression to filter the sheets to process.
	Header	Number of records to be skipped in the beginning of the file.
	Footer	Number of records to be skipped at the end of the file.
	Limit	Maximum number of lines to be processed.
	Affect each sheet(header&footer)	Select this check box if you want to apply the parameters set in the **Header** and **Footer** fields to all excel sheets to be processed.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row** > **Reject** link.
	First column and *Last column*	Define the range of the columns to be processed through setting the first and last columns in the **First column** and **Last column** fields respectively.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.

Advanced settings	*Advanced separator*	Select this check box to change the used data separators.
	Trim all columns	Select this check box to remove the leading and trailing whitespaces from all columns. When this check box is cleared, the **Check column to trim** table is displayed, which lets you select particular columns to trim.
	Check column to trim	This table is filled automatically with the schema being used. Select the check box(es) corresponding to the column(s) to be trimmed.
	Convert date column to string	Available when **Read excel2007 file format (xlsx)** is selected in the **Basic settings** view. Select this check box to show the table **Check need convert date column**. Here you can parse the string columns that contain date values based on the given date pattern. **Column**: all the columns availabe in the schema of the source .xlsx file. **Convert**: select this check box to choose all the columns for conversion (on the condition that they are all of the string type). You can also select the individual check box next to each column for conversion. **Date pattern**: set the date format here.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Read real values for numbers	Select this check box to read numbers in real values. This check box becomes unavailable when you select **Read excel2007 file format (xlsx)** in the **Basic settings** view.
	Stop reading on encountering empty rows	Select this check box to ignore the empty line encountered and, if there are any, the lines that follow this empty line. This check box becomes unavailable when you select **Read excel2007 file format (xlsx)** in the **Basic settings** view.
	Generation mode	Available when **Read excel2007 file format (xlsx)** is selected in the **Basic settings** view. Select the mode used to read the Excel 2007 file. • **Less memory consumed for large excel(Event mode)**: used for large file. This is a memory-saving mode to read the Excel 2007 file as a flow. • **Memory-consuming (User mode)**: used for small file. It needs much memory.
	Don't validate the cells	Select this check box to in order not to validate data. This check box becomes unavailable when you select **Read excel2007 file format (xlsx)** in the **Basic settings** view.
	Ignore the warning	Select this check box to ignore all warnings generated to indicate errors in the Excel file. This check box becomes unavailable when you select **Read excel2007 file format (xlsx)** in the **Basic settings** view.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read an Excel file and to output the data separately depending on the schemas identified in the file. You can use a **Row > Reject** link to filter the data which doesn't correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

No scenario is available for this component yet.

tFileInputFullRow

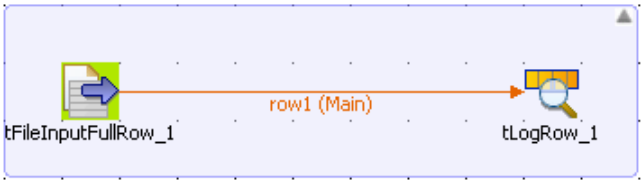
tFileInputFull Row properties

Component family	File/Input	
Function	**tFileInputFullRow** reads a given file row by row.	
Purpose	**tFileInputFullRow** opens a file and reads it row by row and sends complete rows as defined in the Schema to the next Job component, via a Row link.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **tFileInputFullRow**.
	File Name	Name of the file and/or the variable to be processed Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to separate rows.
	Header	Number of rows to be skipped at the beginning of a file
	Footer	Number of rows to be skipped at the end of a file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Skip empty rows	Select this check box to skip empty rows.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row > Reject** link.
Advanced settings	*Encoding*	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Extract lines at random	Select this check box to set the number of lines to be extracted randomly.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read full rows in delimited files that can get very large. You can also create a rejection flow using a **Row > Reject** link to filter the data which does not correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	

Scenario: Reading full rows in a delimited file

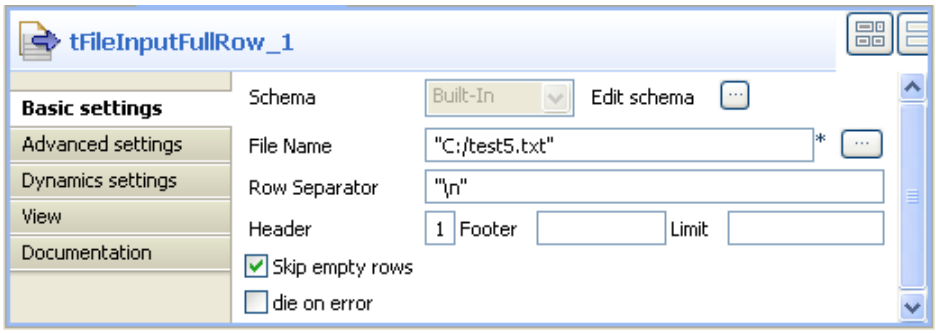
The following scenario creates a two-component Job that aims at reading complete rows in a file and displaying the output in the **Run** log console.

1. Drop a **tFileInputFullRow** and a **tLogRow** from the **Palette** onto the design workspace.
2. Right-click on the **tFileInputFullRow** component and connect it to **tLogRow** using a **Row Main** link.



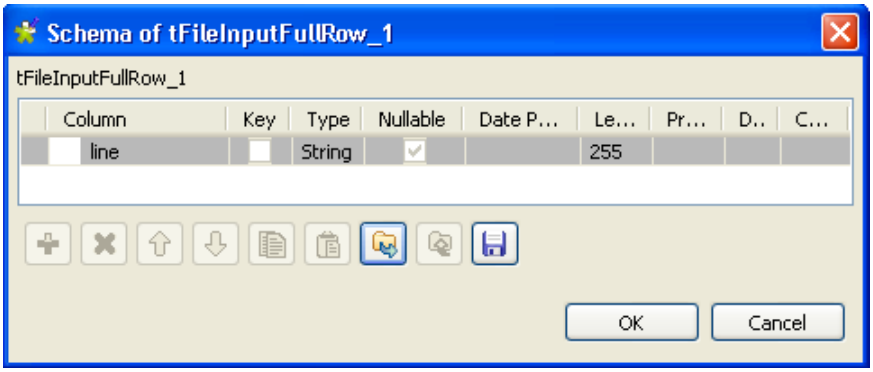
3. In the design workspace, select **tFileInputFullRow**.

4. Click the **Component** tab to define the basic settings for **tFileInputFullRow**.



5. In the **Basic settings** view, set **Schema** to **Built-In**.

6. Click the three-dot **[...]** button next to the **Edit schema** field to see the data to pass on to the **tLogRow** component. Note that the schema is read-only and it consists of one column, *line*.



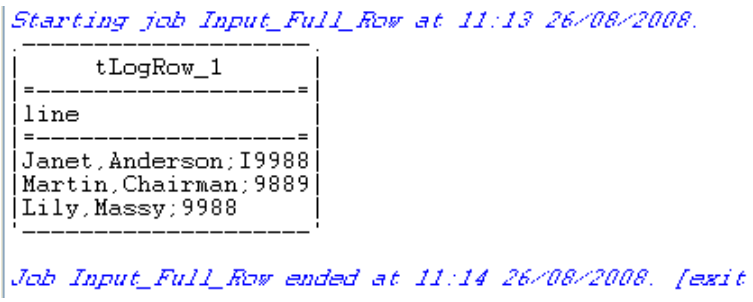
7. Fill in a path to the file to process in the **File Name** field, or click the three-dot [...] button. This field is mandatory. In this scenario, the file to read is *test5*. It holds three rows where each row consists of tow fields separated by a semi colon.

8. Define the **Row separator** used to identify the end of a row.

9. Set the **Header** to 1, in this scenario the footer and the number of processed rows are not set.

10. From the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*

11. Save your Job and press **F6** to execute it.



tFileInputFullRow reads the three rows one by one ignoring field separators, and the complete rows are displayed on the **Run** console.

To extract only fields from rows, you must use **tExtractDelimitedFields**, **tExtractPositionalFields**, and **tExtractRegexFields**. For more information, see section *tExtractDelimitedFields*, section *tExtractPositionalFields* and section *tExtractRegexFields*.

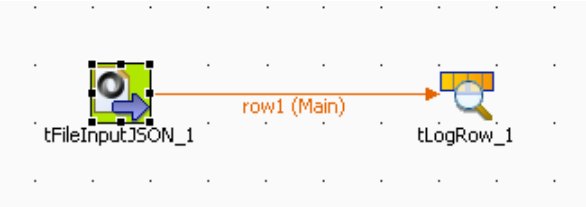
tFileInputJSON

tFileInputJSON properties

Component Family	File	
Function	The **tFileInputJSON** reads a JSON file and extracts data according to the selected schema.	
Purpose	This component opens a file and reads it in order to isolate data according to the schemas which describe this file structure, and to send the data and schemas to the next component(s), via a Row connection.	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use URL	Select this check box to retrieve data directly from the Web. **URL**: type in the URL path from which you will retrieve data.
	FIlename	Name of the file from which you will retrieve data.
	Mapping	**Column**: shows the schema as defined in the Schema editor. **JSONPath Query**: Type in the fields to extract from the JSON input structure.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a JSON file and separate data according to the identified schemas in this file.	
Limitation	n/a	

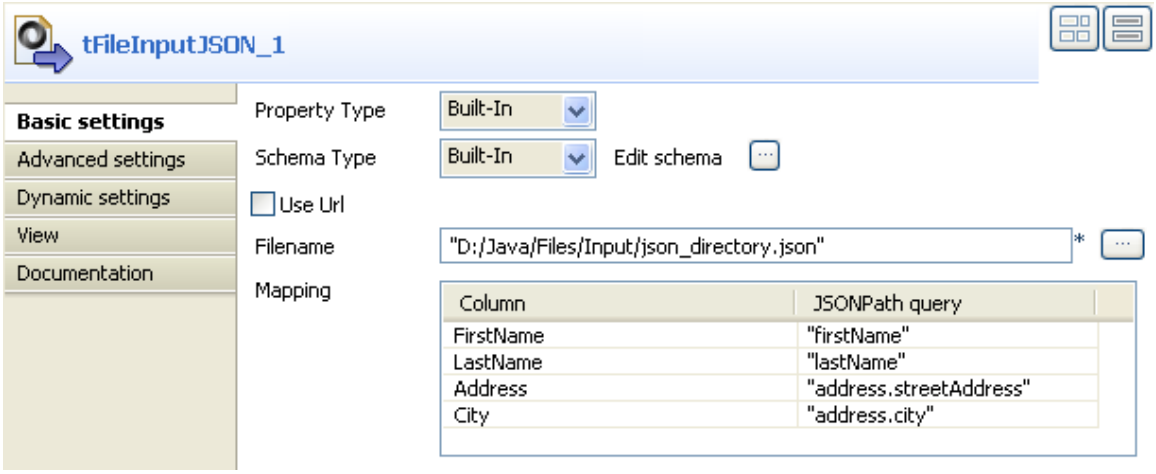
Scenario: Extracting data from the fields of a JSON format file

This is a 2 component scenario which involves reading a JSON file, and extracting its data.



Dropping and linking the components

1. Drag and drop a **tFileInputJSON** component from the **File** family and a **tLogRow** from the **Logs & Errors** family from the **Palette** onto the Job designer.
2. Link the components using a **Main** > **Row** connection.
3. Double-click the **tFileInputJSON** component to set its properties in the **Basic settings**, in the **Component** view:



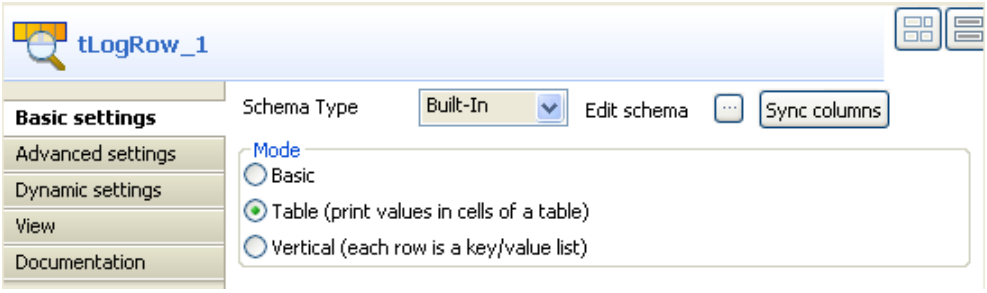
Configuring the components

1. Click the **[...]** button of the **Edit schema** field to open a dialog box in which you will define the output schema to be displayed.
2. Click **OK** to close the dialog box. In the **Mapping** table, the items in the **Column** field are automatically filled in according to the schema you just defined. In this example, the schema is made of four columns: *FirstName*, *LastName*, *Address* and *City*.
3. In the **Filename** field, fill in the path to the JSON file from which you want to retrieve data. If your data are stored on the internet, select the **Use URL** check box, and then, in the same way, fill in the access URL to the file to be processed. In this example, the processed file is presented as follows:

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumbers": [
        { "type": "home", "number": "212 555-1234"
},
        { "type": "fax", "number": "646 555-4567" }
```

4. In the **Mapping** table, the rows in the **Column** field are already filled in. For each of them, type in the tree view level in which retrieve data, in the **JSONPath query** field.

5. In the Job designer, double-click the **tLogRow** to set its properties in the **Basic settings** tab, in the **Component** view.



6. Click **Sync Columns** button to retrieve the schema of the previous component.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

   Press **F6** or click the **Run** button in the **Run** tab to execute it.

2.
```
Starting job tFileInputJSON at 13:54 04/11/2009.

[statistics] connecting to socket on port 3663
[statistics] connected
.---------+--------+-------------+--------.
|               tLogRow_1                  |
|=---------+--------+-------------+--------=|
|FirstName|LastName|Address      |City    |
|=---------+--------+-------------+--------=|
|John     |Smith   |21 2nd Street|New York|
'---------+--------+-------------+--------'

[statistics] disconnected
Job tFileInputJSON ended at 13:54 04/11/2009. [exit code=0]
```

The Job returns the customer information according to the parameters selected in the schema.

tFileInputLDIF

tFileInputLDIF Properties

Component Family	File/Input	
Function	**tFileInputLDIF** reads a given LDIF file row by row.	
Purpose	**tFileInputLDIF** opens a file, reads it row by row, et gives the full rows to the next component as defined in the schema, using a Row connection.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File Name	Name of the file and/or variable to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	add operation as prefix when the entry is modify type	Select this check box to display the operation mode.
	Value separator	Type in the separator required for parsing data in the given file. By default, the separator used is " ,".
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row** > **Reject** link.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to modify the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use field options (for Base64 decode checked)	Select this check box to specify the Base64-encoded columns of the input flow. Once selected, this check box activates the **Decode Base64 encoding values** table to enable you to precise the columns to be decoded from Base64. The data type of the columns to be handled by this check box is **byte[]** that you define in the input schema editor.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read full rows in a voluminous LDIF file. This component enables you to create a data flow, using a **Row** > **Main** link, and to create a reject flow with a **Row** > **Reject** link filtering the data which type does not match the defined type. For an example of usage, see section *Scenario 2: Extracting erroneous XML data via a reject flow* from **tFileInputXML**.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Writing DB data into an LDIF-type file*.

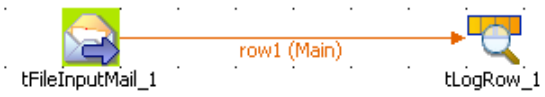
tFileInputMail

tFileInputMail properties

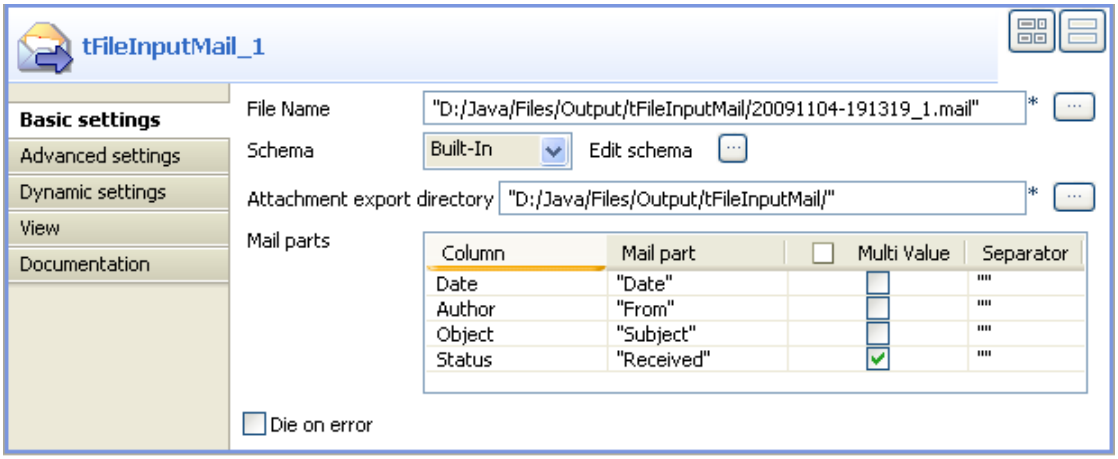
Component family	File/Input	
Function	**tFileInputMail** reads the header and content parts of defined email file.	
Purpose	This component helps to extract standard key data from emails.	
Basic settings	*File name*	Browse to the source email file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Attachment export directory	Enter the path to the directory where you want to export email attachments.
	Mail parts	**Column**: This field is automatically populated with the columns defined in the schema that you propagated.
		Mail part: Type in the label of the header part or body to be displayed on the output.
		Multi value: Select the check box next to the name of the column that is made up of fields of multiple values.
		Field separator*:* Enter a value separator for the field of multiple values.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires output. It is defined as an intermediary step.	
Limitation	n/a	

Scenario: Extracting key fields from an email

This Java scenario describes a two-component Job that extracts some key standard fields and displays the values on the **Run** console.



1. Drop a **tFileInputMail** and a **tLogRow** component from the **Palette** to the design workspace.
2. Connect the two components together using a **Main Row** link.
3. Double-click **tFileInputMail** to display its **Basic settings** view and define the component properties.



4. Click the three-dot button next to the **File Name** field and browse to the mail file to be processed.
5. Set schema type to **Built-in** and click the three-dot button next to **Edit schema** to open a dialog box where you can define the schema including all columns you want to retrieve on your output.
6. Click the plus button in the dialog box to add as many columns as you want to include in the output flow. In this example, the schema has four columns: *Date*, *Author*, *Object* and *Status*.
7. Once the schema is defined, click **OK** to close the dialog box and propagate the schema into the **Mail parts** table.
8. Click the three-dot button next to Attachment export directory and browse to the directory in which you want to export email attachments, if any.
9. In the **Mail part** column of the **Mail parts** table, type in the actual header or body standard keys that will be used to retrieve the values to be displayed.
10. Select the **Multi Value** check box next to any of the standard keys if more than one value for the relative standard key is present in the input file.
11. If needed, define a separator for the different values of the relative standard key in the **Separator** field.
12. Double-click **tLogRow** to display its **Basic settings** view and define the component properties in order for the values to be separated by a carriage return. On Windows OS, type in *\n* between double quotes.
13. Save your Job and press **F6** to execute it and display the output flow on the console.

```
Starting job tFileInputMail at 11:48 06/11/2009.

[statistics] connecting to socket on port 3437
[statistics] connected
Wed, 4 Nov 2009 19:12:47 +0800
(CST)|musicatcher@gmail.com|Talend multi value test|by
10.142.186.14 with SMTP id j14cs69293wff;
        Wed, 4 Nov 2009 03:13:00 -0800 (PST)by 10.150.45.40
with SMTP id s40mr2413104ybs.260.1257333179981;
        Wed, 04 Nov 2009 03:12:59 -0800 (PST)from
mail-gx0-f210.google.com (mail-gx0-f210.google.com
[209.85.217.210])
        by mx.google.com with ESMTP id
33si1593716ywh.127.2009.11.04.03.12.58;
        Wed, 04 Nov 2009 03:12:58 -0800 (PST)by
mail-gx0-f210.google.com with SMTP id 2so6057662gxk.4
        for <musicatcher0@gmail.com>; Wed, 04 Nov 2009 03:12:58
-0800 (PST)by 10.150.75.12 with SMTP id
x12mr2370266yba.341.1257333175484;
        Wed, 04 Nov 2009 03:12:55 -0800 (PST)from nsun
([219.237.242.224])
        by mx.google.com with ESMTPS id
5sm364705ywd.23.2009.11.04.03.12.53
        (version=SSLv3 cipher=RC4-MD5);
        Wed, 04 Nov 2009 03:12:54 -0800 (PST)
[statistics] disconnected
Job tFileInputMail ended at 11:48 06/11/2009. [exit code=0]
```

The header key values are extracted as defined in the **Mail parts** table. Mail reception date, author, subject and status are displayed on the console.

tFileInputMSDelimited

tFileInputMSDelimited properties

Component family	File/Input	
Function	**tFileInputMSDelimited** reads a complex multi-structured delimited file.	
Purpose	**tFileInputMSDelimited** opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next Job components.	
Basic settings	*Multi Schema Editor*	The **[Multi Schema Editor]** helps to build and configure the data flow in a multi-structure delimited file to associate one schema per output. For more information, see section *The Multi Schema Editor*.
	Output	Lists all the schemas you define in the **[Multi Schema Editor]**, along with the related record type and the field separator that corresponds to every schema, if different field separators are used.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Trim all column*	Select this check box to remove leading and trailing whitespaces from defined columns.
	Validate date	Select this check box to check the date format strictly against the input schema.
	Advanced separator (for numbers)	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read multi-structured delimited files and separate fields contained in these files using a defined separator.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

The Multi Schema Editor

The **[Multi Schema Editor]** enables you to:

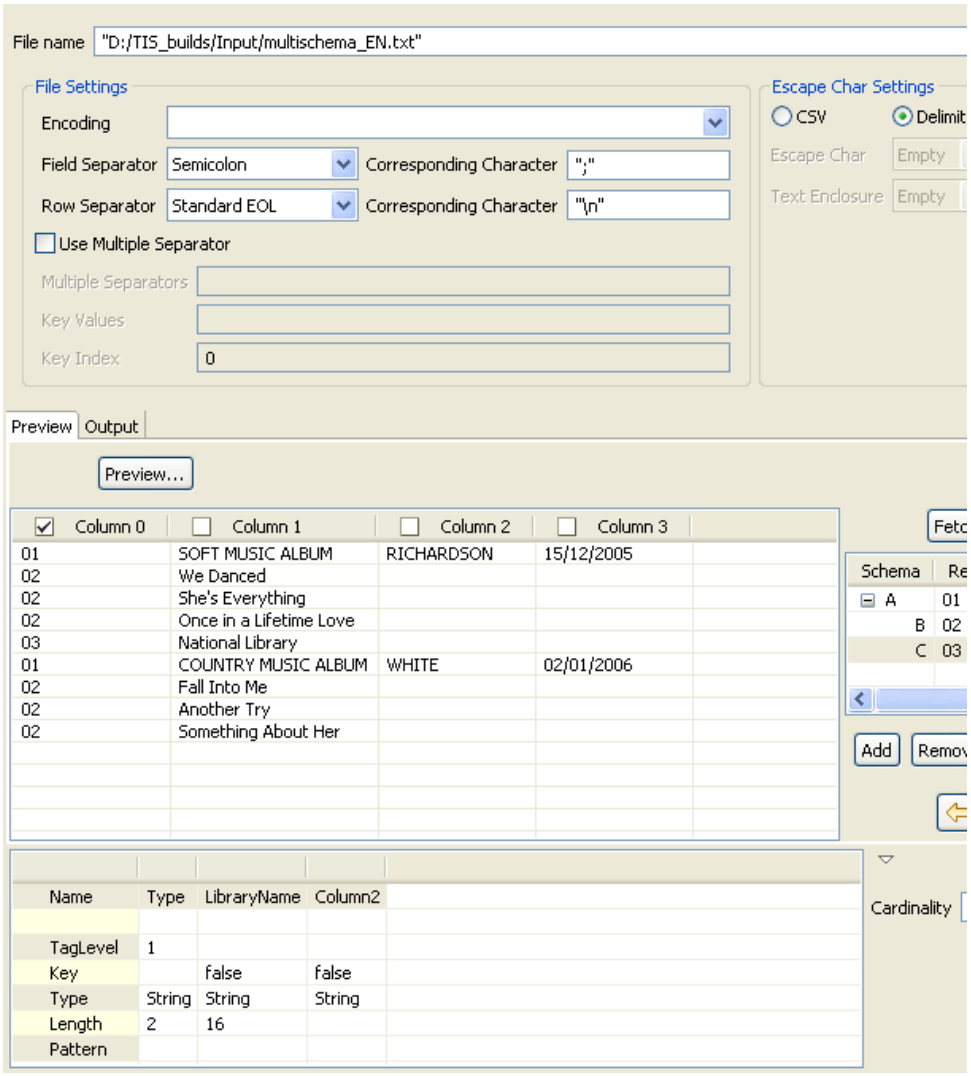
- set the path to the source file,
- define the source file properties,
- define data structure for each of the output schemas.

When you define data structure for each of the output schemas in the **[Multi Schema Editor]**, column names in the different data structures automatically appear in the input schema lists of the components that come after **tFileInputMSDelimited**. However, you can still define data structures directly in the **Basic settings** view of each of these components.

The **[Multi Schema Editor]** also helps to declare the schema that should act as the source schema (primary key) from the incoming data to insure its unicity.The editor uses this mapping to associate all schemas processed in the delimited file to the source schema in the same file.

The editor opens with the first column, that usually holds the record type indicator, selected by default. However, once the editor is open, you can select the check box of any of the schema columns to define it as a primary key.

The below figure illustrates an example of the **[Multi Schema Editor]**.



For detailed information about the usage of the **Multi Schema Editor**, see section *Scenario: Reading a multi structure delimited file*.

Scenario: Reading a multi structure delimited file

The following scenario creates a Java Job which aims at reading three schemas in a delimited file and displaying their data structure on the **Run Job** console.

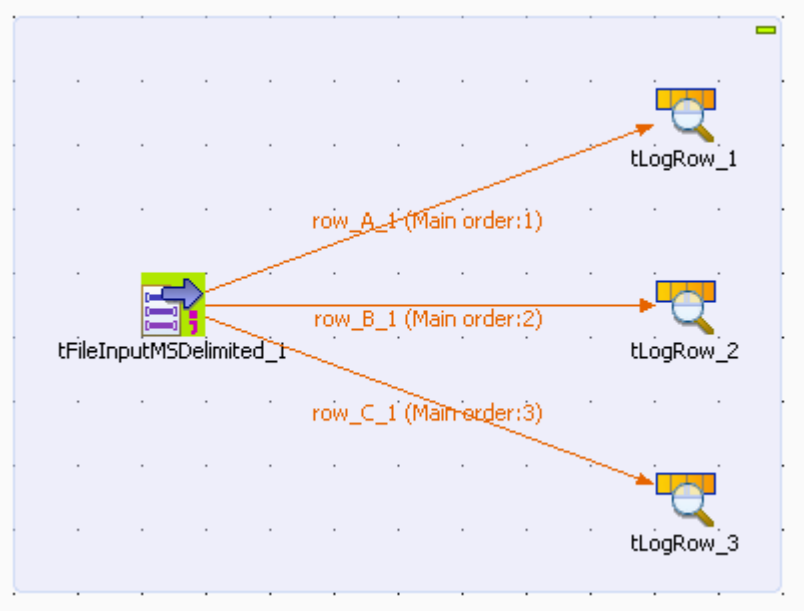
The delimited file processed in this example looks like the following:

```
01;SOFT MUSIC ALBUM;RICHARDSON;15/12/2005
02;We Danced
02;She's Everything
02;Once in a Lifetime Love
03;National Library
01;COUNTRY MUSIC ALBUM;WHITE;02/01/2006
02;Fall Into Me
02;Another Try
02;Something About Her
```

Dropping and linking components

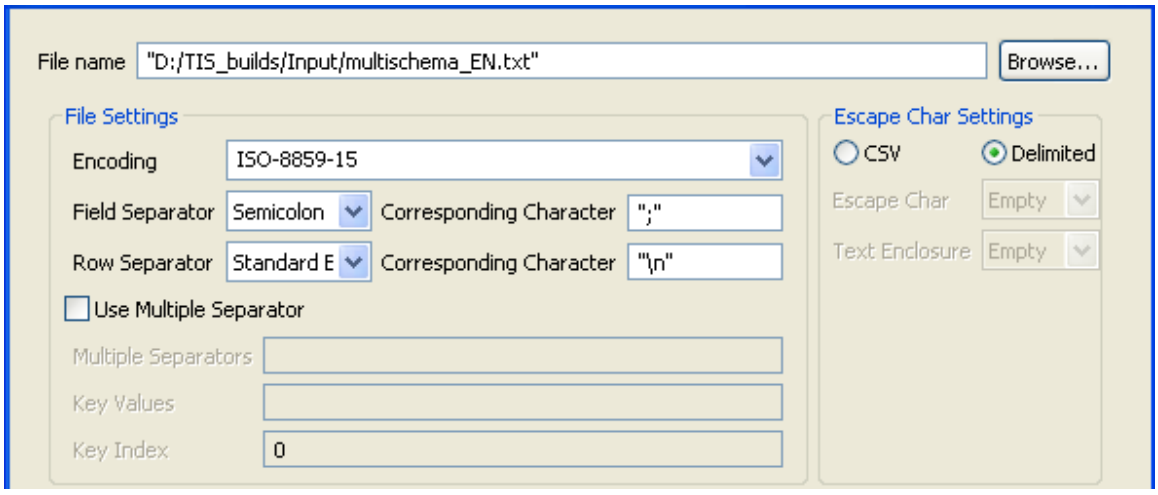
1. Drop a **tFileInputMSDelimited** component and three **tLogRow** components from the **Palette** onto the design workspace.

2. In the design workspace, right-click **tFileInputMSDelimited** and connect it to **tLogRow1**, **tLogRow2**, and **tLogRow3** using the **row_A_1**, **row_B_1**, and **row_C_1** links respectively.



Configuring the components

1. Double-click **tFileInputMSDelimited** to open the **Multi Schema Editor**.



2. Click **Browse...** next to the **File name** field to locate the multi schema delimited file you need to process.

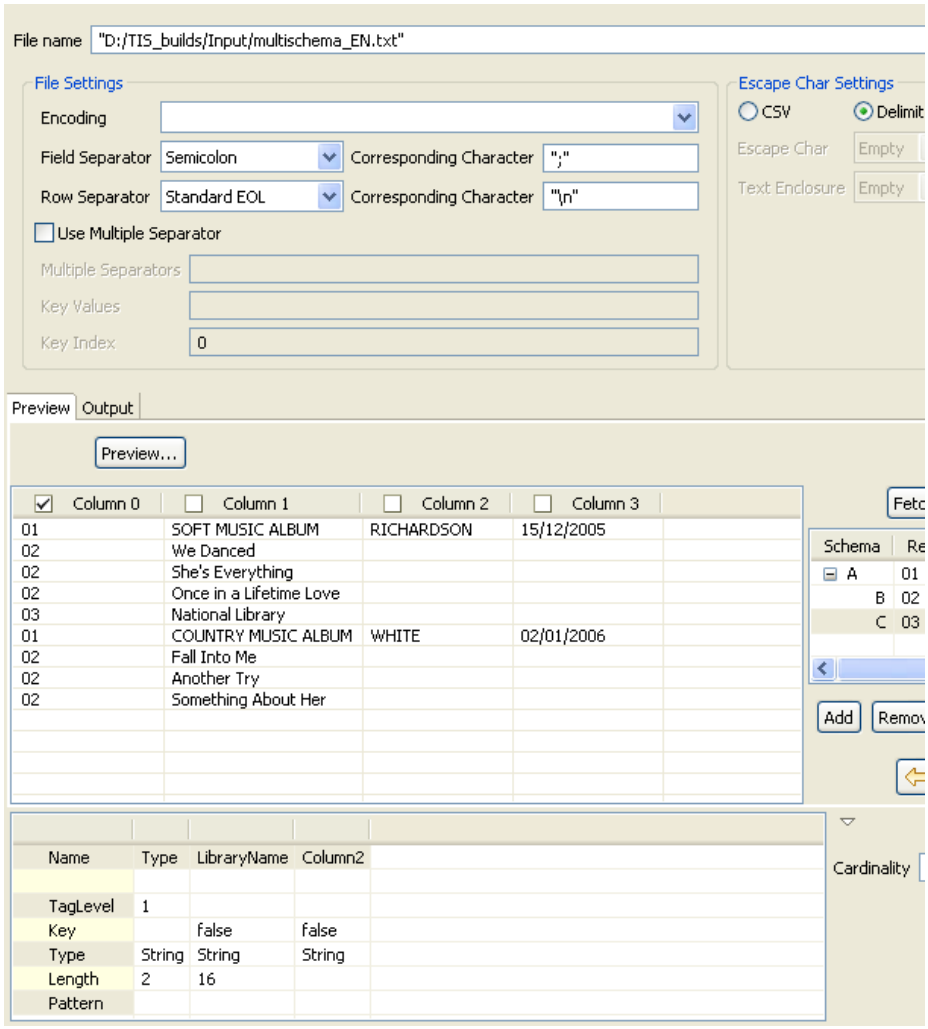
3. In the **File Settings** area:

   -Select from the list the encoding type the source file is encoded in. This setting is meant to ensure encoding consistency throughout all input and output files.

   -Select the field and row separators used in the source file.

Select the **Use Multiple Separator** check box and define the fields that follow accordingly if different field separators are used to separate schemas in the source file.

A preview of the source file data displays automatically in the **Preview** panel.



Column 0 that usually holds the record type indicator is selected by default. However, you can select the check box of any of the other columns to define it as a primary key.

4. Click **Fetch Codes** to the right of the **Preview** panel to list the type of schema and records you have in the source file. In this scenario, the source file has three schema types (*A*, *B*, *C*).

   Click each schema type in the **Fetch Codes** panel to display its data structure below the **Preview** panel.

5. Click in the name cells and set column names for each of the selected schema.

   In this scenario, column names read as the following:

   -Schema *A*: *Type*, *DiscName*, *Author*, *Date*,

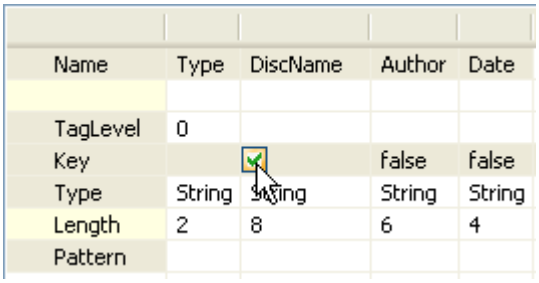
   -Schema *B*: *Type*, *SongName*,

   -Schema *C*: *Type*, *LibraryName*.

   You need now to set the primary key from the incoming data to insure its unicity (*DiscName* in this scenario). To do that:

6. In the **Fetch Codes** panel, select the schema holding the column you want to set as the primary key (schema *A* in this scenario) to display its data structure.

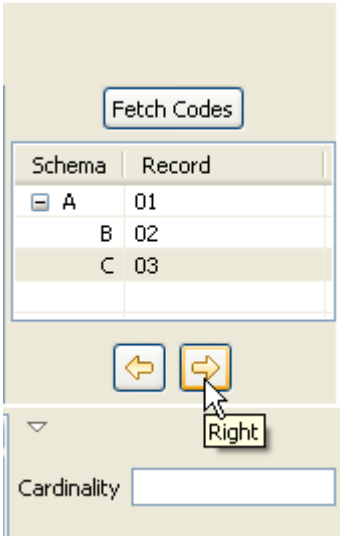
7. Click in the *Key* cell that corresponds to the *DiscName* column and select the check box that appears.



8. Click anywhere in the editor and the *false* in the *Key* cell will become *true*.

   You need now to declare the parent schema by which you want to group the other “children” schemas (*DiscName* in this scenario). To do that:

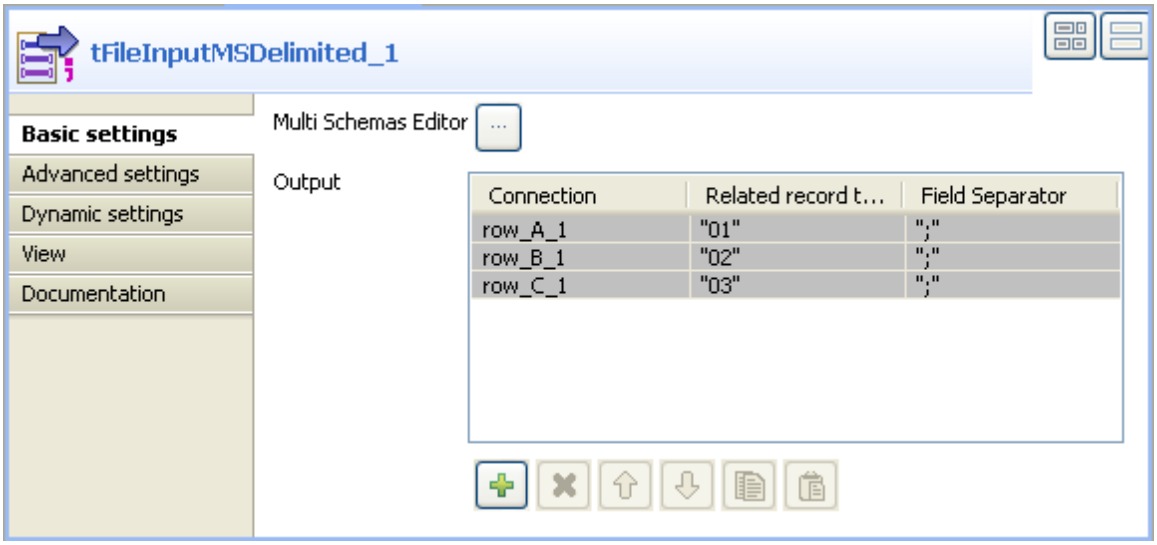
9. In the **Fetch Codes** panel, select schema *B* and click the right arrow button to move it to the right. Then, do the same with schema *C*.



The **Cardinality** field is not compulsory. It helps you to define the number (or range) of fields in “children” schemas attached to the parent schema. However, if you set the wrong number or range and try to execute the Job, an error message will display.

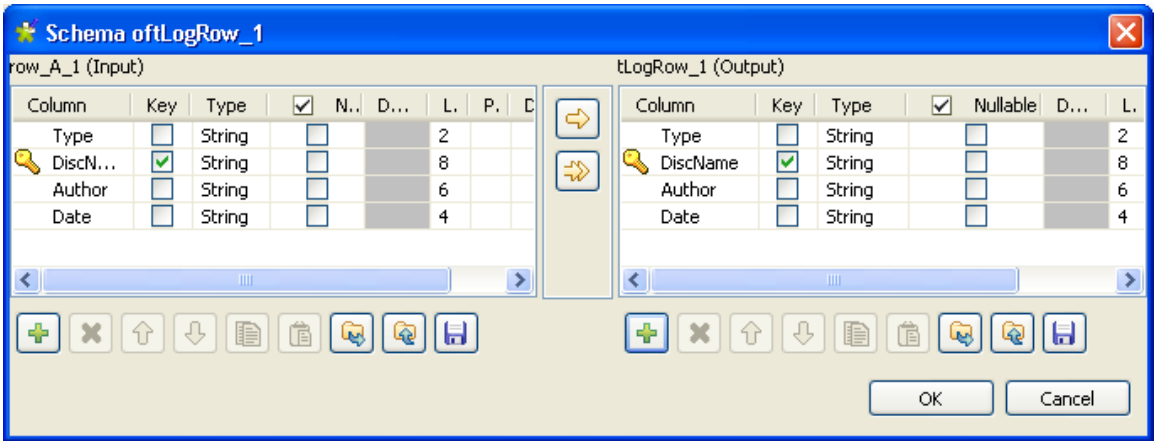
10. In the **[Multi Schema Editor]**, click **OK** to validate all the changes you did and close the editor.

    The three defined schemas along with the corresponding record types and field separators display automatically in the **Basic settings** view of **tFileInputMSDelimited**.



The three schemas you defined in the **[Multi Schema Editor]** are automatically passed to the three **tLogRow** components.

11. If needed, click the **Edit schema** button in the **Basic settings** view of each of the **tLogRow** components to view the input and output data structures you defined in the **Multi Schema Editor** or to modify them.



Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The multi schema delimited file is read row by row and the extracted fields are displayed on the **Run Job** console as defined in the [**Multi Schema Editor**].

```
Starting job pivot at 10:33 18/01/2010.

01|SOFT MUSIC ALBUM|RICHARDSON|15/12/2005
02|We Danced|SOFT MUSIC ALBUM
02|She's Everything|SOFT MUSIC ALBUM
02|Once in a Lifetime Love|SOFT MUSIC ALBUM
03|National Library|SOFT MUSIC ALBUM
01|COUNTRY MUSIC ALBUM|WHITE|02/01/2006
02|Fall Into Me|COUNTRY MUSIC ALBUM
02|Another Try|COUNTRY MUSIC ALBUM
02|Something About Her|COUNTRY MUSIC ALBUM
Job pivot ended at 10:33 18/01/2010. [exit code=0]
```

tFileInputMSPositional

tFileInputMSPositional properties

Component family	File/Input	
Function	**tFileInputMSPositional** reads multiple schemas from a positional file.	
Purpose	**tFileInputMSPositional** opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next Job components.	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	File Name	Name of the file and/or the variable to be processed Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Header Field Position	Start-end position of the schema identifier.
	Records	**Schema**: define as many schemas as needed. **Header value**: value in the row that identifies a schema. **Pattern**: string which represents the length of each column of the schema, separated by commas. Make sure the values defined in this field are relevant with the defined schema. **Reject incorrect row size**: select the check boxes of the schemas where to reject incorrect row size. **Parent row**: Select the parent row from the drop-down list. By default, it is <*Empty*>. **Parent key column**: Type in the parent key column name. If the parent row is not <*Empty*>, this field must be filled with a column name of the parent row schema. **Key column**: Type in the key column name.
	Skip from header	Number of rows to be skipped in the beginning of file.
	Skip from footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Die on parse error	Let the component die if an parsing error occurs.
	Die on unknown header type	Length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this fields are consistent with the schema defined.
Advanced settings	*Process long rows (needed for processing rows longer than 100,000 characters wide)*	Select this check box to process long rows (this is necessary to process rows longer than 100 000 characters).
	Advanced separator (for numbers)	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Trim all column	Select this check box to remove leading and trailing whitespaces from defined columns.

	Validate date	Select this check box to check the date format strictly against the input schema.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a multi schemas positional file and separate fields using a position separator value. You can also create a rejection flow using a **Row > Reject** link to filter the data which does not correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	

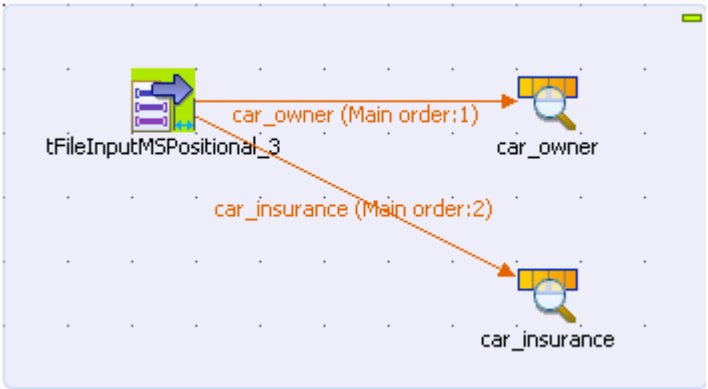
Scenario: Reading data from a positional file

The following scenario reads data from a positional file, which contains two schemas. The positional file is shown below:

```
schema_1 (car_owner):schema_id;car_make;owner;age
schema_2 (car-insurance):schema_id;car_owner;age;car_insurance
1bmw     John      45
1bench   Mike      30
2John      45 yes
2Mike      50 No
```

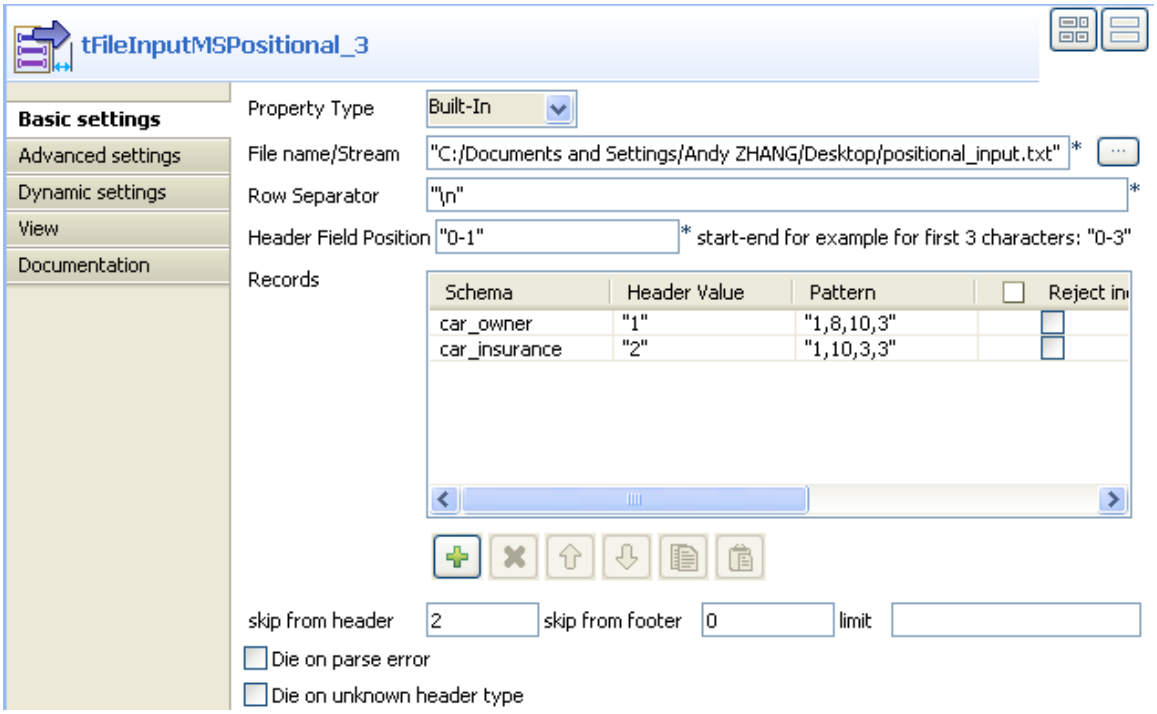
Dropping the components

1. Drop one **tFileInputMSPositional** and two **tLogRow** from the **Palette** to the design workspace.
2. Rename the two **tLogRow** components as **car_owner** and **car_insurance**.



Configuring the components

1. Double-click the **tFileInputMSPositional** component to show its **Basic settings** view and define its properties.



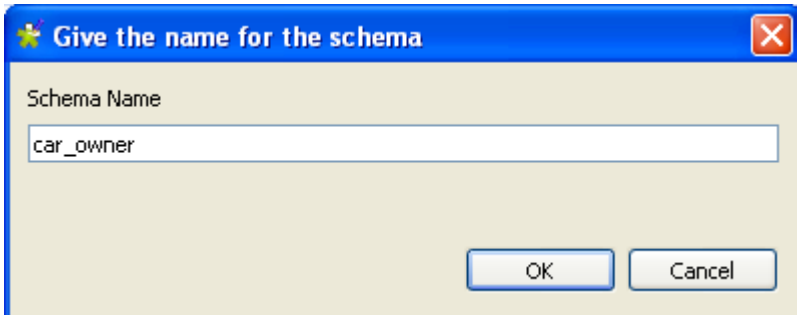
2. In the **File name/Stream** field, type in the path to the input file. Also, you can click the **[...]** button to browse and choose the file.

3. In the **Header Field Position** field, enter the start-end position for the schema identifier in the input file, *0-1* in this case as the first character in each row is the schema identifier.

4. Click the **[+]** button twice to added two rows in the **Records** table.

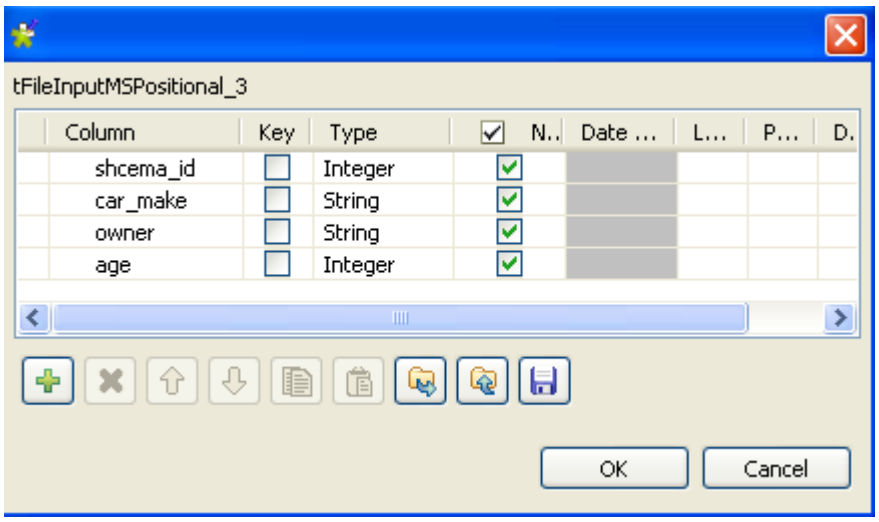
5. Click the cell under the **Schema** column to show the **[...]** button.

   Click the **[...]** button to show the schema naming box.



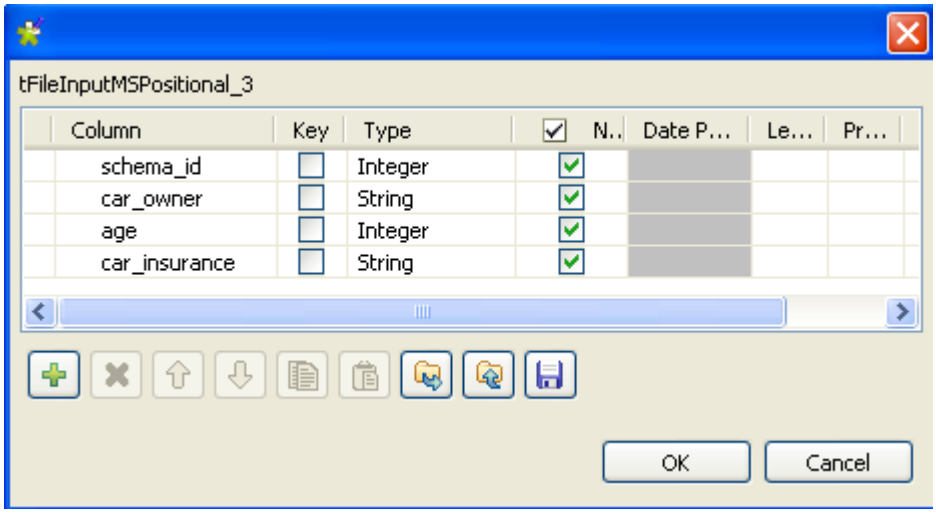
6. Enter the schema name and click **OK**.

   The schema name appears in the cell and the schema editor opens.



7. Define the schema *car_owner*, which has four columns: *schema_id*, *car_make*, *owner* and *age*.

8. Repeat the steps to define the schema *car_insurance*, which has four columns: *schema_id*, *car_owner*, *age* and *car_insurance*.



9. Connect **tFileInputMSPositional** to the **car_owner** component with the **Row** > **car_owner** link, and the **car_insurance** component with the **Row** > **car_insurance** link.

10. In the **Header value** column, type in the schema identifier value for the schema, *1* for the schema *car_owner* and *2* for the schema *car_insurance* in this case.

11. In the **Pattern** column, type in the length of each field in the schema, i.e. the number of characters, number, etc in each field, *1,8,10,3* for the schema *car_owner* and *1,10,3,3* for the schema *car_insurance* in this case.

12. In the **Skip from header** field, type in the number of beginning rows to skip, *2* in this case as the two rows in the beginning just describes the two schemas, instead of the values.

13. Choose **Table (print values in cells of a table)** in the **Mode** area of the components **car_owner** and **car_insurance**.

Executing the Job

1. Press **Ctrl+S** to save the Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

```
[statistics] connecting to socket on port 3754
[statistics] connected
.---------+---------+---+-------------.
|             car_insurance             |
|=--------+---------+---+------------=|
|schema_id|car_owner|age|car_insurance|
|=--------+---------+---+------------=|
|2        |John     |45 |yes          |
|2        |Mike     |50 |No           |
'---------+---------+---+-------------'

.---------+--------+-----+---.
|          car_owner           |
|=--------+--------+-----+--=|
|shcema_id|car_make|owner|age|
|=--------+--------+-----+--=|
|1        |bmw     |John |45 |
|1        |bench   |Mike |30 |
'---------+--------+-----+---'
```

The file is read row by row based on the length values defined in the **Pattern** field and output in two tables with different schemas.

tFileInputMSXML

tFileInputMSXML Properties

Component family	XML or File/Input	
Function	**tFileInputMSXML** reads and outputs multiple schema within an XML structured file.	
Purpose	**tFileInputMSXML** opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next Job components.	
Basic settings	*File Name*	Name of the file and/or the variable to be processed Related topic: see *Talend Open Studio for Big Data User Guide*.
	Root XPath query	The root of the XML tree, which the query is based on.
	Enable XPath in column "Schema XPath loop" But lose the order	Select this check box if you want to define a XPath path in the **Schema XPath loop** field of th **Outputs** array. *This option is only available with the **dom4j** generation mode. Make sure this mode is selected in the **Generation mode** list, in the **Advanced settings** tab of your component. If you use this option, the data will not be returned in order.*
	Outputs	**Schema**: define as many schemas as needed. **Schema XPath loop**: node of the XML tree or XPath path which the loop is based on. *If you want to use a XPath path in the **Schema XPath loop** field, you must select the **Enable XPath in column "Schema XPath loop" but lose the order** check box.* **XPath Queries**: Enter the fields to be extracted from the structured input. **Create empty row**: select the check boxes of the schemas where you want to create empty rows.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Trim all column*	Select this check box to remove leading and trailing whitespaces from defined columns.
	Validate date	Select this check box to check the date format strictly against the input schema.
	Ignore DTD file	Select this check box to ignore the DTD file indicated in the XML file being processed.
	Generation mode	Select the appropriate generation mode according to your memory availability. The available modes are: • **Slow and memory-consuming (Dom4j)** This option allows you to use dom4j to process the XML files of high complexity. • **Fast with low memory consumption (SAX)**
	Encoding	Select the encoding type from the list or select **CUSTOM** and define it manually. This field is compulsory for DB data handling.

	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Limitation	n/a	

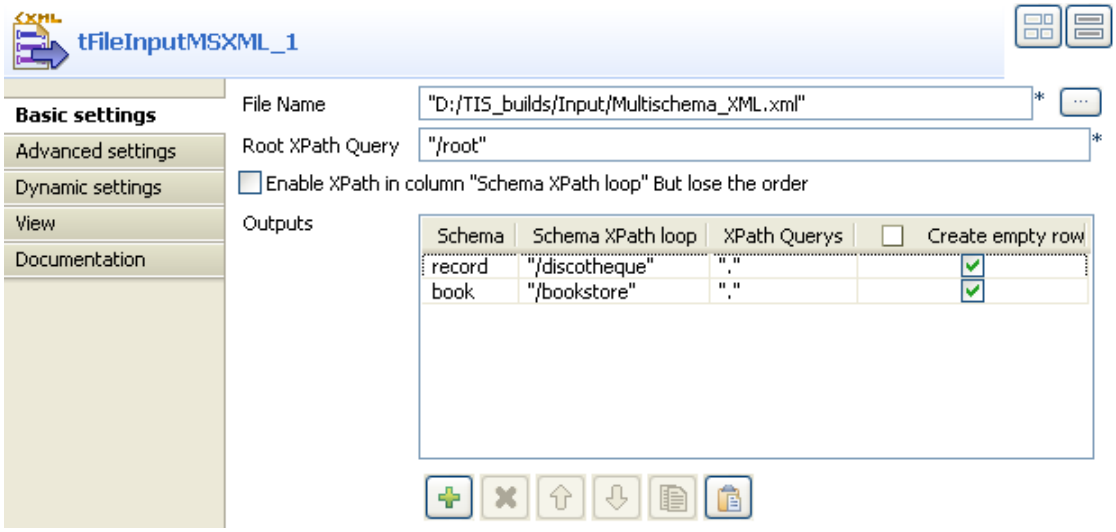
Scenario: Reading a multi structure XML file

The following scenario creates a Java Job which aims at reading a multi schema XML file and displaying data structures on the **Run Job** console.

The XML file processed in this example looks like the following:

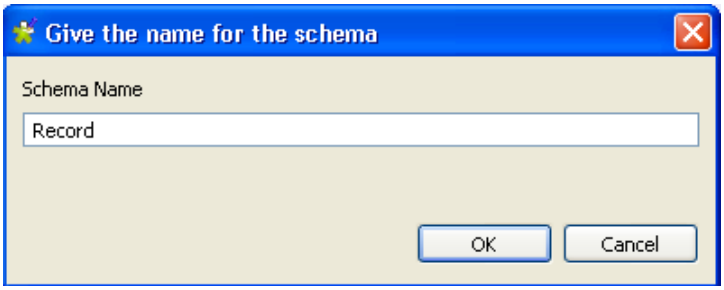
```
- <root>
  - <discotheque>
      <record>Something About Her</record>
      <record>Fall Into Me</record>
      <record>Once In A Liftime</record>
    </discotheque>
  - <bookstore>
      <book>Another Try</book>
      <book>By Myself</book>
      <book>null</book>
    </bookstore>
  </root>
```

1. Drop a **tFileInputMSXML** and two **tLogRow** components from the **Palette** onto the design workspace.
2. Double-click **tFileInputMSXML** to open the component **Basic settings** view.



3. Browse to the XML file you want to process.
4. In the **Root XPath query** field, enter the root of the XML tree, which the query will be based on.
5. Select the **Enable XPath in column "Schema XPath loop" but lose the order** check box if you want to define a XPath path in the **Schema XPath loop** field, in the **Outputs** array. In this scenario, we do not use this option.
6. Click the plus button to add lines in the **Outputs** table where you can define the output schema, two lines in this scenario: *record* and *book*.

7. In the **Outputs** table, click in the **Schema** cell and then click a three-dot button to display a dialog box where you can define the schema name.



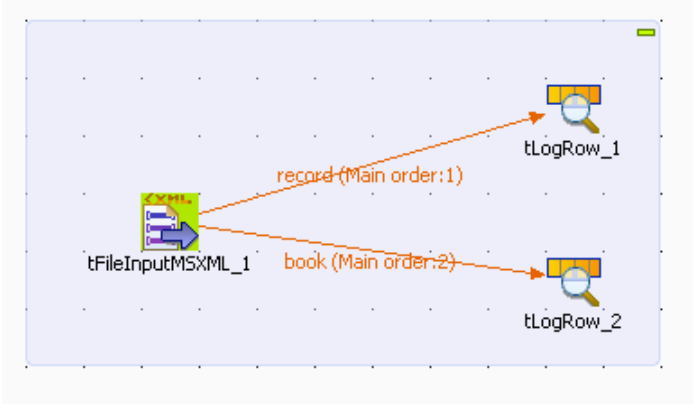
8. Enter a name for the output schema and click **OK** to close the dialog box.

   The **tFileInputMSXML** schema editor displays.

9. Define the schema you previously defined in the **Outputs** table.

10. Do the same for all the output schemas you want to define.

11. In the design workspace, right-click **tFileInputMSXML** and connect it to **tLogRow1**, and **tLogRow2** using the **record** and **book** links respectively.



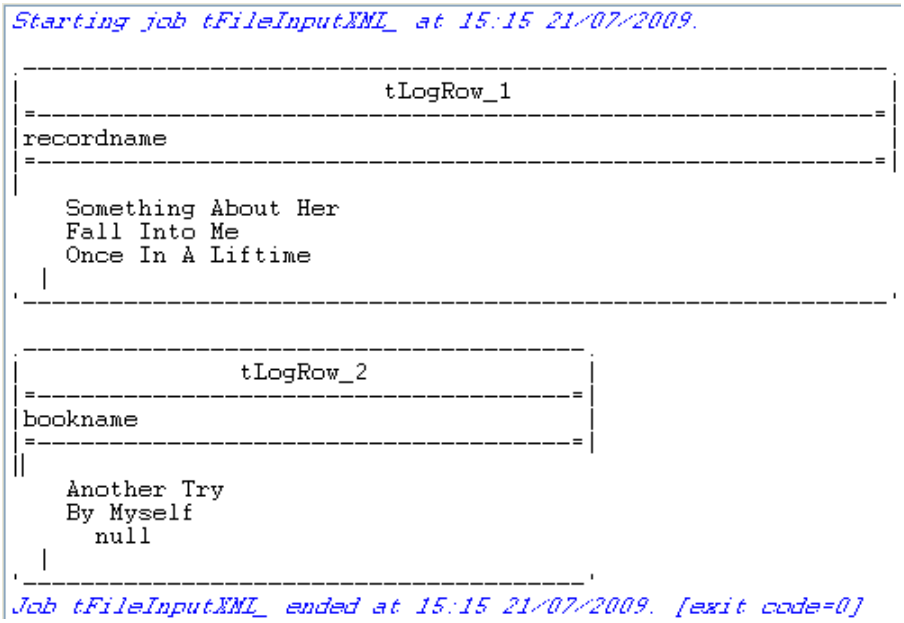
12. In the **Basic settings** view and in the **Schema XPath loop** cell, enter the node of the XML tree, which the loop is based on.

13. In the **XPath Queries** cell, enter the fields to be extracted from the structured XML input.

14. Select the check boxes next to schemas' names where you want to create empty rows.

15. Save your Job and press **F6** to execute it. The defined schemas are extracted from the multi schema XML structured file and displayed on the console.

    The multi schema XML file is read row by row and the extracted fields are displayed on the **Run Job** console as defined.


Starting job tFileInputXML_ at 15:15 21/07/2009.
tLogRow_1
recordname
Something About Her
Fall Into Me
Once In A Liftime
tLogRow_2
bookname
Another Try
By Myself
null
Job tFileInputXML_ ended at 15:15 21/07/2009. [exit code=0]

tFileInputPositional

tFileInputPositional properties

Component family	File/Input	
Function	**tFileInputPositional** reads a given file row by row and extracts fields based on a pattern.	
Purpose	This component opens a file and reads it row by row to split them up into fields then sends fields as defined in the schema to the next Job component, via a Row link.	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	File Name/Stream	**File name**: Name and path of the file to be processed. **Stream:** The data flow to be processed. The data must be added to the flow in order for **tFileInputPositional** to fetch these data via the corresponding representative variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component, for example, the *INPUT_STREAM* variable of **tFileFetch**; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl +Space**) to fill the current field on condition that this variable has been properly defined. Related topic to the available variables: see *Talend Open Studio for Big Data User Guide*Related scenario to the input stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Use byte length as the cardinality	Select this check box to enable the support of double-byte character to this component. JDK 1.6 is required for this feature.
	Customize	Select this check box to customize the data format of the positional file and define the table columns: **Column**: Select the column you want to customize. **Size**: Enter the column size. **Padding char**: Type in between inverted commas the padding character used in order for it to be removed from the field. A space by default. **Alignment**: Select the appropriate alignment parameter.
	Pattern	Length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this field are consistent with the schema defined.
	Skip empty rows	Select this check box to skip empty rows.
	Uncompress as zip file	Select this check box to uncompress the input file.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete

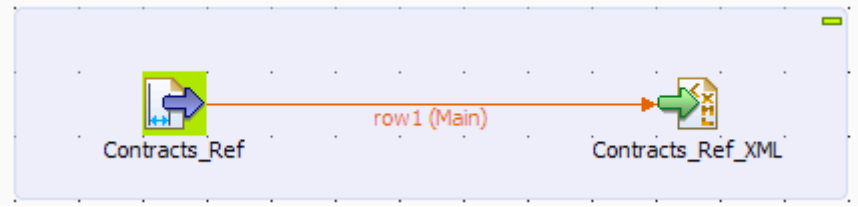
		the process for error-free rows. If needed, you can collect the rows on error using a **Row** > **Reject** link.
	Header	Number of rows to be skipped in the beginning of file
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Needed to process rows longer than 100 000 characters*	Select this check box if the rows to be processed in the input file are longer than 100 000 characters.
	Advanced separator (for numbers)	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Trim all column	Select this check box to remove leading and trailing whitespaces from defined columns.
	Validate date	Select this check box to check the date format strictly against the input schema.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a file and separate fields using a position separator value. You can also create a rejection flow using a **Row > Reject** link to filter the data which does not correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	

Scenario 1: From Positional to XML file

The following scenario describes a two-component Job, which aims at reading data from an input file that contains contract numbers, customer references, and insurance numbers as shown below, and outputting the selected data (according to the data position) into an XML file.

```
Contract        CustomerRef     InsuranceNr
00001           8200            50330
00001           8201            50331
00002           8202            50332
00002           8203            50333
```

Dropping and linking components



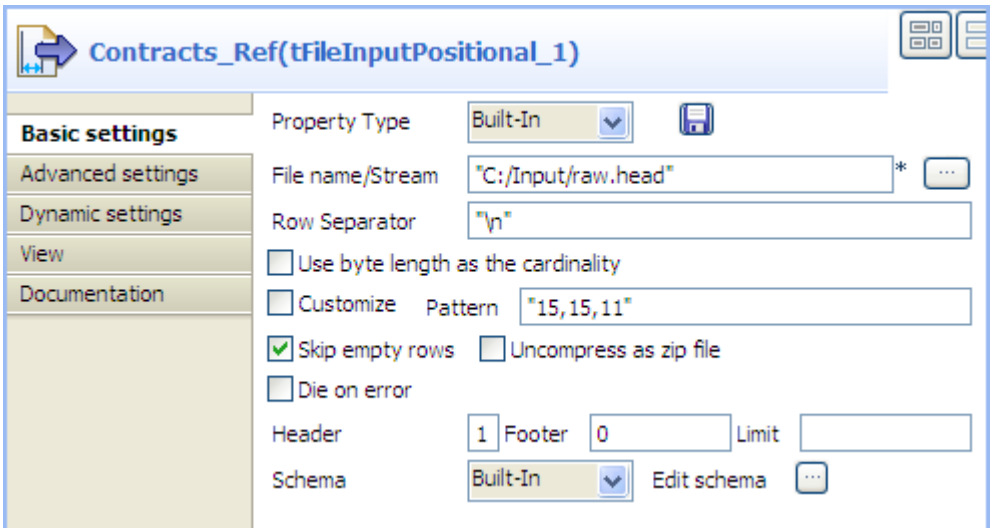
1. Drop a **tFileInputPositional** component from the **Palette** to the design workspace.

2. Drop a **tFileOutputXML** component as well. This file is meant to receive the references in a structured way.

3. Right-click the **tFileInputPositional** component and select **Row** > **Main**. Then drag it onto the **tFileOutputXML** component and release when the plug symbol shows up.

Configuring data input

1. Double-click the **tFileInputPositional** component to show its **Basic settings** view and define its properties.



2. Define the Job Property type if needed. For this scenario, we use the built-in Property type.

   As opposed to the Repository, this means that the Property type is set for this station only.

3. Fill in a path to the input file in the **File Name** field. This field is mandatory.

4. Define the **Row separator** identifying the end of a row if needed, by default, a carriage return.

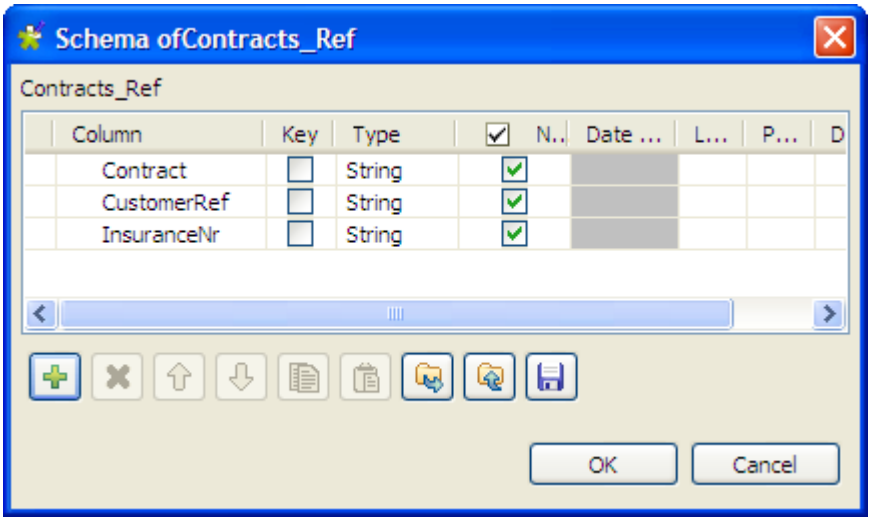
5. If required, select the **Use byte length as the cardinality** check box to enable the support of double-byte character.

6. Define the **Pattern** to delimit fields in a row. The pattern is a series of length values corresponding to the values of your input files. The values should be entered between quotes, and separated by a comma. Make sure the values you enter match the schema defined.

7. Fill in the **Header**, **Footer** and **Limit** fields according to your input file structure and your need. In this scenario, we only need to skip the first row when reading the input file. To do this, fill the **Header** field with *1* and leave the other fields as they are.

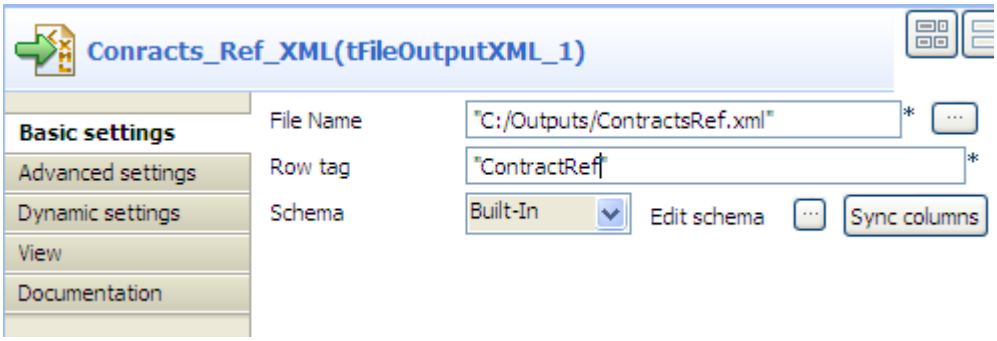
8. Next to **Schema**, select **Repository** if the input schema is stored in the Repository. In this use case, we use a **Built-In** input schema to define the data to pass on to the **tFileOutputXML** component.

9. You can load and/or edit the schema via the **Edit Schema** function. For this schema, define three columns, respectively *Contract*, *CustomerRef* and *InsuranceNr* matching the structure of the input file. Then, click **OK** to close the **[Schema]** dialog box and propagate the changes.

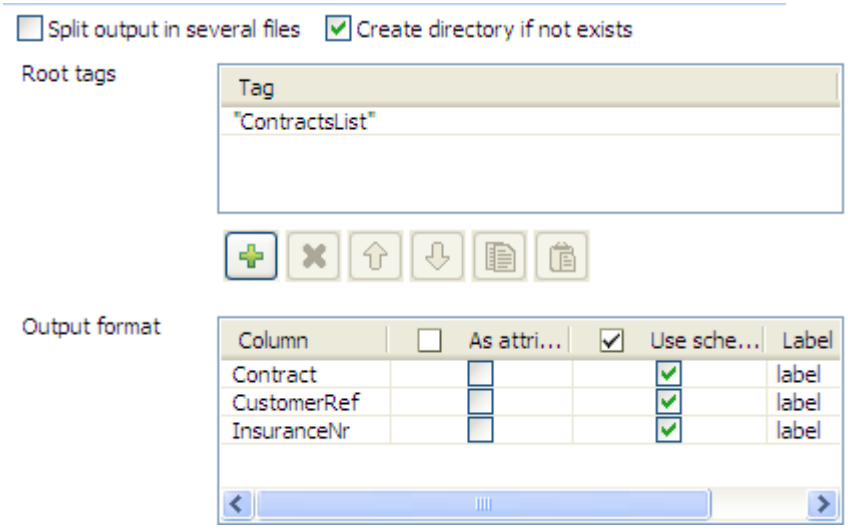


Configuring data output

1. Double-click **tFileOutputXML** to show its **Basic settings** view.



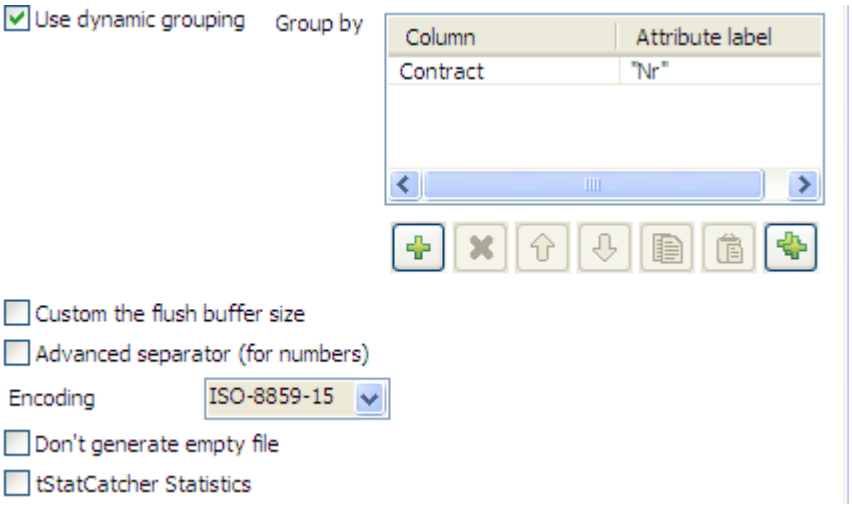
2. Enter the XML output file path.
3. Define the row tag that will wrap each row of data, in this use case *ContractRef*.
4. Click the three-dot button next to **Edit schema** to view the data structure, and click **Sync columns** to retrieve the data structure from the input component if needed.
5. Switch to the **Advanced settings** tab view to define other settings for the XML output.



6. Click the plus button to add a line in the **Root tags** table, and enter a root tag (or more) to wrap the XML output structure, in this case *ContractsList*.

7. Define parameters in the **Output format** table if needed. For example, select the **As attribute** check box for a column if you want to use its name and value as an attribute for the parent XML element, clear the **Use schema column name** check box for a column to reuse the column label from the input schema as the tag label. In this use case, we keep all the default output format settings as they are.

8. To group output rows according to the contract number, select the **Use dynamic grouping** check box, add a line in the **Group by** table, select **Contract** from the **Column** list field, and enter an attribute for it in the **Attribute label** field.



9. Leave all the other parameters as they are.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job to ensure that all the configured parameters take effect.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The file is read row by row based on the length values defined in the **Pattern** field and output as an XML file as defined in the output settings. You can open it using any standard XML editor.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<ContractsList>
    <Contract Nr="00001">
        <ContractRef>
            <CustomerRef>8200</CustomerRef>
            <InsuranceNr>50330</InsuranceNr>
        </ContractRef>
        <ContractRef>
            <CustomerRef>8201</CustomerRef>
            <InsuranceNr>50331</InsuranceNr>
        </ContractRef>
    </Contract>
    <Contract Nr="00002">
        <ContractRef>
            <CustomerRef>8202</CustomerRef>
            <InsuranceNr>50332</InsuranceNr>
        </ContractRef>
        <ContractRef>
            <CustomerRef>8203</CustomerRef>
            <InsuranceNr>50333</InsuranceNr>
        </ContractRef>
    </Contract>
</ContractsList>
```

Scenario 2: Handling a positional file based on a dynamic schema

This scenario describes a four-component Job that reads data from a positional file, writes the data to another positional file, and replaces the padding characters with space. The schema column details are not defined in the positional file components; instead, they leverages a reusable dynamic schema. The input file used in this scenario is as follows:

```
id----name--------city--------
1-----Andrews-----Paris-------
2-----Mark--------London------
3-----Marie-------Paris-------
4-----Michael-----Washington--
```

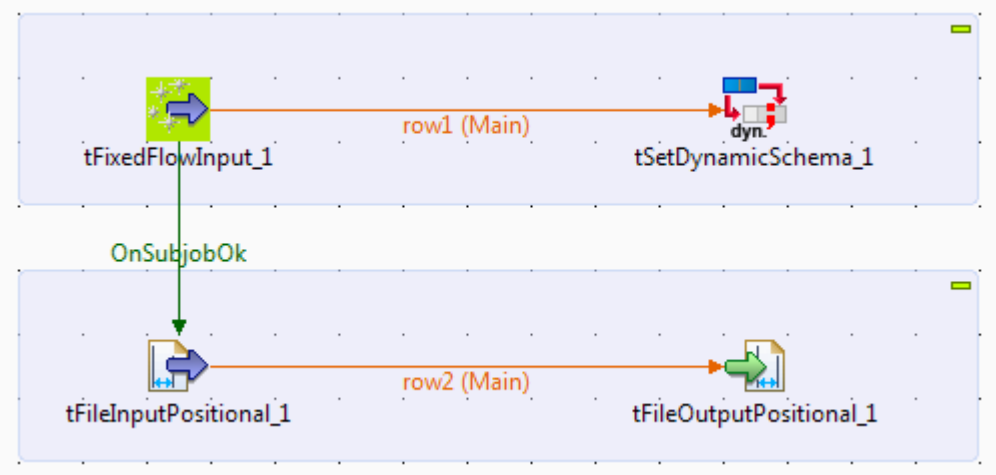
Dropping and linking components

1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput**, **tSetDynamicSchema**, **tFileInputPositional**, and **tFileOutputPositional**.

2. Connect the **tFixedFlowInput** component to the **tSetDynamicSchema** using a **Row** > **Main** connection to form a subjob. This subjob will define a reusable dynamic schema.

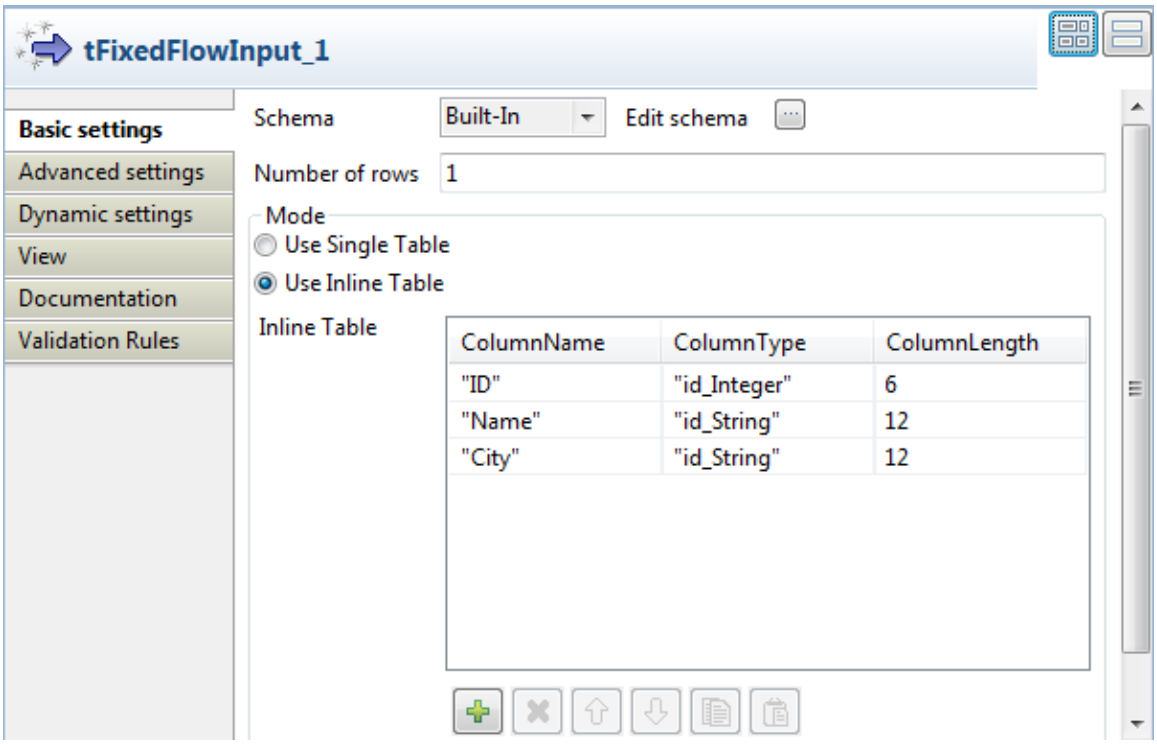
3. Connect the **tFileInputPositional** component to the **tFileOutputPositional** component using a **Row** > **Main** connection to form another subjob. This subjob will read data from the input positional file and write the data to another positional file based on the dynamic schema set in the previous subjob.

4. Connect the **tFixedFlowInput** component to the **tFileInputPositional** component using a **Trigger** > **On Subjob Ok** connection to link the two subjobs together.

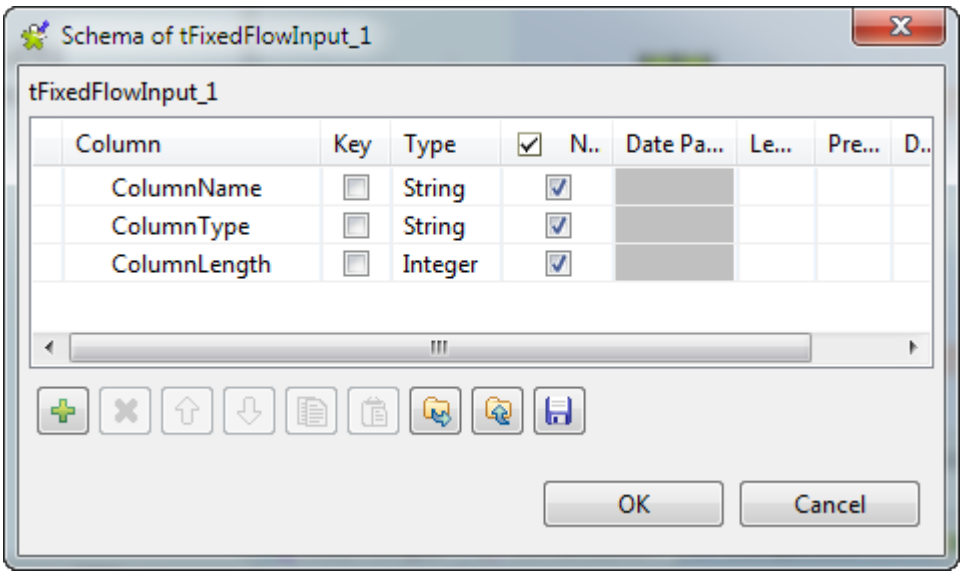


Configuring the first subjob: creating a dynamic schema

1. Double-click the **tFixedFlowInput** component to show its **Basic settings** view and define its properties.



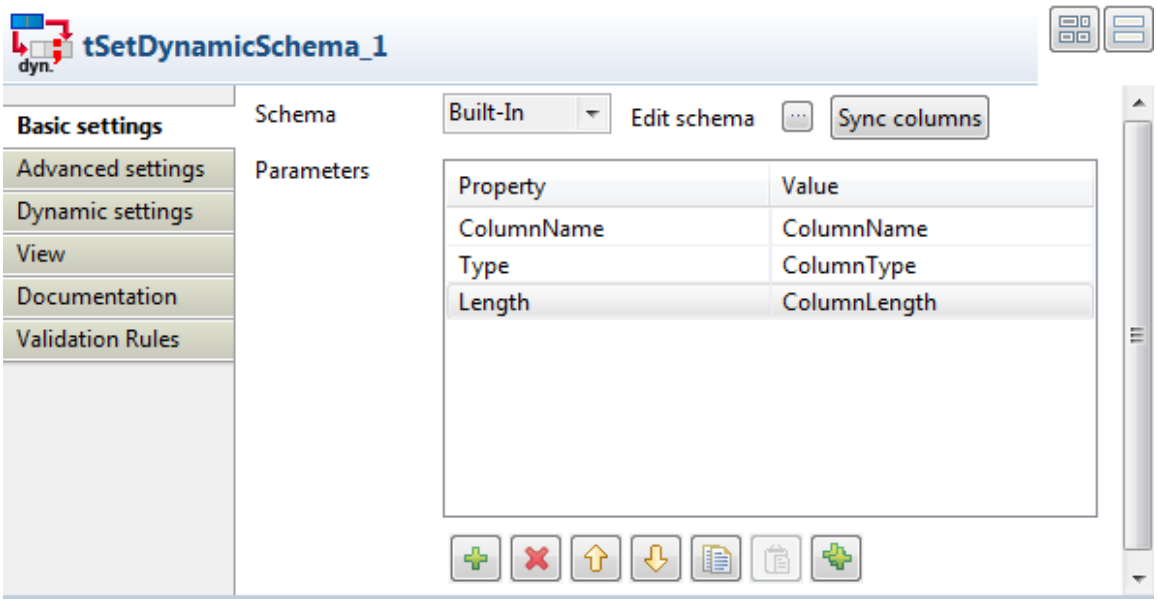
2. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box.



3. Click the **[+]** button to add three columns: *ColumnName*, *ColumnType*, and *ColumnLength*, and set their types to **String**, **String**, and **Integer** respectively to define the minimum properties required for a positional file schema. Then, click **OK** to close the dialog box.

4. Select the **Use Inline Table** option, click the **[+]** button three times to add three lines, give them a name in the **ColumnName** field, according to the actual columns of the input file to read: *ID*, *Name*, and *City*, set their types in the corresponding **ColumnType** field: *id_Interger* for column *ID*, and *id_String* for columns *Name* and *City*, and set the length values of the columns in the corresponding **ColumnLength** field. Note that the column names you give in this table will compose the header of the output file.

5. Double-click the **tSetDynamicSchema** component to open its **Basic settings** view.



6. Click **Sync columns** to ensure that the schema structure is properly retrieved from the preceding component.

7. Under the **Parameters** table, click the **[+]** button to add three lines in the table.

8. Click in the **Property** field for each line, and select **ColumnName**, **Type**, and **Length** respectively.

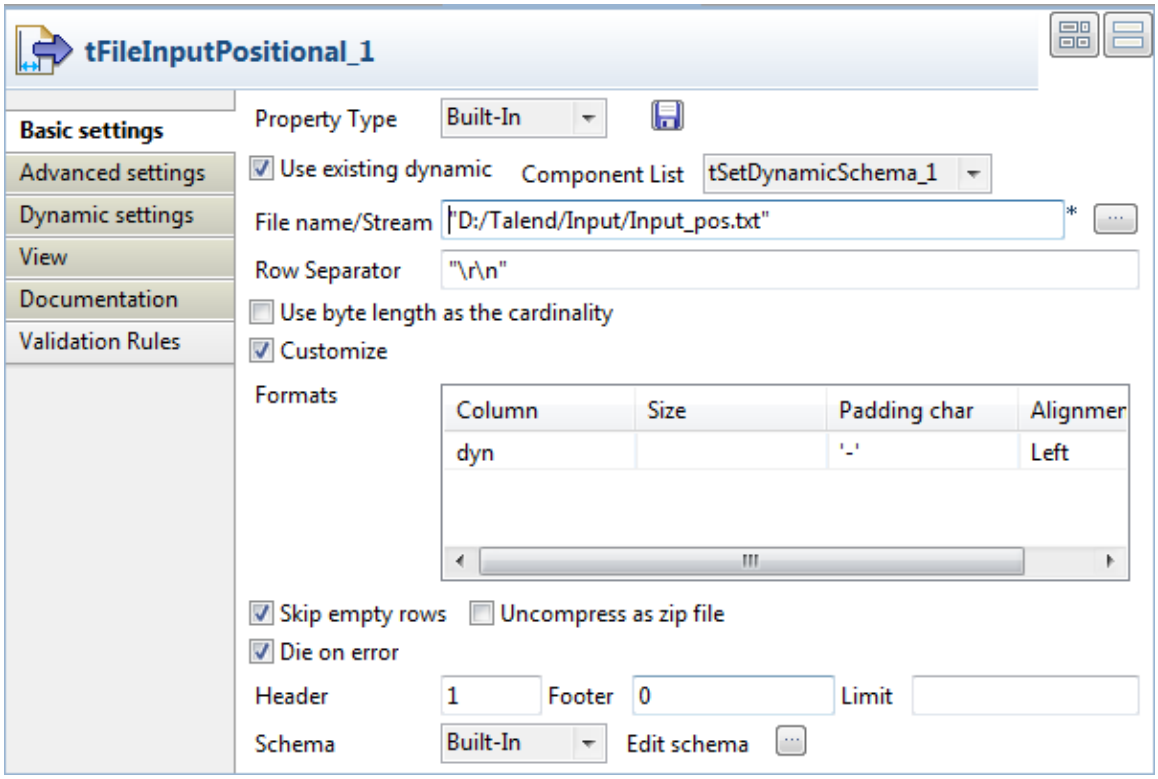
9. Click in the **Value** field for each line, and select **ColumnName**, **ColumnType**, and **ColumnLength** respectively.

Now, with the values set in the inline table of the **tFixedFlowInput** component retrieved, the following data structure is defined in the dynamic schema:

Column Name	Type	Length
ID	Integer	6
Name	String	12
City	String	12

Configuring the second subjob: reading and writing positional data

1. Double-click the **tFileInputPositional** component to open its **Basic settings** view.



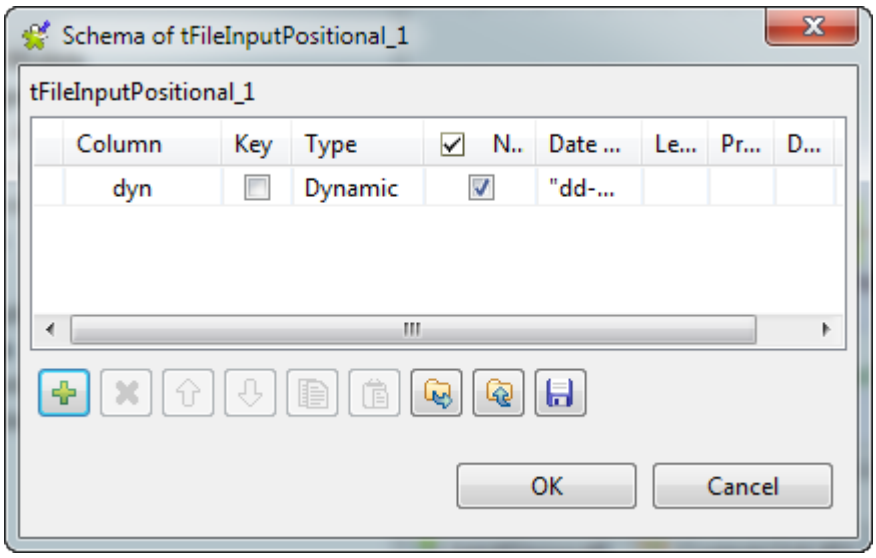
*The dynamic schema feature is only supported in **Built-In** mode and requires the input file to have a header row.*

2. Select the **Use existing dynamic** check box, and in from the **Component List** that appears, select the **tSetDynamicSchema** component you use to create the dynamic schema. In this use case, only one **tSetDynamicSchema** component is used, so it is automatically selected.

3. In the **File name/Stream** field, enter the path to the input positional file, or browse to the file path by clicking the **[...]** button.

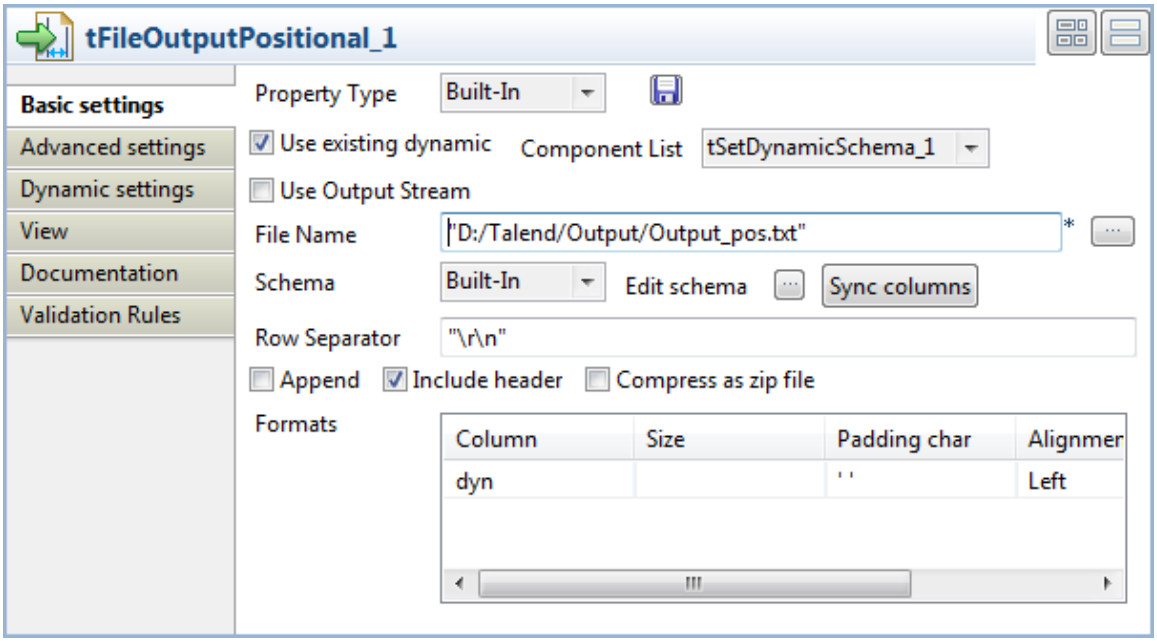
4. Fill in the **Header**, **Footer** and **Limit** fields according to your input file structure and your need. In this scenario, we only need to skip the first row when reading the input file. To do this, fill the **Header** field with *1* and leave the other fields as they are.

5. Click the **[...]** button next to **Edit schema** to open the Schema dialog box, define only one column, *dyn* in this example, and select **Dynamic** from the **Type** list. Then, click **OK** to close the **[Schema]** dialog box and propagate the changes.



6. Select the **Customize** check box, enter `'-'` in the **Padding char** field, and keep the other settings as they are.

7. Double-click the **tFileOutputPositional** component to open its **Basic settings** view.



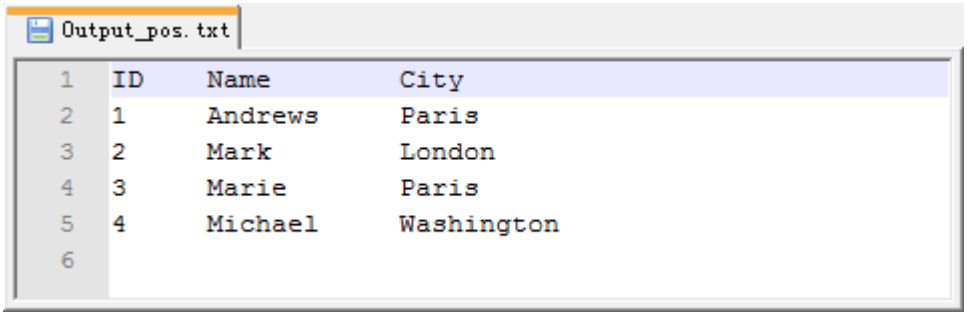
8. Select the **Use existing dynamic** check box, specify the output file path, and select the **Include header** check box.

9. In the **Padding char** field, enter `' '` so that the padding characters will be replaced with space in the output file.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job to ensure that all the configured parameters take effect.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.



The data is read from the input positional file and written into the output positional file, with the padding characters replaced by space.

tFileInputProperties

tFileInputProperties properties

Component family	File/Input	
Function	**tFileInputProperties** reads a text file row by row and extracts the fields.	
Purpose	**tFileInputProperties** opens a text file and reads it row by row then separates the fields according to the model key = value.	
Basic settings	*Schema* and *Edit Schema*	In Talend Open Studio for Big Data, the Property type is always built-in. The schema is either built-in or remotely stored in the Repository but for this component, the schema is read-only. It is made of two column, *Key* and *Value*, corresponding to the parameter name and the parameter value to be copied.
	File format	Select from the list your file format, either: **.properties** or **.ini**.
		.properties: data in the configuration file is written in two lines and structured according to the following way: key = value.
		.ini: data in the configuration file is written in two lines and structured according to the following way: key = value and re-grouped in sections. **Section Name**: enter the section name on which the iteration is based.
	File Name	Name or path to the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a text file and separate data according to the structure key = value.	

Scenario: Reading and matching the keys and the values of different .properties files and outputting the results in a glossary

This four-component Java Job reads two .properties files, one in French and the other in English. The data in the two input files is mapped to output a glossary matching the English and French terms.

The two input files used in this scenario hold localization strings for the **tMysqlInput** component in *Talend Open Studio for Big Data*.

tMysqlInput_messages_en.properties

```
DBD-ODBC.INFO=Required for ODBC-
DBD-Oracle.INFO=Required for Ora
DBD-Pg.INFO=Required for Postgre
DBD-mysql.INFO=Required for MySQ
DBNAME.NAME=Database
DBTABLE.NAME=Table Name
ENABLE_STREAM.NAME=Enable strea
ENCODING.NAME=Encoding
HELP=org.talend.help.tMysqlInput
HOST.NAME=Host
LONG_NAME=Reads a MySQL table a
NB_LINE.NAME=Number of line
NULL_CHAR.NAME=Null Char
PASS.NAME=Password
PORT.NAME=Port
PROPERTIES.NAME=Additional JDBC
QUERY.NAME=Query
QUERYSTORE.NAME=Query Type
```

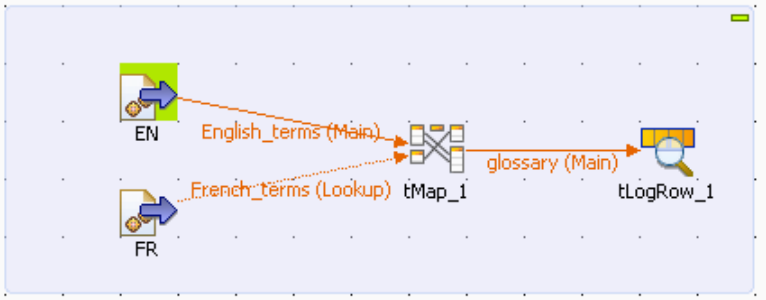
tMysqlInput_messages_fr.properties

```
DBD-ODBC.INFO=Requis pour les cor
DBD-Oracle.INFO=Requis par Oracle
DBD-Pg.INFO=Requis par PostgreSQL
DBD-mysql.INFO=Requis par MySQL
DBNAME.NAME=Base de données
DBTABLE.NAME=Nom de table
ENABLE_STREAM.NAME=Activer la dif
ENCODING.NAME=Encodage
HELP=org.talend.help.tMysqlInput
HOST.NAME=Hôte
LONG_NAME=Lit une table MySQL et
NB_LINE.NAME=Nombre de ligne
NULL_CHAR.NAME=Caractère Null
PASS.NAME=Mot de passe
PORT.NAME=Port
PROPERTIES.NAME=Paramètres JDBC
QUERY.NAME=Requête
```

Dropping and linking the components

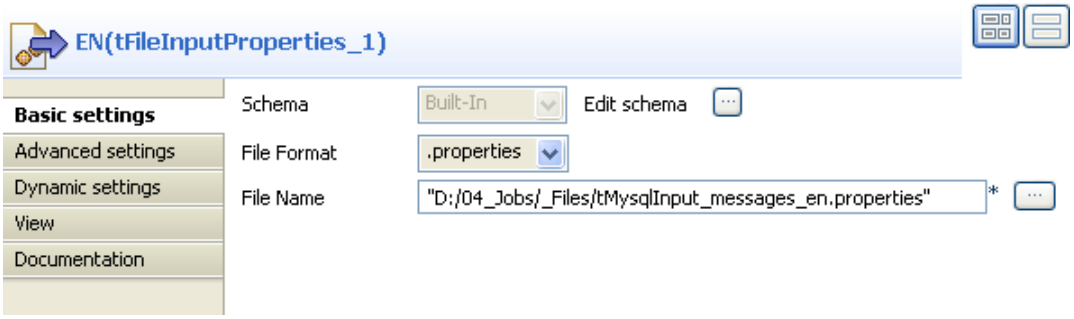
1. Drop the following components from the **Palette** onto the design workspace: **tFileInputProperties** (x2), **tMap**, and **tLogRow**.

2. Connect the component together using **Row** > **Main** links. The second properties file, *FR*, is used as a lookup flow.



Configuring the components

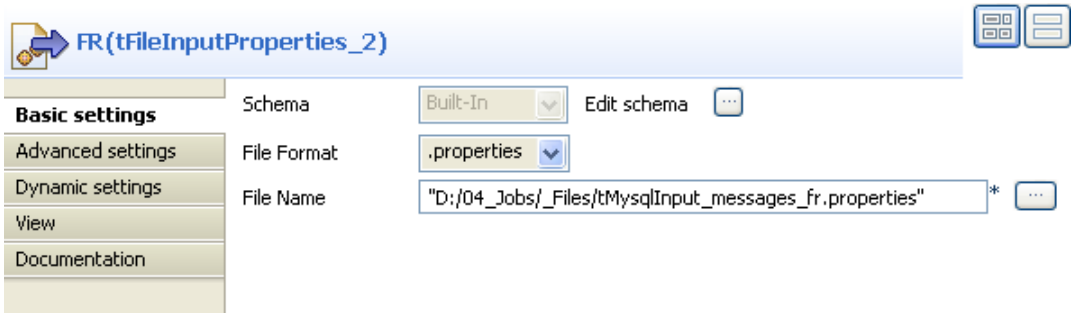
1. Double-click the first **tFileInputProperties** component to open its **Basic settings** view and define its properties.



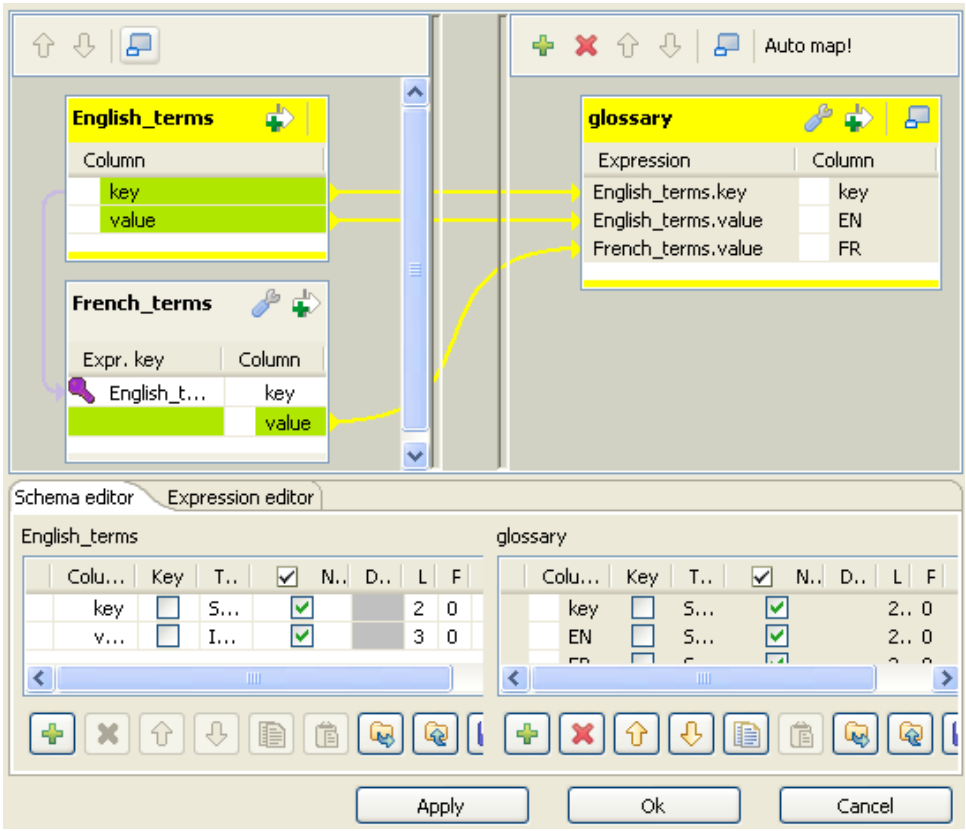
2. In the **File Format** field, select your file format.

3. In the **File Name** field, click the three-dot button and browse to the input .properties file you want to use.

4. Do the same with the second **tFileInputProperties** and browse to the French properties file this time.



5. Double-click the **tMap** component to open the tMap editor.



6. Select all columns from the *English_terms* table and drop them to the output table.

   Select the *key* column from the *English_terms* table and drop it to the *key* column in the *French_terms* table.

7. In the **glossary** table in the lower right corner of the tMap editor, rename the *value* field as *EN* because it will hold the values of the English file.

8. Click the plus button to add a line to the **glossary** table and rename it as *FR*.

9. In the **Length** field, set the maximum length to *255*.

10. In the upper left corner of the tMap editor, select the *value* column in the *English_terms* table and drop it to the *FR* column in the *French_terms* table.

11. Click **OK** to validate your changes and close the editor.

12. In the design workspace, double-click **tLogRow** to display its **Basic settings** and define the component properties.

13. Click **Sync Columns** to retrieve the schema from the preceding component.

Saving and executing the Job

1. Press Ctrl+S to save your Job.

2. Press **F6** or click the **Run** button from the **Run** tab to execute it.

```
Starting job tFileInputProperties at 15:25 29/05/2009.

PORT.NAME | Port | Port
HELP | org.talend.help.tMysqlInput | org.talend.help.tMysqlInput
STRING_QUOTE.NAME | String Quote | Séparateur de chaine de
caractère
DBTABLE.NAME | Table Name | Nom de table
QUERYSTORE.NAME | Query Type | Type de requête
SQL_SYNTAX.NAME | Sql Syntax | Syntaxe SQL
TYPE.ITEM.PGSQL | PostgreSQL | PostgreSQL
TYPE.NAME | Database Driver | Pilote de base de données
TABLE.NAME | Table Name | Nom de table
ENCODING.NAME | Encoding | Encodage
QUERY.NAME | Query | Requête
DBD-ODBC.INFO | Required for ODBC-like connection | Requis pour
les connexions de type ODBC
```

The glossary displays on the console listing three columns holding: the key name in the first column, the English term in the second, and the corresponding French term in the third.

tFileInputRegex

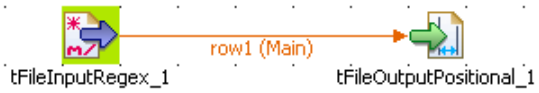
tFileInputRegex properties

Component family	File/Input	
Function	Powerful feature which can replace number of other components of the File family. Requires some advanced knowledge on regular expression syntax	
Purpose	Opens a file and reads it row by row to split them up into fields using regular expressions. Then sends fields as defined in the Schema to the next Job component, via a Row link.	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	File Name/Stream	**File name**: Name of the file and/or the variable to be processed **Stream**: Data flow to be processed. The data must be added to the flow so that it can be collected by the **tFileInputRegex** via the *INPUT_STREAM* variable in the autocompletion list (**Ctrl +Space**) Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Regex	This field can contain multiple lines. Type in your regular expressions including the subpattern matching the fields to be extracted. **Note**: Antislashes need to be doubled in regexp *Regex syntax requires double quotes.*
	Header	Number of rows to be skipped in the beginning of file
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Skip empty rows	Select this check box to skip empty rows.
	Die on error	Select this check box to stop the execution of the Job when an error occurs. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can collect the rows on error using a **Row** > **Reject** link.
Advanced settings	*Encoding*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a file and separate fields contained in this file according to the defined Regex. You can also create a rejection flow using a **Row > Reject** link to filter the data which doesn’t correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	

Limitation	n/a

Scenario: Regex to Positional file

The following scenario creates a two-component Job, reading data from an Input file using regular expression and outputting delimited data into an XML file.

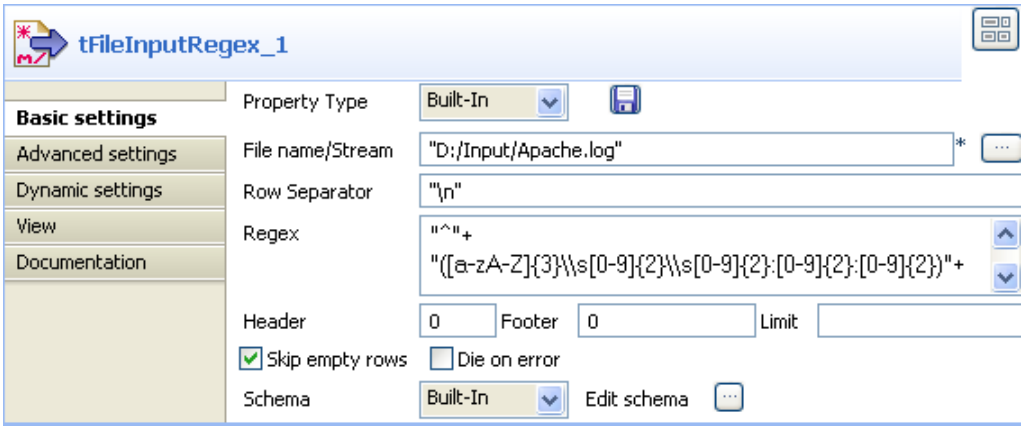


Dropping and linking the components

1. Drop a **tFileInputRegex** component from the **Palette** to the design workspace.
2. Drop a **tFileOutputPositional** component the same way.
3. Right-click on the **tFileInputRegex** component and select **Row** > **Main**. Drag this main row link onto the **tFileOutputPositional** component and release when the plug symbol displays.

Configuring the components

1. Select the **tFileInputRegex** again so the **Component** view shows up, and define the properties:

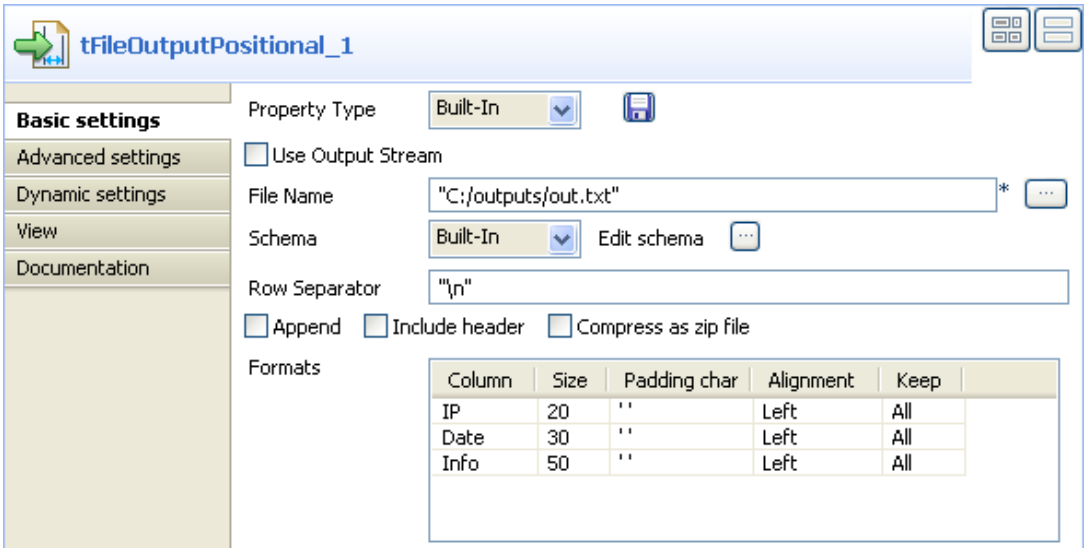


2. The Job is built-in for this scenario. Hence, the Properties are set for this station only.
3. Fill in a path to the file in **File Name** field. This field is mandatory.
4. Define the **Row separator** identifying the end of a row.
5. Then define the **Regular expression** in order to delimit fields of a row, which are to be passed on to the next component. You can type in a regular expression using Java code, and on mutiple lines if needed.

   Regex syntax requires double quotes.

6. In this expression, make sure you include all subpatterns matching the fields to be extracted.
7. In this scenario, ignore the header, footer and limit fields.
8. Select a local (**Built-in**) **Schema** to define the data to pass on to the **tFileOutputPositional** component.
9. You can load or create the schema through the **Edit Schema** function.

10. Then define the second component properties:



11. Enter the Positional file output path.

12. Enter the **Encoding** standard, the output file is encoded in. Note that, for the time being, the encoding consistency verification is not supported.

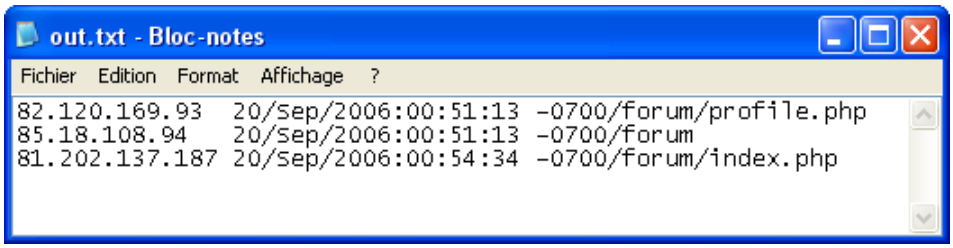
13. Select the **Schema** type. Click on **Sync columns** to automatically synchronize the schema with the Input file schema.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Now go to the **Run** tab, and click on **Run** to execute the Job.

   The file is read row by row and split up into fields based on the **Regular Expression** definition. You can open it using any standard file editor.



tFileInputXML

tFileInputXML belongs to two component families: File and XML. For more information on **tFileInputXML**, see section *tFileInputXML*.

tFileList

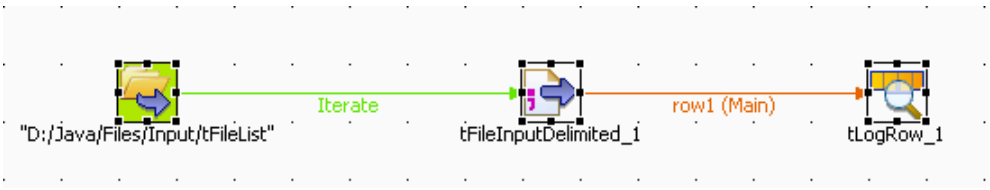
tFileList properties

Component family	File/Management	
Function	**tFileList** iterates on files or folders of a set directory.	
Purpose	**tFileList** retrieves a set of files or folders based on a filemask pattern and iterates on each unity.	
Basic settings	*Directory*	Path to the directory where the files are stored.
	FileList Type	Select the type of input you want to iterate on from the list: **Files** if the input is a set of files, **Directories** if the input is a set of directories, **Both** if the input is a set of the above two types.
	Include subdirectories	Select this check box if the selected input source type includes sub-directories.
	Case Sensitive	Set the case mode from the list to either create or not create case sensitive filter on filenames.
	Generate Error if no file found	Select this check box to generate an error message if no files or directories are found.
	Use Glob Expressions as Filemask	This check box is selected by default. It filters the results using a Global Expression (**Glob Expressions**).
	Files	Click the plus button to add as many filter lines as needed: **Filemask**: in the added filter lines, type in a filename or a filemask using special characters or regular expressions.
	Order by	The folders are listed first of all, then the files. You can choose to prioritise the folder and file order either: **By default**: alphabetical order, by folder then file; **By file name**: alphabetical order or reverese alphabetical order; **By file size**: smallest to largest or largest to smallest; **By modified date**: most recent to least recent or least recent to most recent. If ordering **by file name**, in the event of identical file names then **modified date** takes precedence. If ordering **by file size**, in the event of identical file sizes then **file name** takes precedence. If ordering by **modified date**, in the event of identical dates then **file name** takes precedence.
	Order action	Select a sort order by clicking one of the following radio buttons: **ASC**: ascending order; **DESC**: descending order;
Advanced settings	*Use Exclude Filemask*	Select this check box to enable **Exclude Filemask** field to exclude filtering condition based on file type: **Exclude Filemask**: Fill in the field with file types to be excluded from the **Filemasks** in the **Basic settings** view.

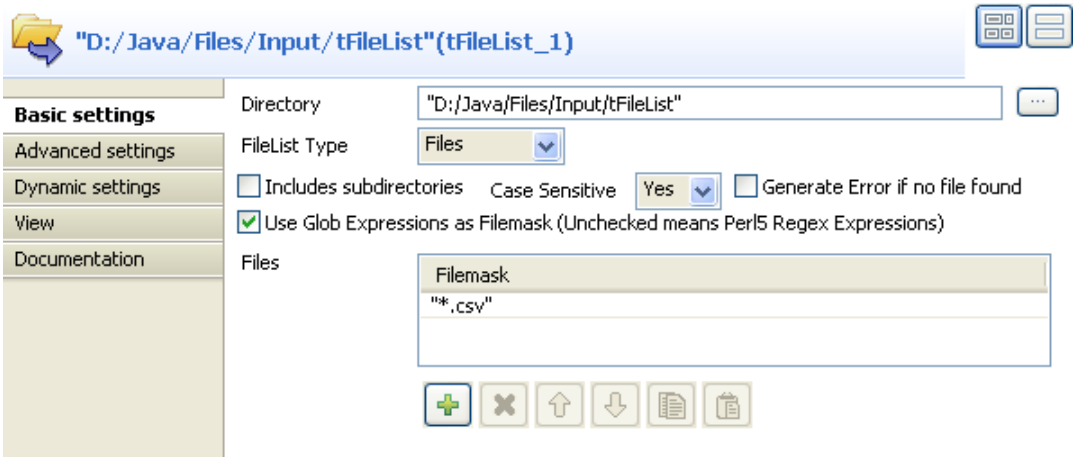
		File types in this field should be quoted with double quotation marks and seperated by comma.
	Format file path to slash(/) style(useful on Windows)	Select this check box to format the file path to slash(/) style which is useful on Windows.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tFileList** provides a list of files or folders from a defined directory on which it iterates	
Global Variables		**Current File Name**: Indicates the current file name. This is available as a **Flow** variable. Returns a string. **Current File Name with Path***:* Indicates the current file name as well as the path to the file. This is available as a **Flow** variable. Returns a string. **Current File Extension:** Indicates the extension of the current file. This is available as a **Flow** variable. Returns a string. **Current File Directory**: Indicates the access path to the folder or subfolder in which the current file is stored. This is available as a **Flow** variable. Returns a string. **Number of files**: Indicates the number of files iterated upon so far. This is available as a **Flow** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Iterating on a file directory

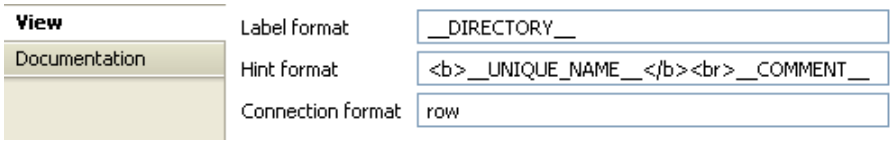
The following scenario creates a three-component Job, which aims at listing files from a defined directory, reading each file by iteration, selecting delimited data and displaying the output in the **Run** log console.



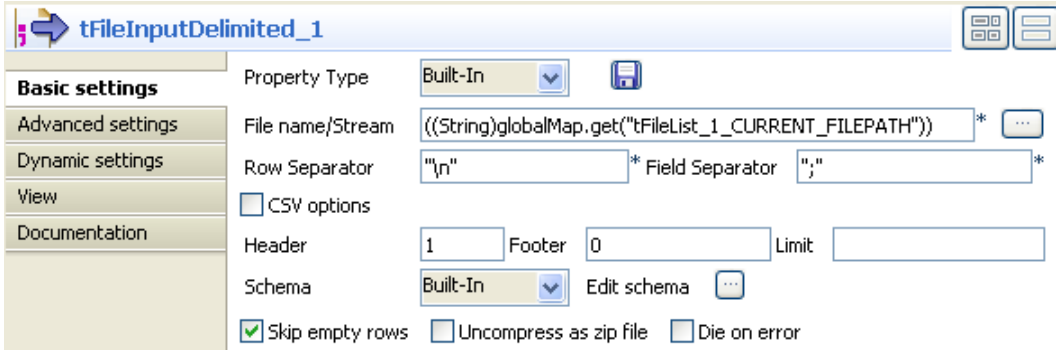
1. Drop the following components from the **Palette** to the design workspace: **tFileList**, **tFileInputDelimited**, and **tLogRow**.
2. Right-click on the **tFileList** component, and pull an **Iterate** connection to the **tFileInputDelimited** component. Then pull a **Main** row from the **tFileInputDelimited** to the **tLogRow** component.
3. Double-click **tFileList** to display its **Basic settings** view and define its properties.



4. Browse to the **Directory** that holds the files you want to process. To display the path on the Job itself, use the label (__DIRECTORY__) that shows up when you put the pointer anywhere in the **Directory** field. Type in this lable in the **Label Format** field you can find if you click the **View** tab in the **Basic settings** view.



5. In the **Basic settings** view and from the **FileList Type** list, select the source type you want to process, **Files** in this example.
6. In the **Case sensitive** list, select a case mode, **Yes** in this example to create case sensitive filter on file names.
7. Keep the **Use Glob Expressions as Filemask** check box selected if you want to use global expressions to filter files.
8. In the **Filemask** field, define a file mask, use special characters if need be.
9. Double-click **tFileInputDelimited** to display its **Basic settings** view and set its properties.



10. Enter the **File Name** field using a variable containing the current filename path, as you filled in the **Basic settings** of **tFileList**. Press **Ctrl+Space bar** to access the autocomplete list of variables.

11. Select the global variable `((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))`. This way, all files in the input directory can be processed.

12. Fill in all other fields as detailed in the **tFileInputDelimited** section. Related topic: section *tMDMInput properties*.

13. Select the last component, **tLogRow**, to display its **Basic settings** view and fill in the separator to be used to distinguish field content displayed on the console. Related topic: section *tLogRow*.

```
Starting job test at 17:16 21/09/2009.

.--+-----------------------------+----------------------+-------.
|                            tLogRow_1                            |
|=-+-----------------------------+----------------------+------=|
|id|CustomerName                 |CustomerAddress       |idState|
|=-+-----------------------------+----------------------+------=|
|1 |Griffith Paving and Sealcoatin|talend@apres91        |7      |
|2 |Bill's Dive Shop             |511 Maple Ave.  Apt. 1B|35    |
|3 |Childress Child Day Care     |662 Lyons Circle      |1      |
|4 |Facelift Kitchen and Bath    |220 Vine Ave.         |41     |
|5 |Terrinni & Son Auto and Truck|770 Exmoor Rd.        |5      |
|6 |Kermit the Pet Shop          |1860 Parkside Ln.     |28     |
|7 |Tub's Furniture Store        |807 Old Trail Rd.     |15     |
|8 |Toggle & Myerson Ltd         |618 Sheriden rd.      |9      |
|9 |Childress Child Day Care     |788 Tennyson Ave.     |12     |
|10|Elle Hypnosis and Therapy Cent|2032 Northbrook Ct.   |1      |
'--+-----------------------------+----------------------+-------'
.--+-----------------------------+----------------------+-------.
|                            tLogRow_1                            |
|=-+-----------------------------+----------------------+------=|
|id|CustomerName                 |CustomerAddress       |idState|
|=-+-----------------------------+----------------------+------=|
|1 |Glenwood Credit Union        |511 Maple Ave.  Apt. 1B|46    |
|2 |Gourmet the Frog             |788 Tennyson Ave.     |1      |
|3 |Acturial Enterprises Ltd.    |3385 University Ave.  |34     |
|4 |Salt & Pepper Catering Service|965 Marion Place Apt. 65C|44  |
|5 |Rythmics Ltd.                |1875 Roger Williams Ave.|22   |
|6 |Parkway Auto Body            |1859 Green Bay Rd.    |27     |
|7 |Futoons Cartoons Emporium    |1486 Old Deerfield Rd.|24     |
|8 |Glenn Oaks Office Supplies   |1882 St. Johns        |10     |
|9 |Garfield Appliance Service   |3150 Skokie Valley Rd.|33     |
|10|New Dehli Auto Exchange      |1957 Huntington Ave.  |49     |
'--+-----------------------------+----------------------+-------'
Job test ended at 17:16 21/09/2009. [exit code=0]
```

The Job iterates on the defined directory, and reads all included files. Then delimited data is passed on to the last component which displays it on the console.

For other scenarios using **tFileList**, see section *tFileCopy*.

tFileOutputARFF

tFileOutputARFF properties

Component family	File/Output	
Function	**tFileOutputARFF** outputs data to an ARFF file.	
Purpose	This component writes an ARFF file that holds data organized according to the defined schema.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	File name	Name or path to the output file and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Attribute Define	Displays the schema you defined in the **[Edit schema** dialog box. **Column**: Name of the column. **Type**: Data type. **Pattern**: Enter the data model (pattern), if necessary.
	Relation	Enter the name of the relation.
	Append	Select this check box to add the new rows at the end of the file.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You can create the schema and store it locally for this component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if it does not exist.
Advanced settings	*Don't generate empty file*	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component along with a **Row** link to collect data from another component and to re-write the data to an ARFF file.	
Global Variables		The Global variables can be used as parameters in most of the fields found in the component properties view. To view these variables, place the cursor in the field and press **Ctrl + Space**. Double click the variable to populate the field. The main global variable associated with **tFileOutputARFF** is: *Number of lines*: Indicates the number of lines processed. This is available as an **After** variable
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if. Incoming links (from one component to another): **Row**: Main; Reject; Iterate.

		Trigger: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For **tFileOutputARFF** related scenario, see section *Scenario: Display the content of a ARFF file*.

tFileOutputDelimited

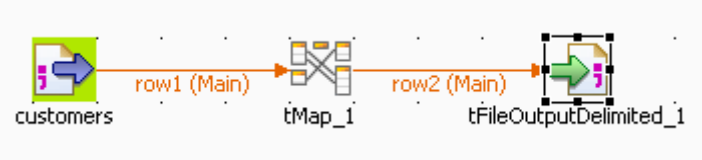
tFileOutputDelimited properties

Component family	File/Output	
Function	**tFileOutputDelimited** outputs data to a delimited file.	
Purpose	This component writes a delimited file that holds data organized according to the defined schema.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use Output Stream	Select this check box process the data flow of interest. Once you have selected it, the **Output Stream** field displays and you can type in the data flow of interest. The data flow to be processed must be added to the flow in order for this component to fetch these data via the corresponding representattive variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl +Space**) to fill the current field on condition that this variable has been properly defined. For further information about how to use a stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	File name	Name or path to the output file and/or the variable to be used. This field becomes unavailable once you have selected the **Use Output Stream** check box. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row Separator	String (ex: “\n” on Unix) to distinguish rows in the output file.
	Field Separator	Character, string or regular expression to separate fields of the output file.
	Append	Select this check box to add the new rows at the end of the file.
	Include Header	Select this check box to include the column header to the file.
	Compress as zip file	Select this check box to compress the output file in zip format.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: You can create the schema and store it locally for this component. Related topic: see *Talend Open Studio for Big Data User Guide*.

	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the **Row** connection is linked with the output component.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	CSV options	Select this check box to take into account all parameters specific to CSV files, in particular **Escape char** and **Text enclosure** parameters.
	Create directory if not exists	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Split output in several files	In case of very big output files, select this check box to divide the output delimited file into several files. **Rows in each output file**: set the number of lines in each of the output files.
	Custom the flush buffer size	Select this check box to define the number of lines to write before emptying the buffer. **Row Number**: set the number of lines to write.
	Output in row mode	Writes in row mode.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write a delimited file and separate fields using a field separator value.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario 1: Writing data in a delimited file

This scenario describes a three-component Job that extracts certain data from a file holding information about clients, *customers*, and then writes the extracted data in a delimited file.



Dropping and linking components

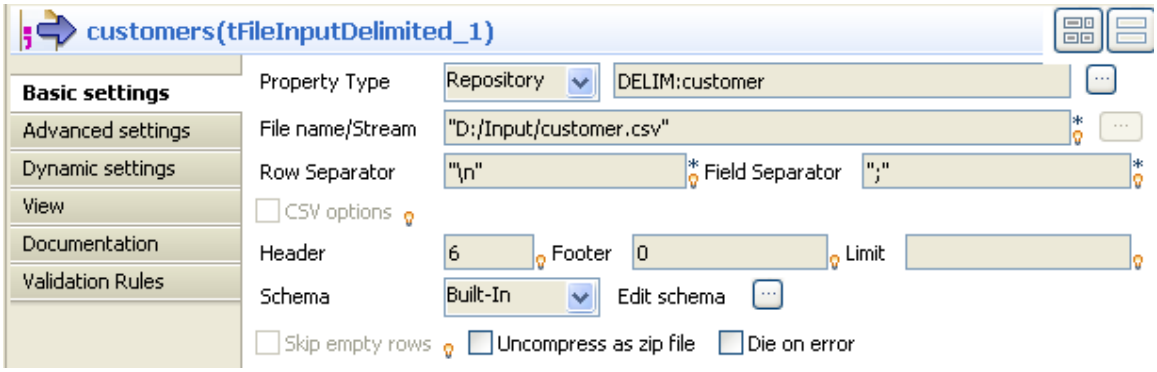
1. Drop a **tFileInputDelimited** component, a **tMap** component and a **tFileOutputDelimited** component from the **Palette** to the design workspace.

2. Link the components together using **Row** > **Main** connections.

Configuring the components

Configuring the input component

1. Double-click **tFileInputDelimited** to open its **Basic settings** view.



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Click the **[...]** button next to the **File Name** field and browse to the input file, *customer.csv* in this example.

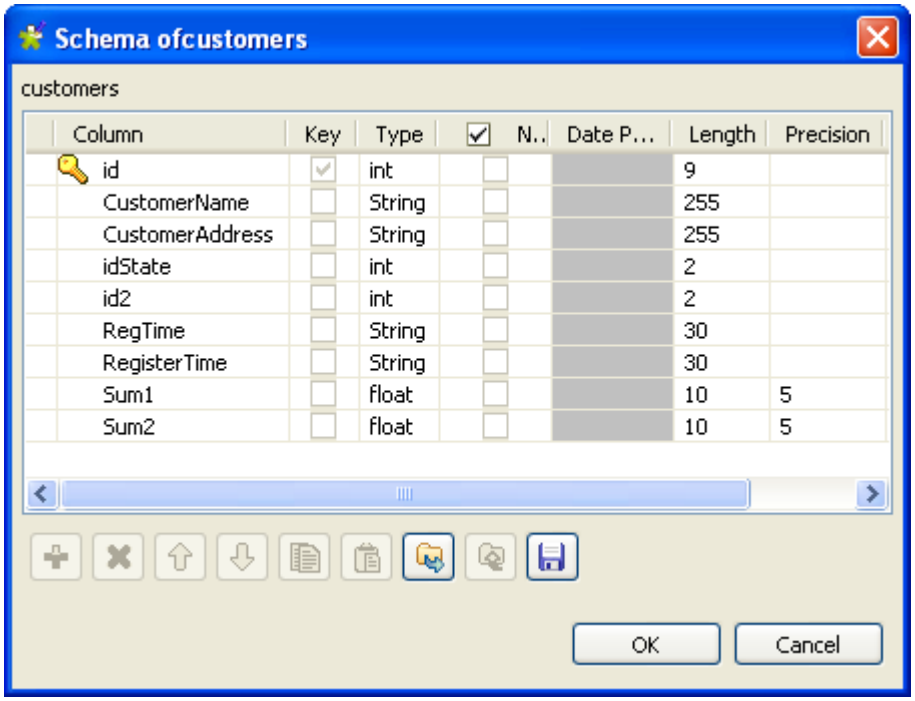
If the path of the file contains some accented characters, you will get an error message when executing your Job. For more information regarding the procedures to follow when the support of accented characters is missing, see the Installation Guide *of the **Talend** solution you are using.*

3. In the **Row Separators** and **Field Separators** fields, enter respectively "\n" and ";" as line and field separators.

4. If needed, set the number of lines used as header and the number of lines used as footer in the corresponding fields and then set a limit for the number of processed rows.

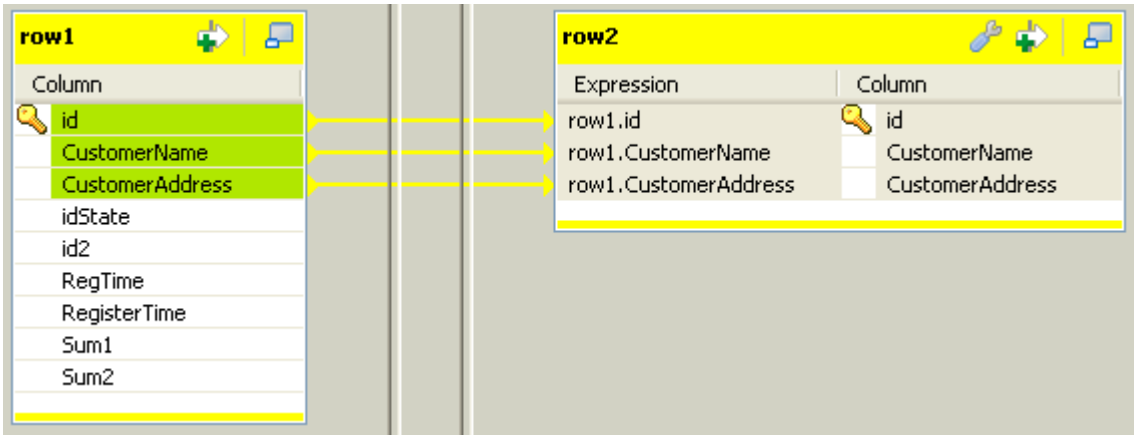
   In this example, **Header** is set to 6 while **Footer** and **Limit** are not set.

5. Click the **[...]** button next to **Edit Schema** to open the **[Schema]** dialog box and define the input schema as shown below, and then click **OK** to close the dialog box.



Configuring the mapping component

1. In the design workspace, double-click **tMap** to open its editor.



2. In the **tMap** editor, click on top of the panel to the right to open the **[Add a new output table]** dialog box.

3. Enter a name for the table you want to create, *row2* in this example.

4. Click **OK** to validate your changes and close the dialog box.

5. In the table to the left, *row1*, select the first three lines (*Id*, *CustomerName* and *CustomerAddress*) and drop them to the table to the right

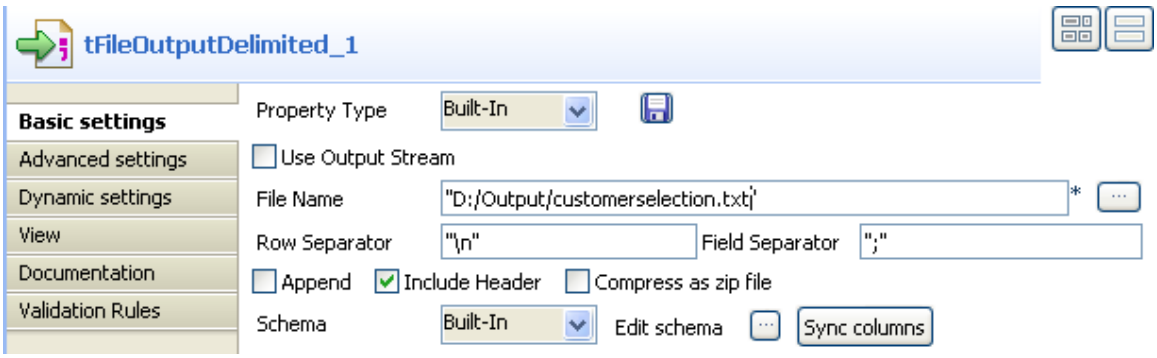
6. In the **Schema editor** view situated in the lower left corner of the **tMap** editor, change the type of *RegisterTime* to **String** in the table to the right.

Column	Key	Type	N..	Date Pattern ...	Length	Precision	Def...	Comm...
idState		int			2			
id2		int			2			
RegTime		String			30			
RegisterTime		String			30			
Sum1		float			10	5		
Sum2		float			10	5		

7. Click **OK** to save your changes and close the editor.

Configuring the output component

1. In the design workspace, double-click **tFileOutputDelimited** to open its **Basic settings** view and define the component properties.



2. In the **Property Type** field, set the type to **Built-in** and fill in the fields that follow manually.

3. Click the **[...]** button next to the **File Name** field and browse to the output file you want to write data in, *customerselection.txt* in this example.

4. In the **Row Separator** and **Field Separator** fields, set "\n" and ";" respectively as row and field separators.

5. Select the **Include Header** check box if you want to output columns headers as well.

6. Click **Edit schema** to open the schema dialog box and verify if the recuperated schema corresponds to the input schema. If not, click **Sync Columns** to recuperate the schema from the preceding component.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

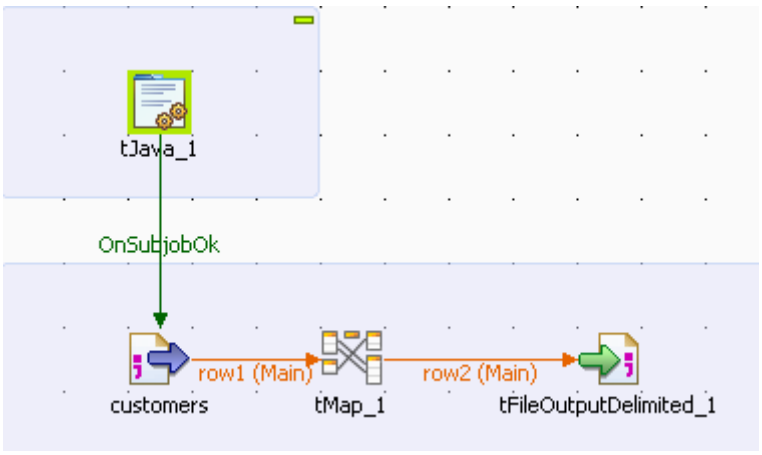
customerselection.txt

```
1  id;CustomerName;CustomerAddress
2  1;Griffith Paving and Sealcoatin;talend@apres91
3  2;Bill's Dive Shop;511 Maple Ave.  Apt. 1B
4  3;Childress Child Day Care;662 Lyons Circle
5  4;Facelift Kitchen and Bath;unknown
6  5;Terrinni & Son Auto and Truck;770 Exmoor Rd.
7  6;Kermit the Pet Shop;1860 Parkside Ln.
8  7;Tub's Furniture Store;807 Old Trail Rd.
9  8;Toggle & Myerson Ltd;618 Sheriden rd.
10 9;Childress Child Day Care;788 Tennyson Ave.
11 10;Elle Hypnosis and Therapy Cent;2032 Northbrook Ct.
```

The three specified columns *Id*, *CustomerName* and *CustomerAddress* are output in the defined output file.

Scenario 2: Utilizing Output Stream to save filtered data to a local file

Based on the preceding scenario, this scenario saves the filtered data to a local file using output stream.

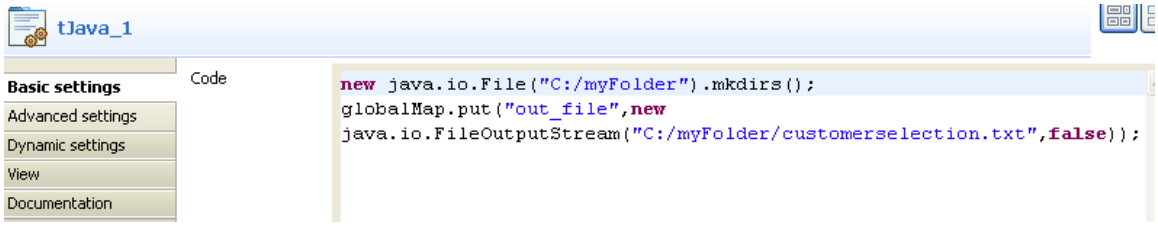


Dropping and linking components

1. Drop **tJava** from the **Palette** to the design workspace.
2. Connect **tJava** to **tFileInputDelimited** using a **Trigger** > **On Subjob OK** connection.

Configuring the components

1. Double-click **tJava** to open its **Basic settings** view.

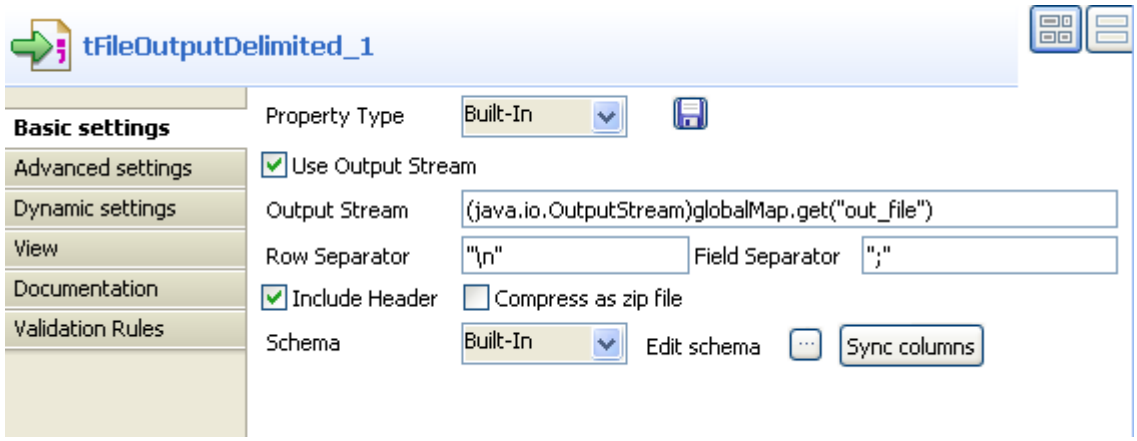


2. In the **Code** area, type in the following command:

```
new java.io.File("C:/myFolder").mkdirs();
globalMap.put("out_file",new
java.io.FileOutputStream("C:/myFolder/customerselection.txt",false));
```

In this scenario, the command we use in the **Code** area of **tJava** will create a new folder *C:/myFolder* where the output file *customerselection.txt* will be saved. You can customize the command in accordance with actual practice.

3. Double-click **tFileOutputDelimited** to open its **Basic settings** view.



4. Select **Use Output Stream** check box to enable the **Output Stream** field in which you can define the output stream using command.

   Fill in the **Output Stream** field with following command:

   ```
   (java.io.OutputStream)globalMap.get("out_file")
   ```

   You can customize the command in the **Output Stream** field by pressing **CTRL**+**SPACE** to select built-in command from the list or type in the command into the field manually in accordance with actual practice. In this scenario, the command we use in the **Output Stream** field will call the `java.io.OutputStream` class to output the filtered data stream to a local file which is defined in the **Code** area of **tJava** in this scenario.

5. Click **Sync columns** to retrieve the schema defined in the preceding component.

6. Leave rest of the components as they were in the previous scenario.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The three specified columns *Id*, *CustomerName* and *CustomerAddress* are output in the defined output file.

customerselection.txt

```
1  id;CustomerName;CustomerAddress
2  1;Griffith Paving and Sealcoatin;talend@apres91
3  2;Bill's Dive Shop;511 Maple Ave.  Apt. 1B
4  3;Childress Child Day Care;662 Lyons Circle
5  4;Facelift Kitchen and Bath;unknown
6  5;Terrinni & Son Auto and Truck;770 Exmoor Rd.
7  6;Kermit the Pet Shop;1860 Parkside Ln.
8  7;Tub's Furniture Store;807 Old Trail Rd.
9  8;Toggle & Myerson Ltd;618 Sheriden rd.
10 9;Childress Child Day Care;788 Tennyson Ave.
11 10;Elle Hypnosis and Therapy Cent;2032 Northbrook Ct.
```

tFileOutputExcel

tFileOutputExcel Properties

Component family	File/Output	
Function	**tFileOutputExcel** outputs data to an MS Excel type of file.	
Purpose	**tFileOutputExcel** writes an MS Excel file with separated data value according to a defined schema.	
Basic settings	*Write excel 2007 file format (xlsx)*	Select this check box to write the processed data into the *.xlsx* format of Excel 2007.
	Use Output Stream	Select this check box process the data flow of interest. Once you have selected it, the **Output Stream** field displays and you can type in the data flow of interest. The data flow to be processed must be added to the flow in order for this component to fetch these data via the corresponding representattive variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of writing manually, you could select the variable of interest from the auto-completion list (**Ctrl+Space**) to fill the current field on condition that this variable has been properly defined. For further information about how to use a stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	File name	Name or path to the output file. This field becomes unavailable once you have selected the **Use Output Stream** check box Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sheet name	Name of the xsl sheet.
	Include header	Select this check box to include a header row to the output file.
	Append existing file	Select this check box to add the new lines at the end of the file. **Append existing sheet**: Select this check box to add the new lines at the end of the Excel sheet.
	Is absolute Y pos.	Select this check box to add information in specified cells: **First cell X**: cell position on the X-axis (X-coordinate or Abcissa). **First cell Y**: cell position on the Y-axis (Y-coordinate). **Keep existing cell format**: select this check box to retain the original layout and format of the cell you want to write into.
	Font	Select in the list the font you want to use.
	Define all columns auto size	Select this check box if you want the size of all your columns to be defined automatically. Otherwise, select the **Auto size** check

		boxes next to the column names you want their size to be defined automatically.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Create directory if not exists*	This check box is selected by default. This option creates the directory that will hold the output files if it does not already exist.
	Custom the flush buffer size	Available when **Write excel2007 file format (xlsx)** is selected in the **Basic settings** view. Select this check box to set the maximum number of rows in the **Row number** field that are allowed in the buffer.
	Advanced separator (for numbers)	Select this check box to modify the separators you want to use for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write an XML file with data passed on from other components using a **Row** link.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For **tFileOutputExcel** related scenario, see section *tSugarCRMInput*;

For scenario about the usage of **Use Output Stream** check box, see section *Scenario 2: Utilizing Output Stream to save filtered data to a local file*.

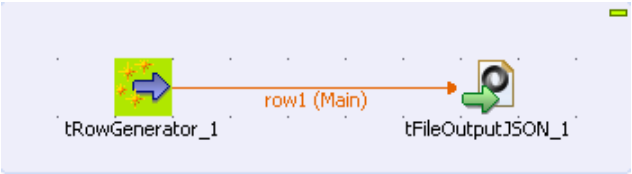
tFileOutputJSON

tFileOutputJSON properties

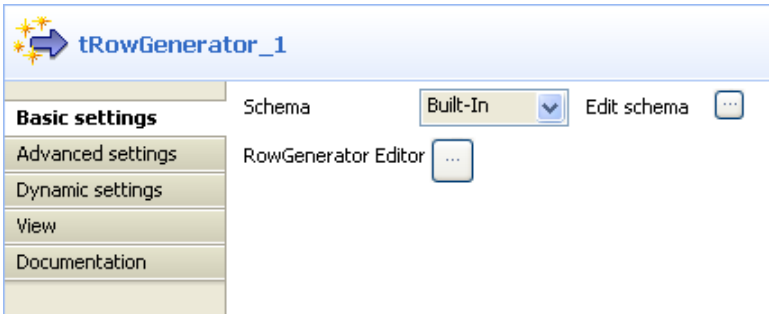
Component Family	File	
Function	**tFileOutputJSON** writes data to a JSON structured output file.	
Purpose	**tFileOutputJSON** receives data and rewrites it in a JSON structured data block in an output file.	
Basic settings	*File Name*	Name and path of the output file.
	Name of data block	Enter a name for the data block to be written, between double quotation marks.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
Advanced settings	*Create directory if not exists*	This check box is selected by default. This option creates the directory that will hold the output files if it does not already exist.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to rewrite received data in a JSON structured output file.	
Limitation	n/a	

Scenario: Writing a JSON structured file

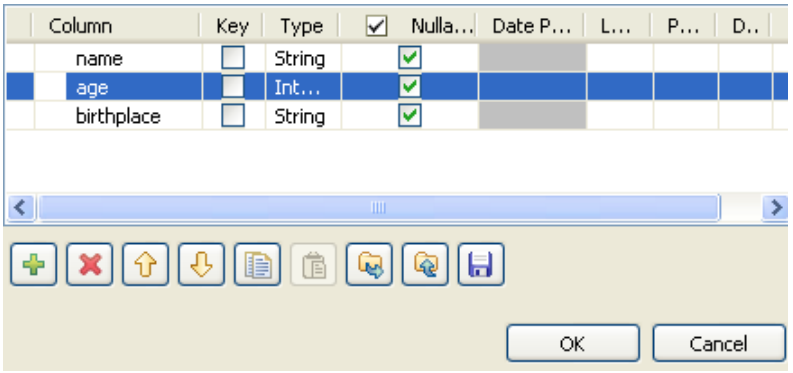
This is a 2 component scenario in which a **tRowGenerator** component generates random data which a **tFileOutputJSON** component then writes to a JSON structured output file.



1. Drop a **tRowGenerator** and a **tFileOutputJSON** component onto the workspace from the **Palette**.
2. Link the components using a **Row > Main** connection.
3. Double click **tRowGenerator** to define its **Basic Settings** properties in the **Component** view.



4. Click **[...]** next to **Edit Schema** to display the corresponding dialog box and define the schema.



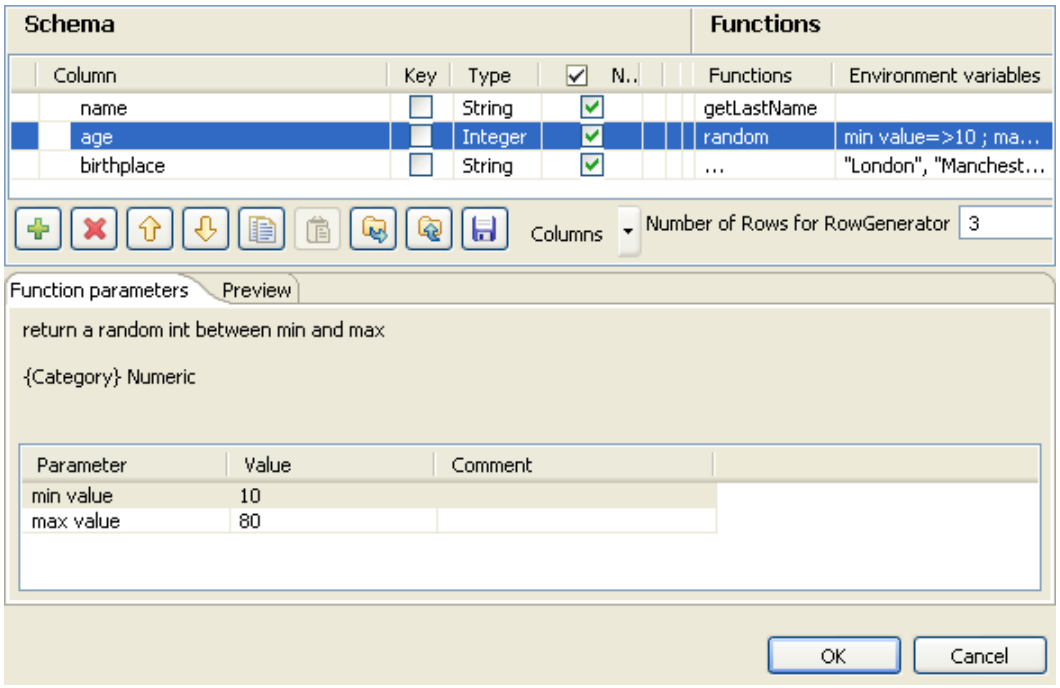
5. Click **[+]** to add the number of columns desired.

6. Under **Columns** type in the column names.

7. Under **Type**, select the data type from the list.

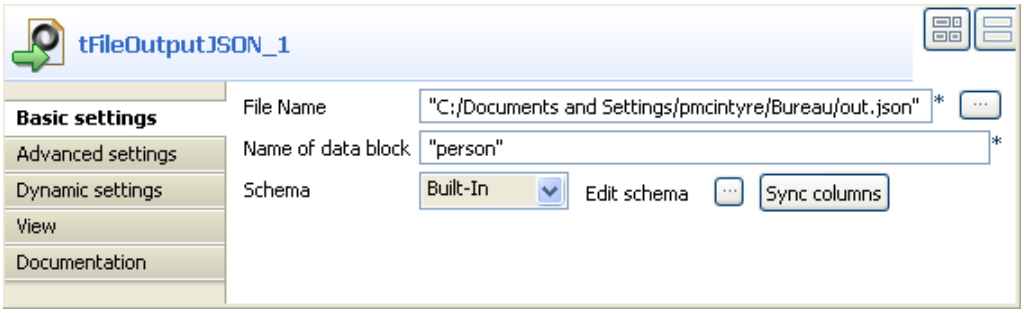
8. Click **OK** to close the dialog box.

9. Click **[+]** next to **RowGenerator Editor** to open the corresponding dialog box.



10. Under **Functions**, select pre-defined functions for the columns, if required, or select **[...]** to set customized function parameters in the **Function parameters** tab.

11. Enter the number of rows to be generated in the corresponding field.
12. Click **OK** to close the dialog box.
13. Click **tFileOutputJSON** to set its **Basic Settings** properties in the **Component** view.



14. Click **[...]** to browse to where you want the output JSON file to be generated and enter the file name.
15. Enter a name for the data block to be generated in the corresponding field, between double quotation marks.
16. Select **Built-In** as the **Schema** type.
17. Click **Sync Columns** to retrieve the schema from the preceding component.
18. Press **F6** to run the Job.

```
{
    "person":[
        {
            "birthplace":"Manchester",
            "age":48,
            "name":"Carter"
        },
        {
            "birthplace":"Liverpool",
            "age":39,
            "name":"Clinton"
        },
        {
            "birthplace":"London",
            "age":53,
            "name":"Taylor"
        }
]
}
```

The data from the input schema is written in a JSON structured data block in the output file.

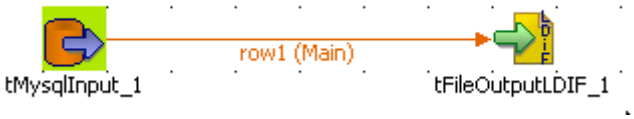
tFileOutputLDIF

tFileOutputLDIF Properties

Component family	File/Output	
Function	**tFileOutputLDIF** outputs data to an LDIF type of file which can then be loaded into a LDAP directory.	
Purpose	**tFileOutputLDIF** writes or modifies a LDIF file with data separated in respective entries based on the schema defined,.or else deletes content from an LDIF file.	
Basic settings	*File name*	Name or path to the output file and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Wrap	Wraps the file content, every defined number of characters.
	Change type	Select **Add**, **Modify** or **Delete** to respectively create an LDIF file, modify or remove an existing LDIF file. In case of modification, set the type of attribute changes to be made.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
	Append	Select this check box to add the new rows at the end of the file.
Advanced settings	*Create directory if not exists*	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Custom the flush buffer size	Select this check box to define the number of lines to write before emptying the buffer. **Row Number**: set the number of lines to write.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write an XML file with data passed on from other components using a Row link.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in *Talend Open Studio for Big Data* . For details, see the *Warnings and error icons on components* and *How to install external modules* sections of *Talend Open Studio for Big Data User Guide*.	

Scenario: Writing DB data into an LDIF-type file

This scenario describes a two component Job which aims at extracting data from a database table and writing this data into a new output LDIF file.

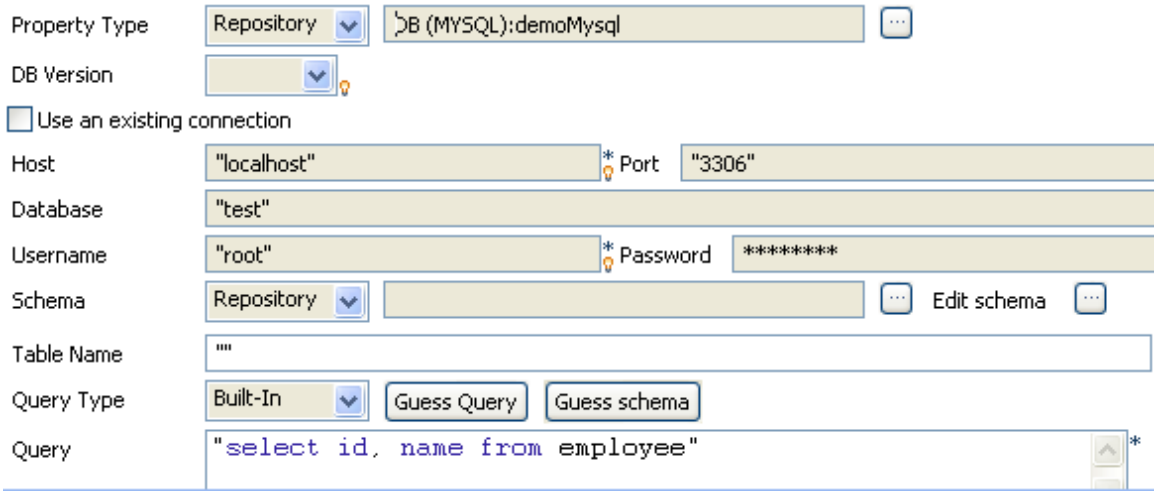


Dropping and linking components

1. Drop a **tMysqlInput** component and a **tFileOutputLDIF** component from the **Palette** to the design area.
2. Connect the components together using a **Row > Main** link.

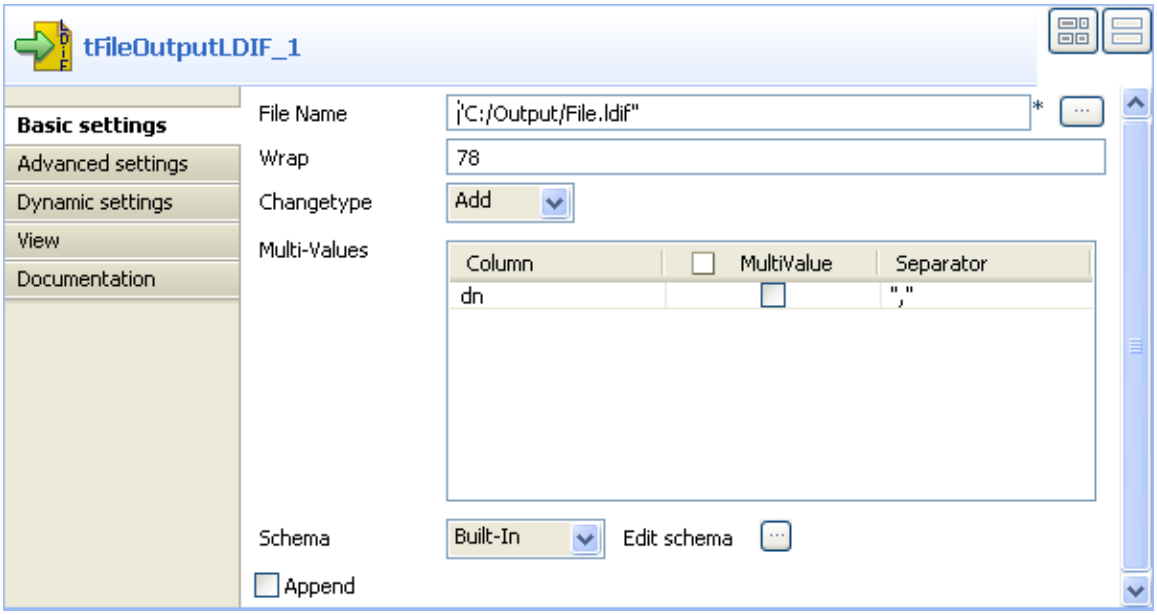
Configuring the components

1. Select the **tMysqlInput** component, and go to the **Component** panel then select the **Basic settings** tab.



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

2. Define the DB connection and schema manually.
3. Then double-click on **tFileOutpuLDIF** and define the **Basic settings**.
4. Browse to the folder where you store the Output file. In this use case, a new LDIF file is to be created. Thus type in the name of this new file.
5. In the **Wrap** field, enter the number of characters held on one line. The text coming afterwards will get wrapped onto the next line.



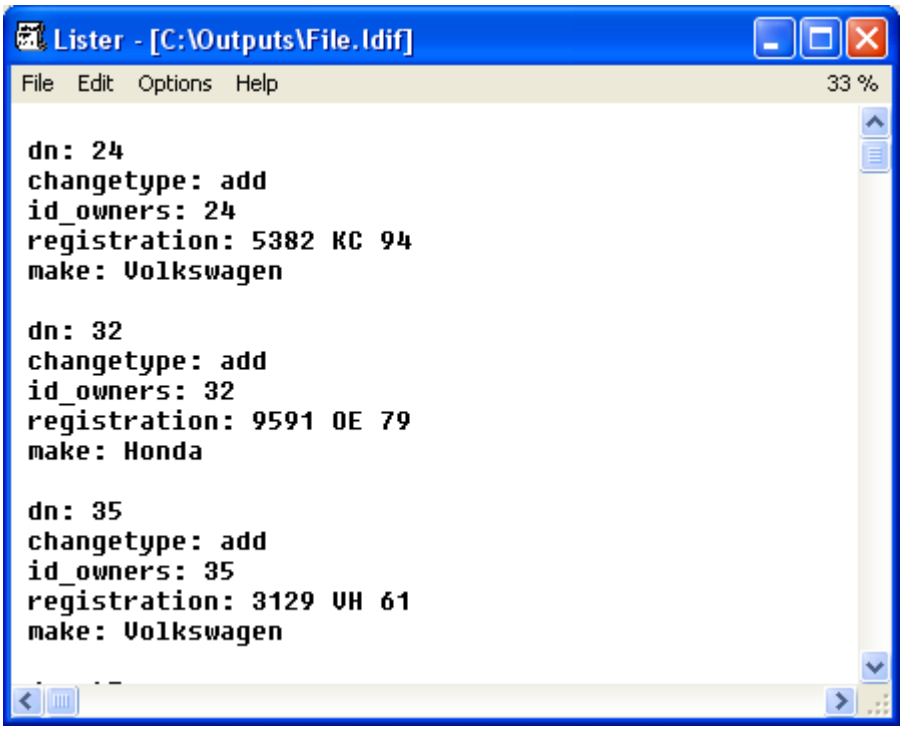
6. Select **Add** as **Change Type** as the newly created file is by definition empty. In case of modification type of Change, you'll need to define the nature of the modification you want to make to the file.

7. As the **Schema** type, select **Built-in** and use the **Sync Columns** button to retrieve the input schema definition.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.



The LDIF file created contains the data from the DB table and the type of change made to the file, in this use case, addition.

tFileOutputMSDelimited

tFileOutputMSDelimited properties

Component family	File/ Output	
Function	**tFileOutputMSDelimited** writes multiple schema in a delimited file.	
Purpose	**tFileOutputMSDelimited** creates a complex multi-structured delimited file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	*File Name*	Name and path to the file to be created and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row Separator	String (ex: "\n"on Unix) to distinguish rows.
	Field Separator	Character, string or regular expression to separate fields.
	Use Multi Field Separators	Select this check box to set a different field separator for each of the schemas using the **Field separator** field in the **Schemas** area.
	Schemas	The table gets automatically populated by schemas coming from the various incoming rows connected to **tFileOutputMSDelimited**. Fill out the dependency between the various schemas: **Parent row**: Type in the parent flow name (based on the **Row** name transferring the data). **Parent key column**: Type in the key column of the parent row. **Key column**: Type in the key column for the selected row.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	CSV options	Select this check box to take into account all parameters specific to CSV files, in particular **Escape char** and **Text enclosure** parameters.
	Create directory if not exists	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write a multi-schema delimited file and separate fields using a field separator value.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

No scenario is available for this component yet.

tFileOutputMSPositional

tFileOutputMSPositional properties

Component family	File/Output	
Function	**tFileOutputMSPositional** writes multiple schemas in a positional file.	
Purpose	**tFileOutputMSPositional** creates a complex multi-structured file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	*File Name*	Name and path to the file to be created and/or variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Schemas	The table gets automatically populated by schemas coming from the various incoming rows connected to **tFileOutputMSPositional**. Fill out the dependency between the various schemas: **Parent row**: Type in the parent flow name (based on the Row name transferring the data). **Parent key column**: Type in the key column of the parent row **Key column**: Type in the key column for the selected row. **Pattern**: Type in the pattern that positions the fields separator for each incoming row. **Padding char**: type in the padding character to be used **Alignment**: Select the relevant alignment parameter
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Create directory if not exists	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write a multi-schema positional file and separate fields using a position separator value.	

Related scenario

No scenario is available for this component yet.

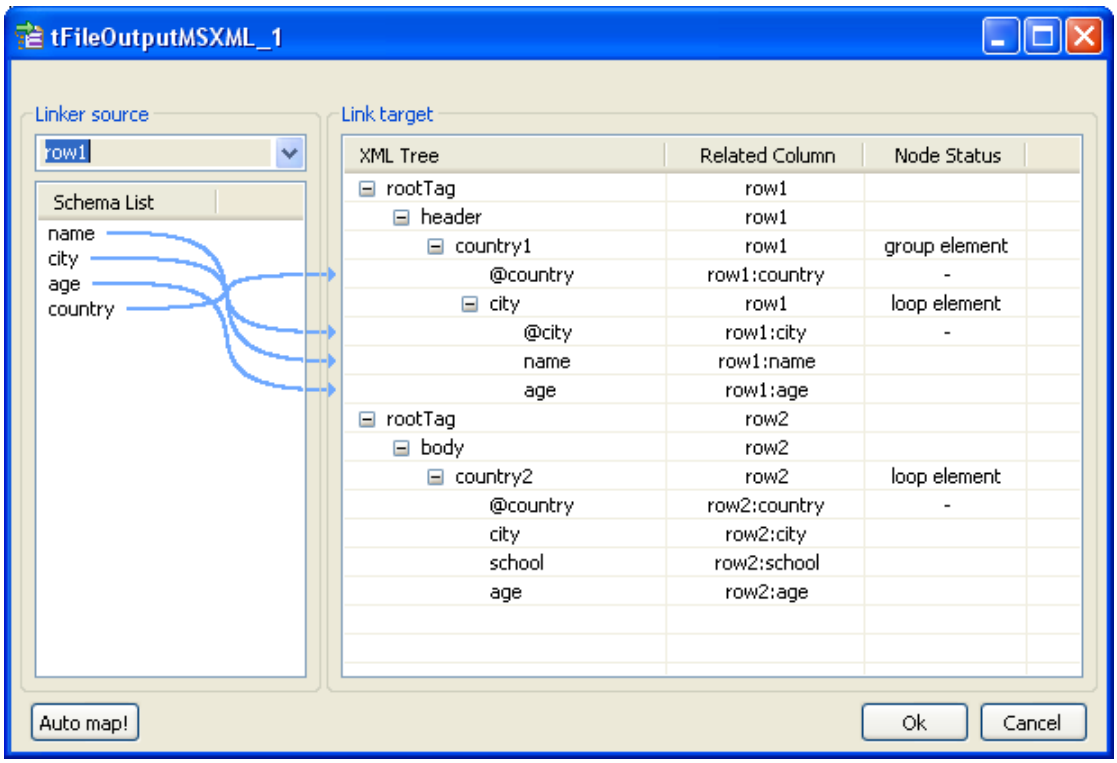
tFileOutputMSXML

tFileOutputMSXML Properties

Component family	File/Output	
Function	**tFileOutputMSXML** writes multiple schema within an XML structured file.	
Purpose	**tFileOutputMSXML** creates a complex multi-structured XML file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	*File Name*	Name and path to the file to be created and or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Configure XML tree	Opens the dedicated interface to help you set the XML mapping. For details about the interface, see section *Defining the MultiSchema XML tree*.
Advanced settings	*Create directory only if not exists*	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Advanced separator (for numbers)	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Encoding	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select this check box if you do not want to generate empty files.
	Trim the whitespace characters	Select this check box to remove leading and trailing whitespace from the columns.
	Escape text	Select this check box to escape special characters.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Limitation	n/a	

Defining the MultiSchema XML tree

Double-click on the **tFileOutputMSXML** component to open the dedicated interface or click on the three-dot button on the **Basic settings** vertical tab of the **Component** tab.



To the left of the mapping interface, under **Linker source**, the drop-down list includes all the input schemas that should be added to the multi-schema output XML file (on the condition that more than one input flow is connected to the **tFileOutputMSXML** component).

And under **Schema List**, are listed all columns retrieved from the input data flow in selection.

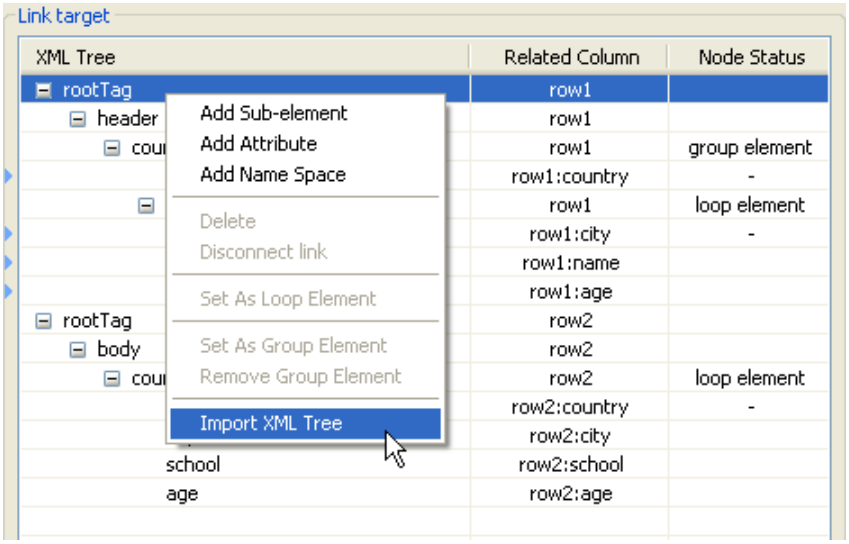
To the right of the interface, are expected all XML structures you want to create in the output XML file.

You can create manually or easily import the XML structures. Then map the input schema columns onto each element of the XML tree, respectively for each of the input schemas in selection under **Linker source**.

Importing the XML tree

The easiest and most common way to fill out the XML tree panel, is to import a well-formed XML file.

1. Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
2. Right-click on the root tag to display the contextual menu.
3. On the menu, select **Import XML tree**.
4. Browse to the file to import and click **OK**.



The **XML Tree** column is hence automatically filled out with the correct elements. You can remove and insert elements or sub-elements from and to the tree:

5. Select the relevant element of the tree.
6. Right-click to display the contextual menu
7. Select Delete to remove the selection from the tree or select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** to enrich the tree.

Creating manually the XML tree

If you don't have any XML structure already defined, you can manually create it.

1. Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
2. Right-click on the root tag to display the contextual menu.
3. On the menu, select **Add sub-element** to create the first element of the structure.

   You can also add an attribute or a child element to any element of the tree or remove any element from the tree.
4. Select the relevant element on the tree you just created.
5. Right-click to the left of the element name to display the contextual menu.
6. On the menu, select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** or **Delete**.

Mapping XML data from multiple schema sources

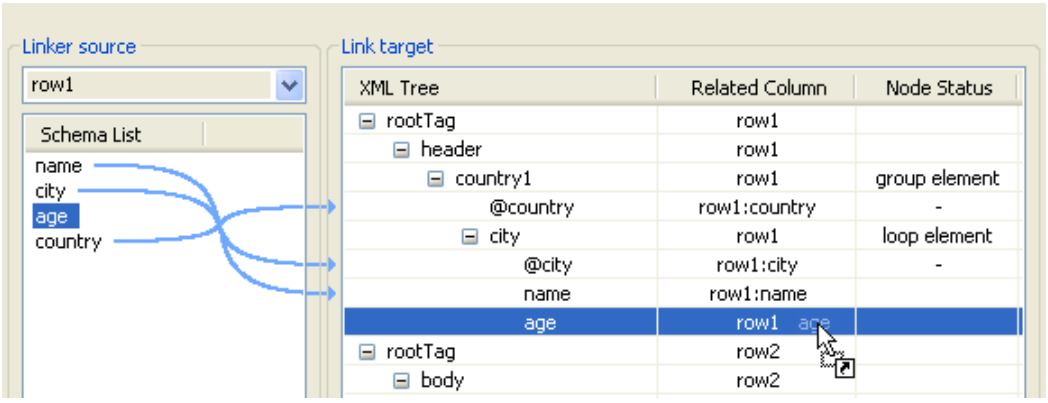
Once your XML tree is ready, select the first input schema that you want to map.

You can map each input column with the relevant XML tree element or sub-element to fill out the **Related Column**:

1. Click on one of the **Schema column name**.

2. Drag it onto the relevant sub-element to the right.

3. Release the mouse button to implement the actual mapping.



A light blue link displays that illustrates this mapping. If available, use the **Auto-Map** button, located to the bottom left of the interface, to carry out this operation automatically.

You can disconnect any mapping on any element of the XML tree:

4. Select the element of the XML tree, that should be disconnected from its respective schema column.

5. Right-click to the left of the element name to display the contextual menu.

6. Select **Disconnect link**.

The light blue link disappears.

Defining the node status

Defining the XML tree and mapping the data is not sufficient. You also need to define the loop elements **for each of the source in selection** and if required the group element.

Loop element

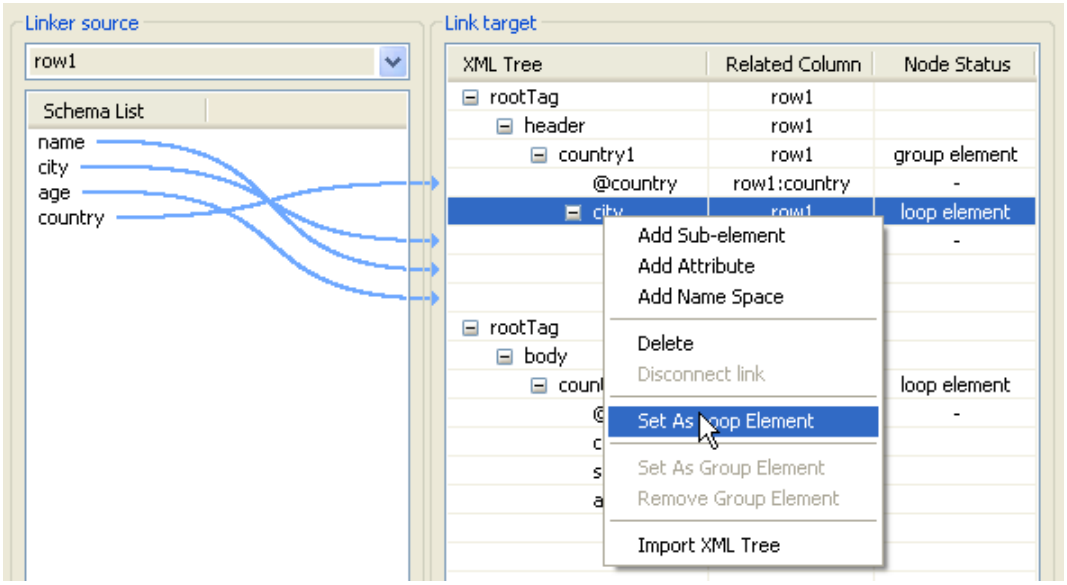
The loop element allows you to define the iterating object. Generally the Loop element is also the row generator.

To define an element as loop element:

1. Select the relevant element on the XML tree.

2. Right-click to the left of the element name to display the contextual menu.

3. Select **Set as Loop Element**.



The **Node Status** column shows the newly added status.

There can only be one loop element at a time.

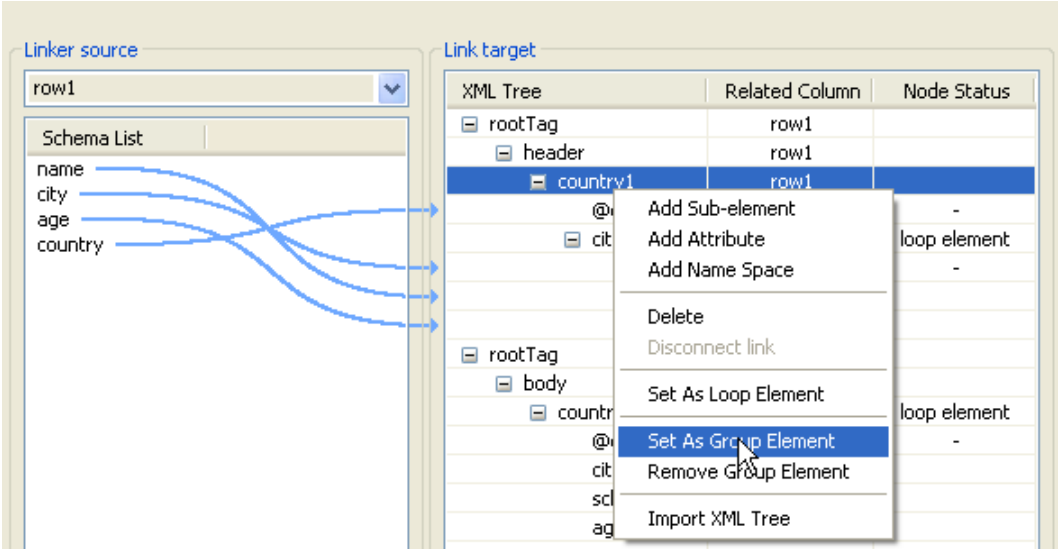
Group element

The group element is optional, it represents a constant element where the Groupby operation can be performed. A group element can be defined on the condition that a loop element was defined before.

When using a group element, the rows should be sorted, in order to be able to group by the selected node.

To define an element as group element:

1. Select the relevant element on the XML tree.
2. Right-click to the left of the element name to display the contextual menu.
3. Select **Set as Group Element**.



The **Node Status** column shows the newly added status and any group status required are automatically defined, if needed.

Click **OK** once the mapping is complete to validate the definition for this source and perform the same operation for the other input flow sources.

Related scenario

No scenario is available for this component yet.

tFileOutputPositional

tFileOutputPositional Properties

Component Family	File/Output	
Function	**tFileOutputPositional** writes a file row by row according to the length and the format of the f ields or columns in a row.	
Purpose	It writes a file row by row, according to the data structure (schema) coming from the input flow.	
Basic settings	*Property type*	In Talend Open Studio for Big Data, the Property type is always built-in.
		Built-in: No property data stored centrally.
	Use Output Stream	Select this check box process the data flow of interest. Once you have selected it, the **Output Stream** field displays and you can type in the data flow of interest. The data flow to be processed must be added to the flow in order for this component to fetch these data via the corresponding representative variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl+Space**) to fill the current field on condition that this variable has been properly defined. For further information about how to use a stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	File Name	Name or path to the file to be processed and or the variable to be used. This field becomes unavailable once you have selected the **Use Output Stream** check box. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Schema and Edit Schema	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows in the output file.
	Append	Select this check box to add the new rows at the end of the file.
	Include header	Select this check box to include the column header to the file.
	Compress as zip file	Select this check box to compress the output file in zip format.
	Formats	Customize the positional file data format and fill in the columns in the Formats table. **Column**: Select the column you want to customize. **Size**: Enter the column size. **Padding char**: Type in between quotes the padding characters used. A space by default.

		Alignment: Select the appropriate alignment parameter. **Keep**: If the data in the column or in the field are too long, select the part you want to keep.
Advanced settings	*Advanced separator (for numbers)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Use byte length as the cardinality	Select this checkbox to add support of double-byte character to this component. JDK 1.6 is required for this feature.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if it does not exist.
	Custom the flush buffer size	Select this check box to define the number of lines to write before emptying the buffer. **Row Number**: set the number of lines to write.
	Output in row mode	Writes in row mode.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to read a file and separate the fields using the specified separator.	

Related scenario

For a related scenario, see section *Scenario 2: Handling a positional file based on a dynamic schema*.

For scenario about the usage of **Use Output Stream** check box, see section *Scenario 2: Utilizing Output Stream to save filtered data to a local file*.

tFileOutputProperties

tFileOutputProperties properties

Component family	File/Output	
Function	**tFileInputProperties** writes a configuration file of the type **.ini** or **.properties**.	
Purpose	**tFileInputProperties** writes a configuration file containing text data organized according to the model key = value.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in. For this component, the schema is read-only. It is made of two column, *Key* and *Value*, corresponding to the parameter name and the parameter value to be copied.
	File format	Select from the list file format: either **.properties** or **.ini**.
		.properties: data in the configuration file is written in two lines and structured according to the following way: key = value.
		.ini: data in the configuration file is written in two lines and structured according to the following way: key = value and re-grouped in sections. **Section Name**: enter the section name on which the iteration is based.
	File Name	Name or path to the file to be processed and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Encoding*	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write files where data is organized according to the structure key = value.	

Related scenarios

For a related scenario, see section *Scenario: Reading and matching the keys and the values of different .properties files and outputting the results in a glossary* of section *tFileInputProperties*.

tFileOutputXML

tFileOtputXML belongs to two component families: File and XML. For more information on **tFileOutputXML,** see section *tFileOutputXML*.

tFileProperties

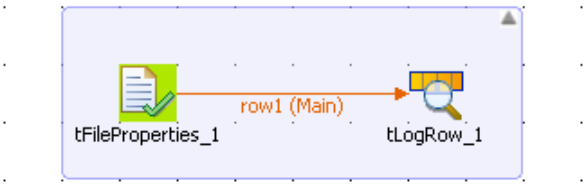
tFileProperties Properties

Component family	File/Management	
Function	**tFileProperties** creates a single row flow that displays the properties of the processed file.	
Purpose	**tFileProperties** obtains information about the main properties of a defined file.	
Basic settings	*Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File	Name or path to the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Calculate MD5 Hash	Select this check box to check the MD5 of the downloaded file.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as standalone component.	
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

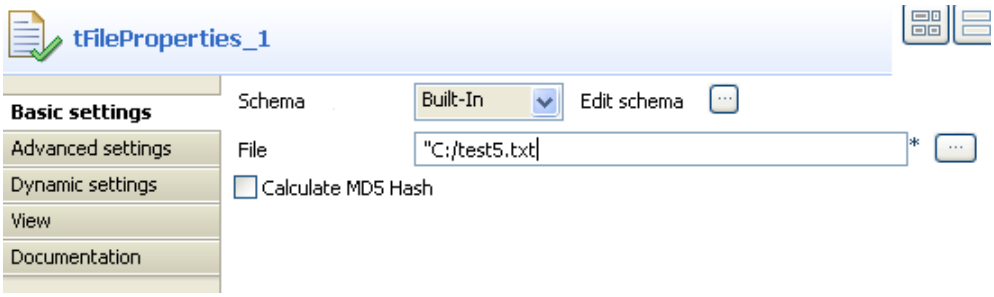
Scenario: Displaying the properties of a processed file

This Java scenario describes a very simple Job that displays the properties of the specified file.

1. Drop a **tFileProperties** component and a **tLogRow** component from the **Palette** onto the design workspace.
2. Right-click on **tFileProperties** and connect it to **tLogRow** using a **Main Row** link.



3. In the design workspace, select **tFileProperties**.
4. Click the **Component** tab to define the basic settings of **tFileProperties**.



5. Set **Schema** type to **Built-In**.
6. If desired, click the **Edit schema** button to see the read-only columns.
7. In the **File** field, enter the file path or browse to the file you want to display the properties for.
8. In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*.
9. Press **F6** to execute the Job.

```
Starting job File_Properties at 14:57 26/08/2008.
.-------------------------------------------------.
|                #1. tLogRow_1                    |
+---------------+---------------------------------+
| key           | value                           |
+---------------+---------------------------------+
| abs_path      | C:\test5.txt                    |
| dirname       | C:\                             |
| basename      | test5.txt                       |
| mode_string   | rw                              |
| size          | 86                              |
| mtime         | 1219674736421                   |
| mtime_string  | Mon Aug 25 16:32:16 CEST 2008   |
+---------------+---------------------------------+

Job File_Properties ended at 14:57 26/08/2008. [exit code=0]
```

The properties of the defined file are displayed on the console.

tFileRowCount

tFileRowCount properties

Component Family	File/Management	
Function	**tFileRowCount** counts the number of rows in a file.	
Purpose	**tFileRowCount** opens a file and reads it row by row in order to determine the number of rows inside.	
Basic settings	*File Name*	Name and path of the file to be processed and/or the variable to be used. See also: *Talend Open Studio for Big Data User Guide*.
	Row separator	String (ex: “\n”on Unix) to distinguish rows in the output file.
	Ignore empty rows	Select this checkbox to ignore the empty rows while the component is counting the rows in the file.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	**tFileRowCount** is a standalone component, it must be used with a **OnSubjobOk** connection to **tJava**.	
Global Variables		**Number of counted lines**: Returns the number of rows in a file. This is available as a **Flow** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

No scenario is available for this component yet.

tFileTouch

tFileTouch properties

Component Family	File/Management	
Function	**tFileTouch** creates an empty file.	
Purpose	This component creates an empty file, and creates the destination directory if it does not exist.	
Basic settings	*File Name*	Path and name of the file to be created and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if it does not exist.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component can be used as a standalone component.	
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.

Related scenario

No scenario is available for this component yet.

tFileUnarchive

tFileUnarchive Properties

Component family	File/Management	
Function	Decompresses the archive file provided as parameter and puts it in the extraction directory.	
Purpose	Decompresses an archive file for further processing. Such formats are supported: *.tar.gz , *.tgz, *.tar, *.gz and *.zip.	
Basic settings	*Archive file*	File path to the archive.
	Extraction Directory	Folder where the unzipped file(s) will be put.
	Use archive name as root directory	Select this check box to create a folder named as the archive, if it does not exist, under the specified directory and extract the zipped file(s) to that folder.
	Check the integrity before unzip	Select this check box to run an integrity check before unzipping the archive.
	Extract file paths	Select this check box to reproduce the file path structure zipped in the archive.
	Need a password	Select this check box and provide the correct password if the archive to be unzipped is password protected. Note that the encrypted archive must be one created by the **tFileArchive** component; otherwise you will see error messages or get nothing extracted even if no error message is displayed.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component can be used as a standalone component but it can also be used within a Job as a Start component using an **Iterate** link.	
Global Variables		**Current File**: Retrieves the name of the decompressed archive file. This is available as a **Flow** variable. Returns a string. **Current File Path**: Retrieves the path to the decompressed archive file.This is available as a **Flow** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate. **Trigger**: Run if; On Subjob Ok; On Subjob Error; On component Ok; On Component Error; Synchronize; Parallelize.

		For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation		*Such files can be decompressed: *.tar.gz , *.tgz, *.tar, *.gz and *.zip.*

Related scenario

For **tFileUnarchive** related scenario, see section *tFileCompare*.

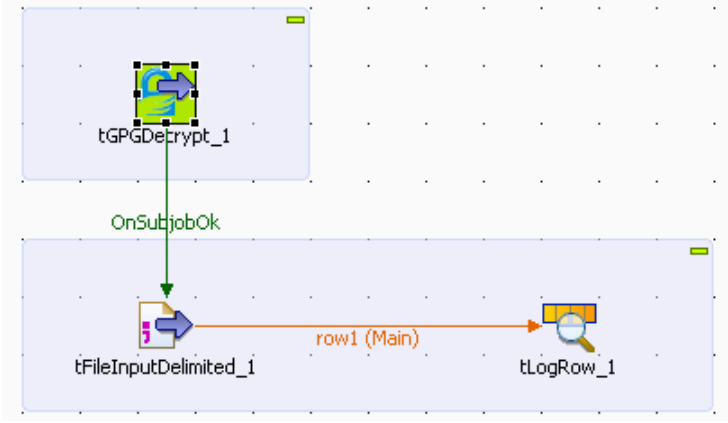
tGPGDecrypt

tGPGDecrypt Properties

Component family	File/Management	
Function	Decrypts a GnuPG-encrypted file and saves the decrypted file in the specified target directory.	
Purpose	This component calls the `gpg -d` command to decrypt a GnuPG-encrypted file and saves the decrypted file in the specified directory.	
Basic settings	*Input encrypted file*	File path to the encrypted file.
	Output decrypted file	File path to the output decrypted file.
	GPG binary path	File path to the GPG command.
	Passphrase	Enter the passphrase used in encrypting the specified input file.
	No TTY Terminal	Select this check box to speficy that no TTY terminal is used by adding the `--no-tty` option to the decryption command.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component can be used as a standalone component.	
Limitation	n/a	

Scenario: Decrypt a GnuPG-encrypted file and display its content

The following scenario describes a three-component Job that decrypts a GnuPG-encrypted file and displays the content of the decrypted file on the **Run** console.



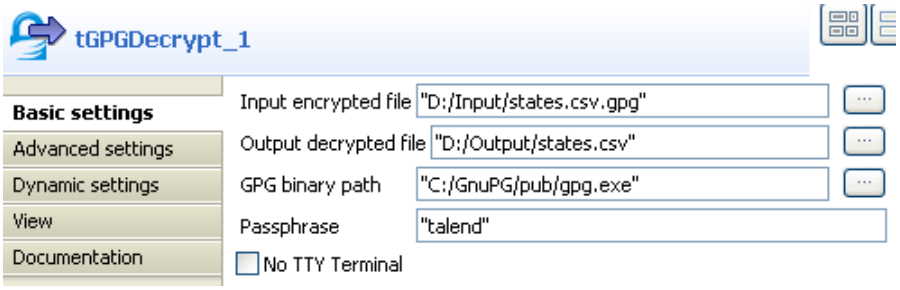
Dragging and linking the components

1. Drop a **tGPGDecrypt** component, a **tFileInputDelimited** component, and a **tLogRow** component from the **Palette** to the design workspace.

2. Connect the **tGPGDecrypt** component to the **tFileInputDelimited** component using a **Trigger** > **OnSubjobOk** link, and connect the **tFileInputDelimited** component to the **tLogRow** component using a **Row** > **Main** link.

Configuring the components

1. Double-click the **tGPGDecrypt** to open its **Component** view and set its properties:



2. In the **Input encrypted file** field, browse to the file to be decrypted.

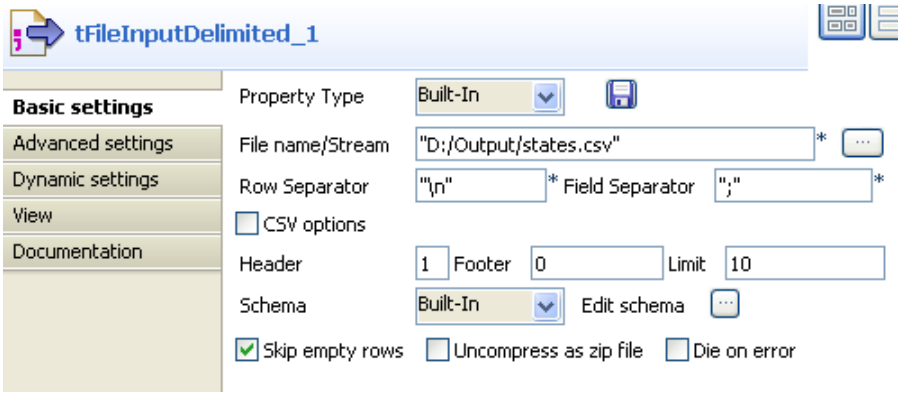
3. In the **Output decrypted file** field, enter the path to the decrypted file.

*If the file path contains accented characters, you will get an error message when running the Job. For more information on what to do when the accents are not supported, see the Installation Guide of the **Talend** solution you are using.*

4. In the **GPG binary path** field, browse to the GPG command file.

5. In the **Passphrase** field, enter the passphrase used when encrypting the input file.

6. Double-click the **tFileInputDelimited** component to open its **Component** view and set its properties:



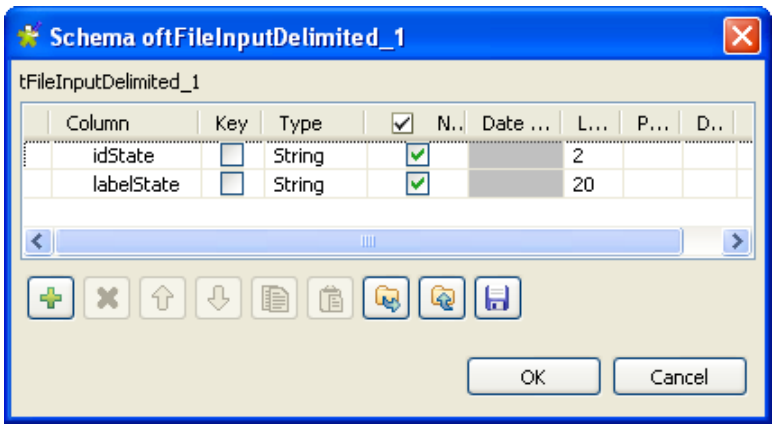
7. In the **File name/Stream** field, define the path to the decrypted file, which is the output path you have defined in the **tGPGDecrypt** component.

8. In the **Header**, **Footer** and **Limit** fields, define respectively the number of rows to be skipped in the beginning of the file, at the end of the file and the number of rows to be processed.

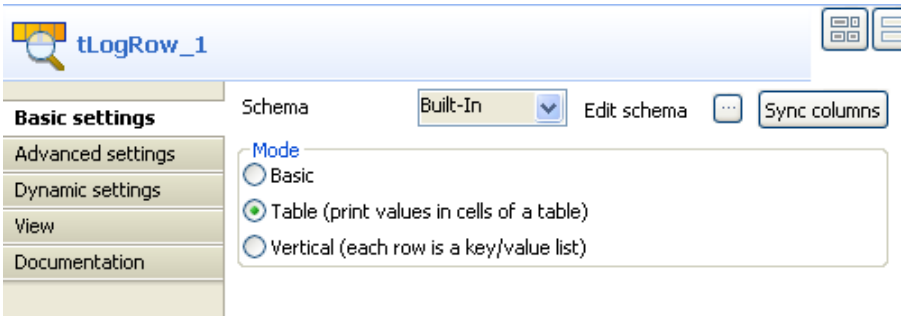
9. Use a **Built-In** schema. This means that it is available for this Job only.

10. Click **Edit schema** and edit the schema for the component. Click twice the **[+]** button to add two columns that you will call *idState* and *labelState*.

11. Click **OK** to validate your changes and close the editor.



12. Double-click the **tLogRow** component and set its properties:



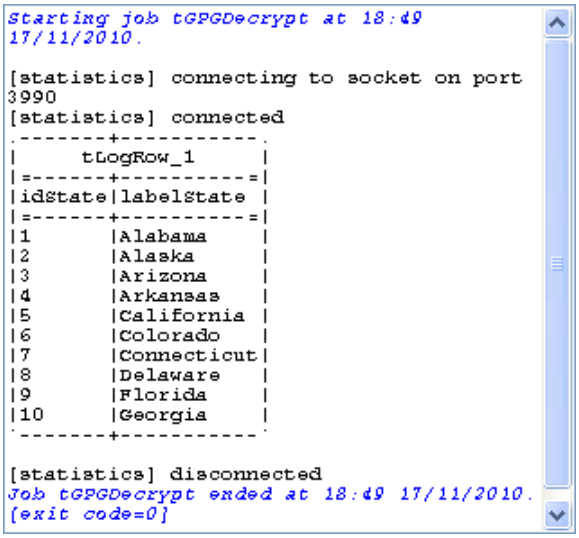
13. Use a **Built-In** schema for this scenario.

14. In the **Mode** area, define the console display mode according to your preference. In this scenario, select **Table (print values in cells of a table)**.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job

2. Press **F6** or click **Run** from the **Run** tab to run it.



The specified file is decrypted and the defined number of rows of the decrypted file are printed on the **Run** console.

tHDFSCompare

tHDFSCompare component belongs to two component families: Big Data and File. For more information about **tHDFSCompare**, see section *tHDFSCompare*.

tHDFSConnection

tHDFSConnection component belongs to two component families: Big Data and File. For more information about **tHDFSConnection**, see section *tHDFSConnection*.

tHDFSCopy

tHDFSCopy belongs to two component families: Big Data and File. For more information on **tHDFSCopy**, see section *tHDFSCopy*.

tHDFSDelete

tHDFSDelete component belongs to two component families: Big Data and File. For more information about **tHDFSDelete**, see section *tHDFSDelete*.

tHDFSExist

tHDFSExist component belongs to two component families: Big Data and File. For more information about **tHDFSExist**, see section *tHDFSExist*.

tHDFSGet

tHDFSGet component belongs to two component families: Big Data and File. For more information about **tHDFSGet**, see section *tHDFSGet*.

tHDFSList

tHDFSList belongs to two component families: Big Data and File. For more information on **tHDFSList**, see section *tHDFSList*.

tHDFSInput

tHDFSInput component belongs to two component families: Big Data and File. For more information about **tHDFSInput**, see section *tHDFSInput*.

tHDFSOutput

tHDFSOutput component belongs to two component families: Big Data and File. For more information about **tHDFSOutput**, see section *tHDFSOutput*.

tHDFSProperties

tHDFSProperties component belongs to two component families: Big Data and File. For more information about **tHDFSProperties**, see section *tHDFSProperties*.

tHDFSPut

tHDFSPut component belongs to two component families: Big Data and File. For more information about **tHDFSPut**, see section *tHDFSPut*.

tHDFSRename

tHDFSRename component belongs to two component families: Big Data and File. For more information about **tHDFSRename**, see section *tHDFSRename*.

tHDFSRowCount

tHDFSRowCount component belongs to two component families: Big Data and File. For more information about **tHDFSRowCount**, see section *tHDFSRowCount*.

tNamedPipeClose

tNamedPipeClose properties

Component family	File/Input	
Function	**tNamedPipeClose** closes a named-pipe opened with **tNamedPipeOpen** at the end of a process.	
Purpose	This component is used to close a named-pipe at the end of a process.	
Basic settings	*Pipe*	Select an existing named-pipe from the list to close.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is usually used to close a named-pipe at the end of a Job.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Writing and loading data through a named-pipe*.

tNamedPipeOpen

tNamedPipeOpen properties

Component family	File/Input	
Function	**tNamedPipeOpen** opens a named-pipe for writing data into it.	
Purpose	This component is used in inner-process communication, it opens a named-pipe for writing data into it.	
Basic settings	*Name*	Fill in the field with the name of the named-pipe.
	Delete if already exist	Select this checkbox to avoid duplicate named-pipe.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is usually used as the starting component in a inner-process communication Job.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Writing and loading data through a named-pipe*.

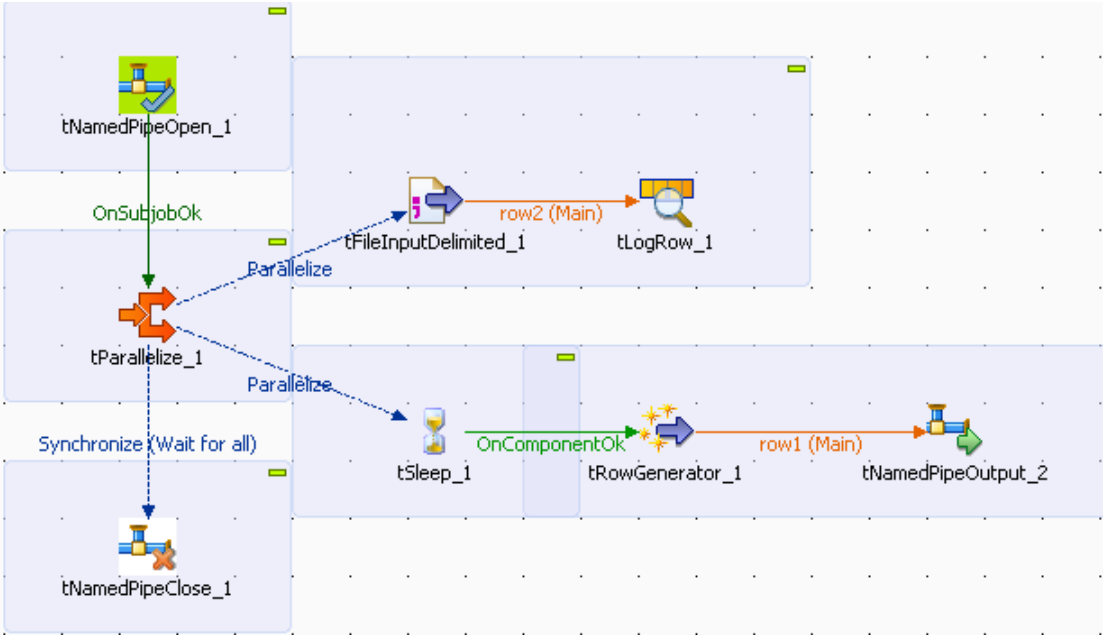
tNamedPipeOutput

tNamedPipeOutput properties

Component family	File/Input	
Function	**tNamedPipeOutput** writes data into an existing open named-pipe.	
Purpose	This component allows you to write data into an existing open named-pipe.	
Basic settings	*Use existing pipe connection*	Select this check box to use an existing named-pipe in the **Pipe component** list, or clear this check box to specify a named-pipe in **Pipe name** field.
	Pipe component	Select an existing named-pipe component from the list. This check box will display only when you select **Use existing pipe connection**.
	Pipe name	Fill in the field with the name of an existing named-pipe. This check box will display only when you clear **Use existing pipe connection**.
	Row separator	String (ex: “\n”on Unix) to distinguish rows in the output file.
	Field separator	Character, string or regular expression to separate fields of the output file.
	CSV options	Select this check box to take into account all parameters specific to CSV files, in particular **Escape char** and **Text enclosure** parameters.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Delete pipe if it exists	Select this checkbox to avoid duplicate named-pipe.
Advanced settings	*Boolean type*	Select a boolean type from the list.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is usually connected to another component in a subjob that reads data from a source.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Writing and loading data through a named-pipe

The following scenario creates a multi-component Job, which writes data into an open named-pipe and displays the data onto the console.



Dropping and linking the components

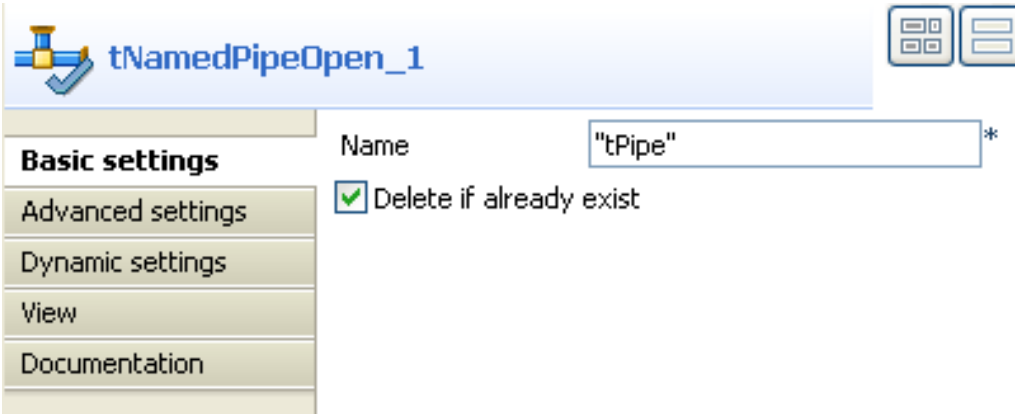
1. Drop the following components from the **Palette** to the design workspace: **tNamedPipeOpen**, **tParallelize**, **tNamedPipeClose**, **tFileInputDelimited**, **tSleep**, **tLogRow**, **tRowGenerator** and **tNamedPipeOutput**.
2. Connect **tNamedPipeOpen** to **tParallelize** using a **Trigger** > **OnSubjobOk** connection.
3. Connect **tParallelize** to **tFileInputDelimited** using a **Trigger** > **Parallelize** connection.
4. Connect **tParallelize** to **tSleep** using a **Trigger** > **Parallelize** connection.
5. Connect **tFileInputDelimited** to **tLogRow** using a **Row** > **Main** connection.
6. Connect **tParallelize** to **tNamedPipeClose** using a **Trigger** > **Synchronize (Wait for all)** connection.
7. Connect **tSleep** to **tRowGenerator** using a **Trigger** > **OnComponentOk** connection.
8. Connect **tRowGenerator** to **tNamedPipeOutput** using a **Row** > **Main** connection.

Configuring the components

Configuring the input component

1. Double-click **tNamedPipeOpen** to define its propeties in its **Basic settings** view.

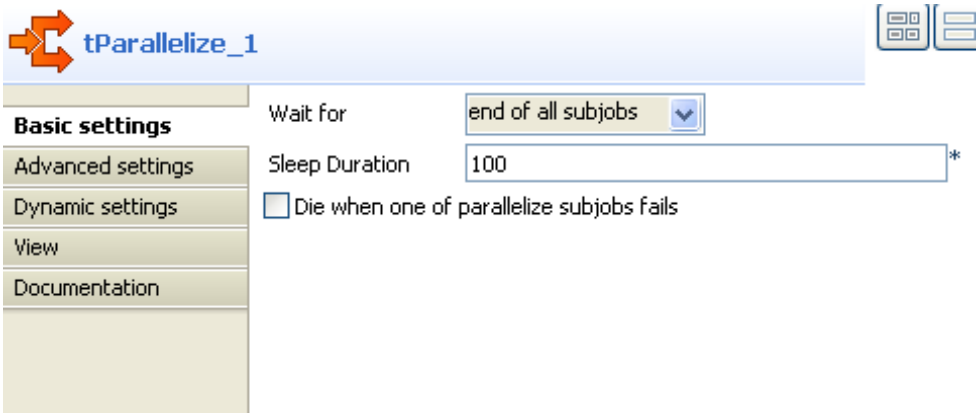
   Fill in the **Name** field with the name of a named-pipe and select **Delete if already exist** to avoid duplicate named-pipe.



2. Double-click **tParallelize** to define its properties in its **Basic settings** view.

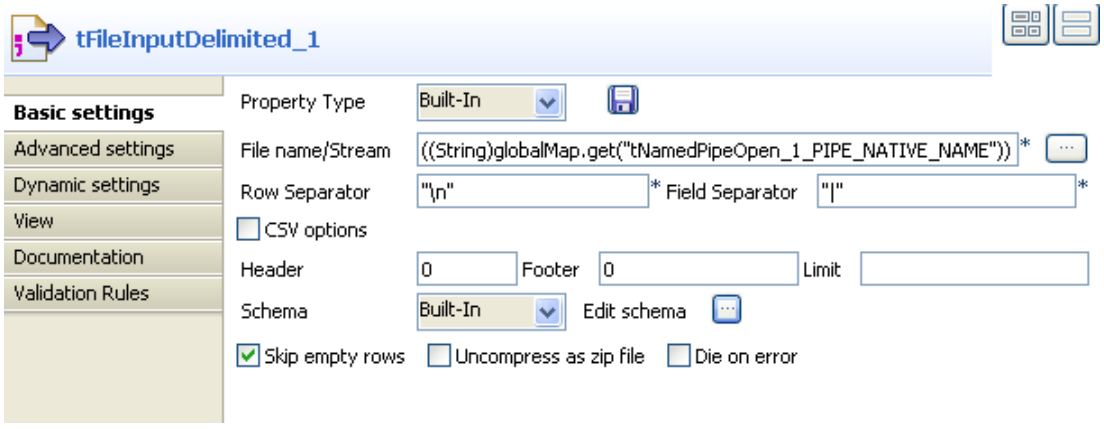
   Select **end of all subjobs** from the **Wait for** list.

   Fill in the **Sleep Duration** field with *100* to set the sleep duration.



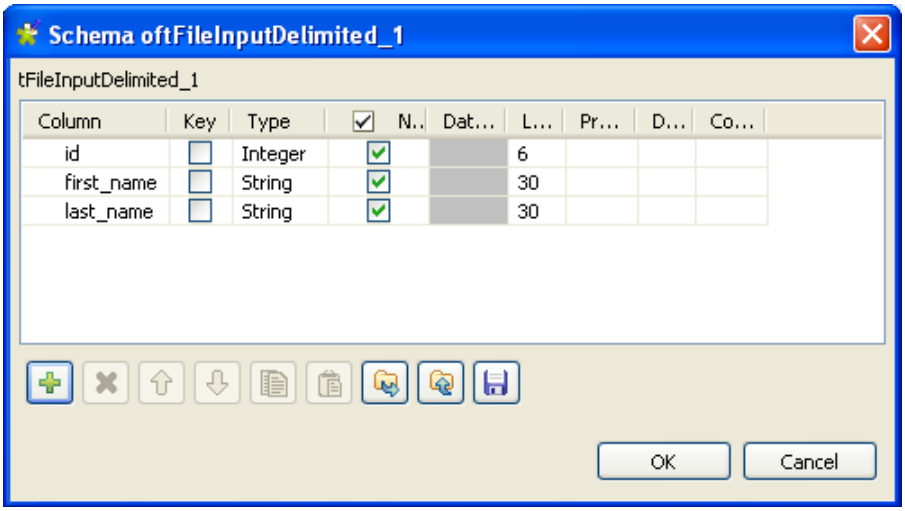
3. Double-click **tFileInputDelimited** to define its properties in its **Basic settings** view.

   Fill in the **File name/Stream** field with the following expression to use the name of the existing named-pipe defined in the **Basic settings** view of **tNamedPipeOpen**:

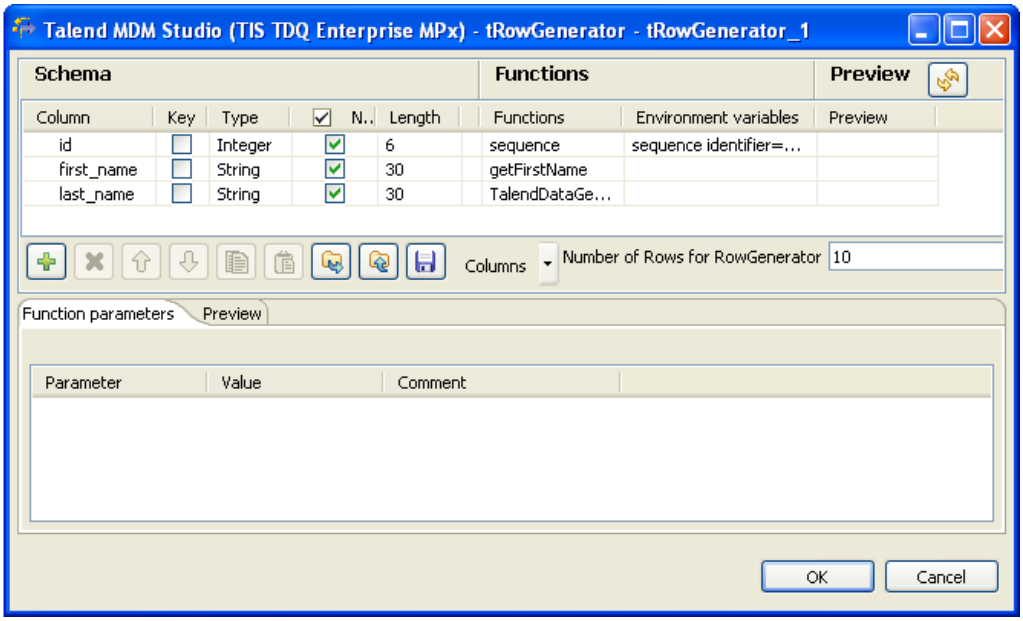


4. `((String)globalMap.get("tNamedPipeOpen_1_PIPE_NATIVE_NAME"))`

5. Click the three-dot button next to **Edit schema**.



6. Click the plus button to add three columns for **tFileInputDelimited**. Fill the three **Column** fields with *id*, *first_name* and *last_name* and set the **Type** of *id* to **Integer**. Keep the rest of the settings as default.
7. Click **OK** to save the settings for the schema.
8. Keep the rest of the settings in the **Basic settings** view of **tFileInputDelimited** as default.
9. Double-click **tSleep** and fill the **Pause (in seconds)** field with *1*.
10. Double-click **tRowGenerator** to define its properties in its **Basic settings** view.
11. Click **RowGenerator Editor** to define the schema.

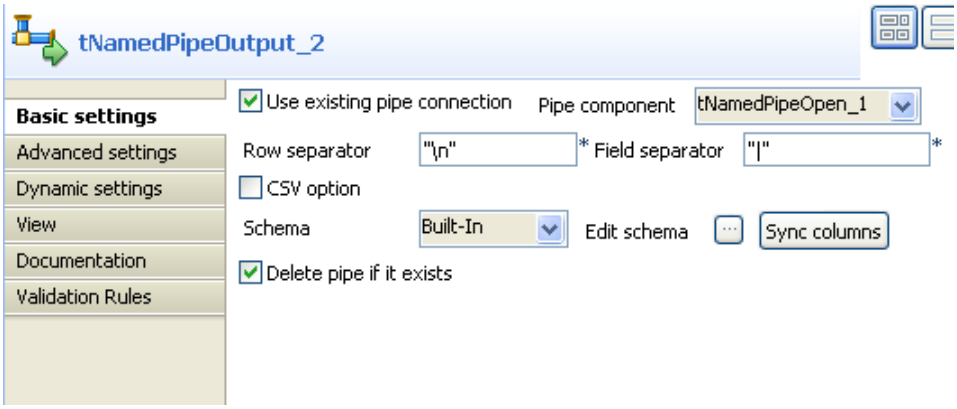


12. Click the plus button to add three columns for **tRowGenerator**. Fill the three **Column** fields with *id*, *first_name* and *last_name* and set the **Type** of *id* to Integer. Keep the rest of the settings of **Type** as default.
13. Select **sequence** from the list in the **Functions** field for *id*.
14. Select **getFirstName** from the list in the **Functions** field for Column *first_name*.
15. Select **TalendDataGenerator.getLastName** from the list in the **Functions** field for Column *last_name*.
16. Select *id*, fill the **Value** field under **Function parameters** tab with *s1* for **sequence identifier**, *1001* for **start value** and *1* for **step**.

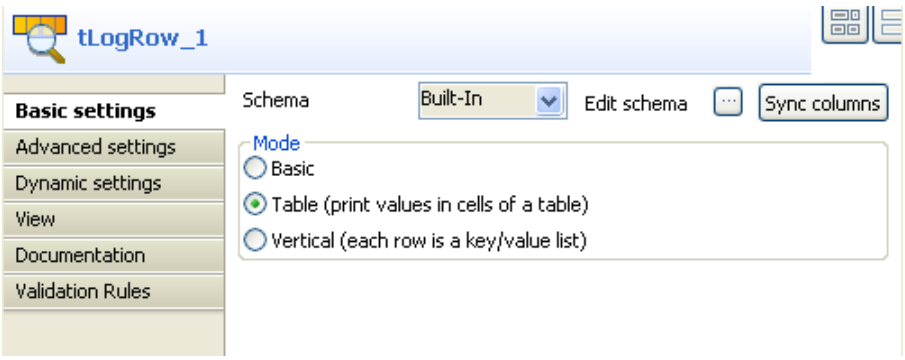
17. Click **OK** to save the settings.

Configuring the output component

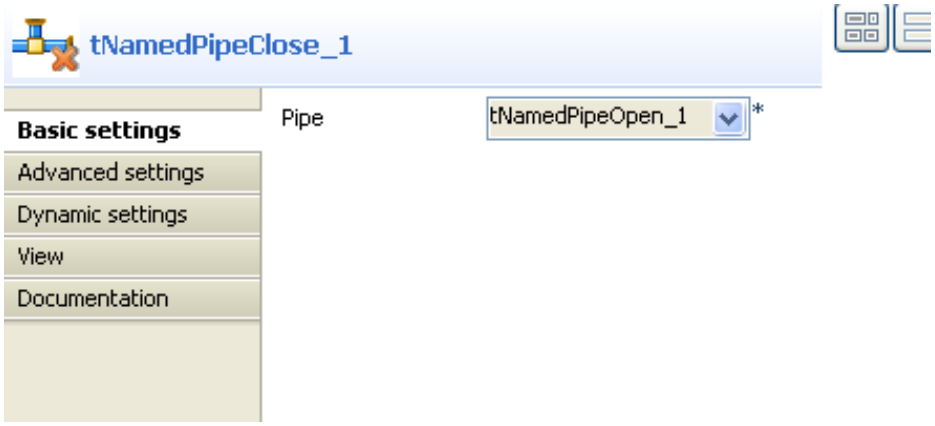
1. Double-click **tNamedPipeOutput** to define its properties in its **Basic settings** view.



2. Select the **Use existing pipe connection** checkbox and select **tNamedPipeOpen_1** from the **Pipe component** list.
3. Select **Delete pipe if it exists** to avoid duplicate named-pipe.
4. Click **Sync columns** to retrieve the schema from the preceding component.
5. Leave the rest of the settings as they are.
6. Double-click **tLogRow** to define its properties in its **Basic settings** view.



7. Click **Sync columns** to retrieve the schema from the preceding component.
8. Select **Table** in the **Mode** area.



9. Double-click **tNamedPipeClose** to define its properties in its **Basic settings** view.

10. Select **tNamedPipeOpen_1** from the **Pipe** list.

Saving and executing the Job

- Press **F6** to execute the Job.

```
Starting job namedPipe_Sample_01 at 18:13 19/05/2011.

[statistics] connecting to socket on port 3842
[statistics] connected
.----+----------+---------.
|        tLogRow_1         |
|=---+----------+--------=|
|id  |first_name|last_name|
|=---+----------+--------=|
|1001|Bill      |Cleveland|
'----+----------+---------'
[statistics] disconnected
Job namedPipe_Sample_01 ended at 18:13 19/05/2011.
[exit code=0]
```

The data written into the named-pipe is displayed onto the console.

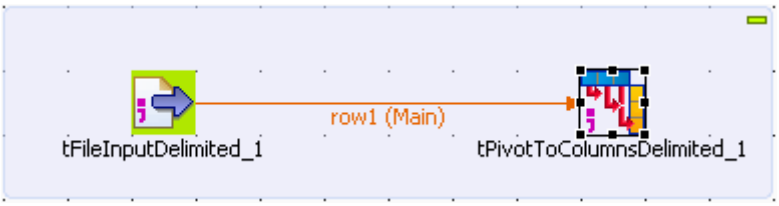
tPivotToColumnsDelimited

tPivotToColumnsDelimited Properties

Component family	File/Output	
Function	**tPivotToColumnsDelimited** outputs data based on an aggregation operation carried out on a pivot column.	
Purpose	**tPivotToColumnsDelimited** is used to fine-tune the selection of data to output	
Basic settings	*Pivot column*	Select the column from the incoming flow that will be used as pivot for the aggregation operation.
	Aggregation column	Select the column from the incoming flow that contains the data to be aggregated.
	Aggregation function	Select the function to be used in case several values are available for the pivot column.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	File Name	Name or path to the output file and/or the variable to be used. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Field separator	Character, string or regular expression to separate fields of the output file.
	Row separator	String (ex: “\n”on Unix) to distinguish rows in the output file.
Usage	This component requires an input flow.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Using a pivot column to aggregate data

The following scenario describes a Job that aggregates data from a delimited input file, using a defined pivot column.



Dropping and linking components

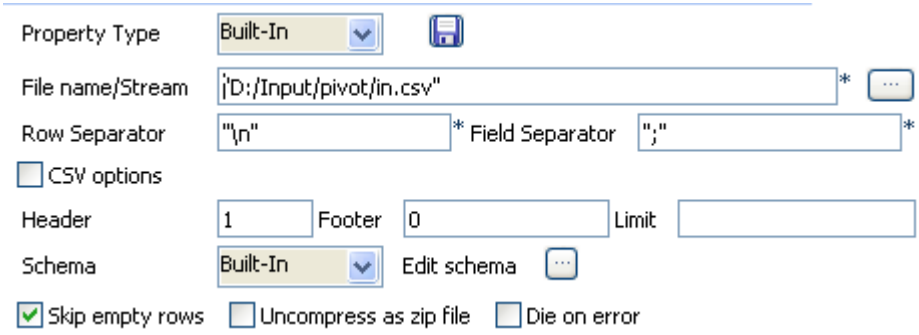
1. Drop the following component from the **Palette** to the design workspace: **tFileInputDelimited**, **tPivotToColumnsDelimited**.

2. Link the two components using a **Row** > **Main** connection.

Configuring the components

Set the input component

1. Double-click the **tFileInputDelimited** component to open its **Basic settings** view.



2. Browse to the input file to fill out the **File Name** field.

   The file to use as input file is made of 3 columns, including: *ID*, *Question* and the corresponding *Answer*

Id	Question	Answer
1	Name	Juan
2	Name	Jean
3	Name	John
1	Gender	M
2	Gender	F
3	Gender	M
1	Surgery	Yes
2	Surgery	No
3	Surgery	Yes
1	Age	45
2	Age	23
3	Age	42
1	Name	Mary

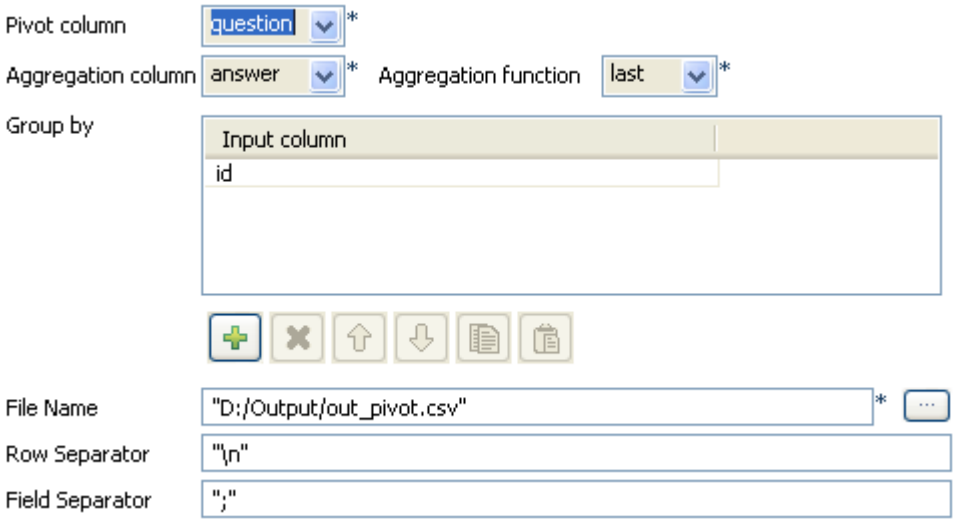
3. Define the **Row** and **Field** separators, in this example, respectively: carriage return and semi-colon

4. As the file contains a header line, define it also.

5. Set the schema describing the three columns: *ID*, *Questions*, *Answers*.

Set the output component

1. Double-click the **tPivotToColumnsDelimited** component to open its **Basic settings** view.



2. In the **Pivot column** field, select the pivot column from the input schema. this is often the column presenting most duplicates (pivot aggregation values).
3. In the **Aggregation column** field, select the column from the input schema that should gets aggregated.
4. In the **Aggregation function** field, select the function to be used in case duplicates are found out.
5. In the **Group by** table, add an Input column, that will be used to group by the aggregation column.
6. In the **File Name** field, browse to the output file path. And on the **Row** and **Field separator** fields, set the separators for the aggregated output rows and data.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

id	Name	Gender	Surgery	Age
1	Mary	M	Yes	45
2	Jean	M	No	23
3	John	F	Yes	42

The output file shows the newly aggregated data.

tSqoopExport

tSqoopExport component belongs to two component families: Big Data and File. For more information about **tSqoopExport**, see section *tSqoopExport*.

tSqoopImport

tSqoopImport component belongs to two component families: Big Data and File. For more information about **tSqoopImport**, see section *tSqoopImport*.

tSqoopImportAllTables

tSqoopImportAllTables component belongs to two component families: Big Data and File. For more information about **tSqoopImportAllTables**, see section *tSqoopImportAllTables*.

Internet components

This chapter details the main components which belong to the **Internet** family in the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Internet family comprises all of the components which help you to access information via the Internet, through various means including Web services, RSS flows, SCP, Emails, FTP etc.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tFileFetch

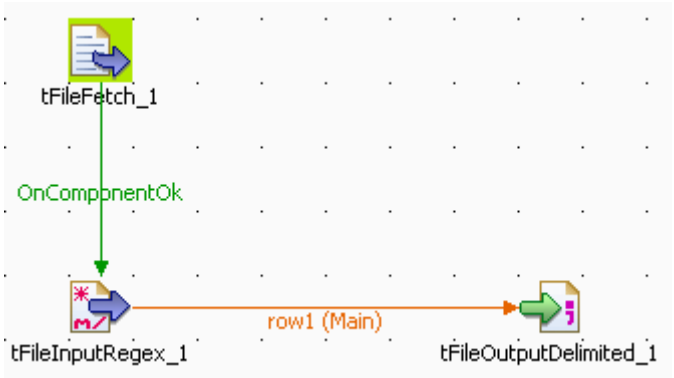
tFileFetch properties

Component family	Internet	
Function	**tFileFetch** retrieves a file via a defined protocol	
Purpose	**tFileFetch** allows you to retrieve file data according to the protocol which is in place.	
Basic settings	*Protocol*	Select the protocol you want to use from the list and fill in the corresponding fields: **http, https, ftp, smb**. The properties differ slightly depending on the type of protocol selected. The additional fields are defined in this table, after the basic settings.
	URI	Type in the URI of the site from which the file is to be fetched.
	Use cache to save resource	Select this check box to save the data in the cache. This option allows you to process the file data flow (in streaming mode) without saving it on your drive. This is faster and improves performance.
smb	*Domain*	Enter the Microsoft server domain name
smb	*Username and Password*	Enter the authentication information required to access the server.
	Destination Directory	Browse to the destination folder where the file fetched is to be placed.
	Destination Filename	Enter a new name for the file fetched.
http, https, ftp	*Create full path according to URI*	This check box is selected by default. It allows you to reproduce the URI directory path. To save the file at the root of your destination directory, clear the check box.
http, https	*Add header*	Select this check box if you want to add one or more HTTP request headers as fetch conditions. In the Headers table, enter the name(s) of the HTTP header parameter(s) in the Headers field and the corresponding value(s) in the Value field.
http, https	*POST method*	This check box is selected by default. It allows you to use the POST method. In the Parameters table, enter the name of the variable(s) in the Name field and the corresponding value in the Value field. Clear the check box if you want to use the GET method.
http, https, ftp	*Die on error*	Clear this check box to skip the rows in error and to complete the process for the error free rows
http, https, ftp, smb	*Read Cookie*	Select this check box for **tFileFetch** to load a web authentication cookie.
http, https, ftp, smb	*Save Cookie*	Select this check box to save the web page authentication cookie. This means you will not have to log on to the same web site in the future.
http, https, ftp, smb	*Cookie directory*	Click **[...]** and browse to where you want to save the cookie in your directory, or to where the cookie is already saved.
http, https, ftp, smb	*Cookie policy*	Choose a cookie policy from this drop-down list. Four options are available, i.e. BROWSER_COMPATIBILITY, DEFAULT, NETSCAPE and RFC_2109.
http, https, ftp, smb	*Single cookie header*	Check this box to put all cookies into one request header for maximum compatibility among different servers.

Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at each component level.
http, https	*Timeout*	Enter the number of seconds after which the protocol connection should close.
http, https	*Print response to console*	Select this check box to print the server response in the console.
http, https	*Upload file*	Select this check box to upload one or more files to the server. In the Name field, enter the name of the file you want to upload and in the File field, indicate the path.
http, https, ftp	*Enable proxy server*	Select this check box if you are connecting via a proxy and complete the fields which follow with the relevant information.
http, https	*Enable NTLM Credentials*	Select this check box if you are using an NTLM authentication protocol. **Domain**: The client domain name. **Host**: The client's IP address.
http, https	*Need authentication*	Select this check box and enter the username and password in the relevant fields, if they are required to access the protocol.
http, https	*Support redirection*	Select this check box to repeat the redirection request until redirection is successful and the file can be retrieved.
Usage	This component is generally used as a start component to feed the input flow of a Job and is often connected to the Job using an **OnSubjobOk** or **OnComponentOk** link, depending on the context.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario 1: Fetching data through HTTP

This scenario describes a three-component Job which retrieves data from an HTTP website and select data that will be stored in a delimited file.



Dropping and linking components

1. Drop a **tFileFetch**, a **tFileInputRegex** and a **tFileOutputDelimited** onto your design workspace.
2. Link **tFileFetch** to **tFileInputRegex** using a **Trigger** > **On Subjob Ok** or **On Component Ok** connection.
3. Link **tFileInputRegex** to **tFileOutputDelimited** using a **Row** > **Main** connection.

Configuring the components

1. In the **Basic settings** view of **tFileFetch**, select the protocol you want to use from the list. Here, use the HTTP protocol.
2. Type in the URI where the file to be fetched can be retrieved from.
3. In the **Destination directory** field, browse to the folder where the fetched file is to be stored.
4. In the **Filename** field, type in a new name for the file if you want it to be changed. In this example, *filefetch.txt*.
5. If needed, select the **Add header** check box and define one or more HTTP request headers as fetch conditions. For example, to fetch the file only if it has been modified since 19:43:31 GMT, October 29, 1994, fill in the **Name** and **Value** fields with "If-Modified-Since" and "Sat, 29 Oct 1994 19:43:31 GMT" respectively in the **Headers** table. For details about HTTP request header definitions, see Header Field Definitions.
6. Select the **tFileInputRegex**, set the **File name** so that it corresponds to the file fetched earlier.
7. Using a regular expression, in the **Regex** field, select the relevant data from the fetched file. In this example: <td(?: class="leftalign")?> \s* (t\w+) \s* </td>

   Regex syntaxe requires double quotation marks.
8. Define the **header**, **footer** and **limit** if need be. In this case, ignore these fields.
9. Define the schema describing the flow to be passed on to the final output.

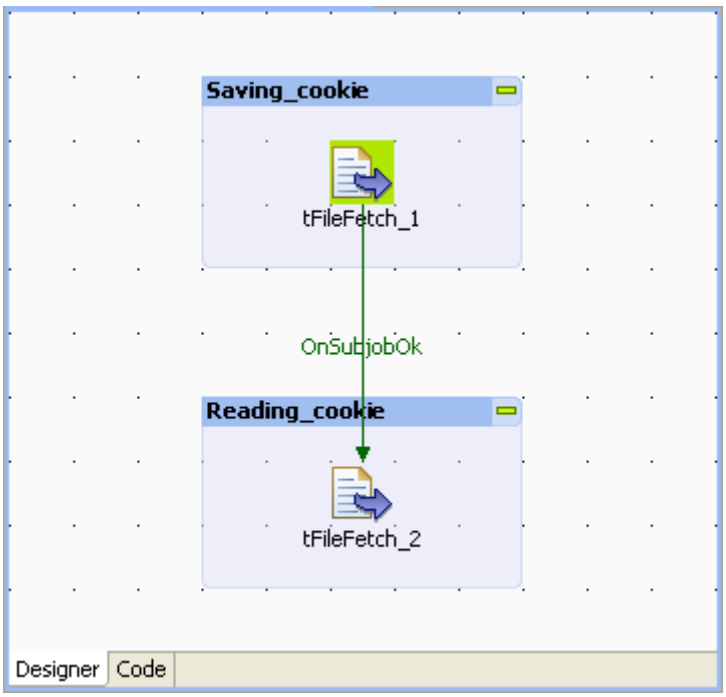
   The schema should be automatically propagated to the final output, but to be sure, check the schema in the **Basic settings** panel of the **tFileOutputDelimited** component.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Then press **F6** or click **Run** on the **Run** tab to execute the Job.

Scenario 2: Reusing stored cookie to fetch files through HTTP

This scenario describes a two-component Job which logs in a given HTTP website and then using cookie stored in a user-defined local directory, fetches data from this website.



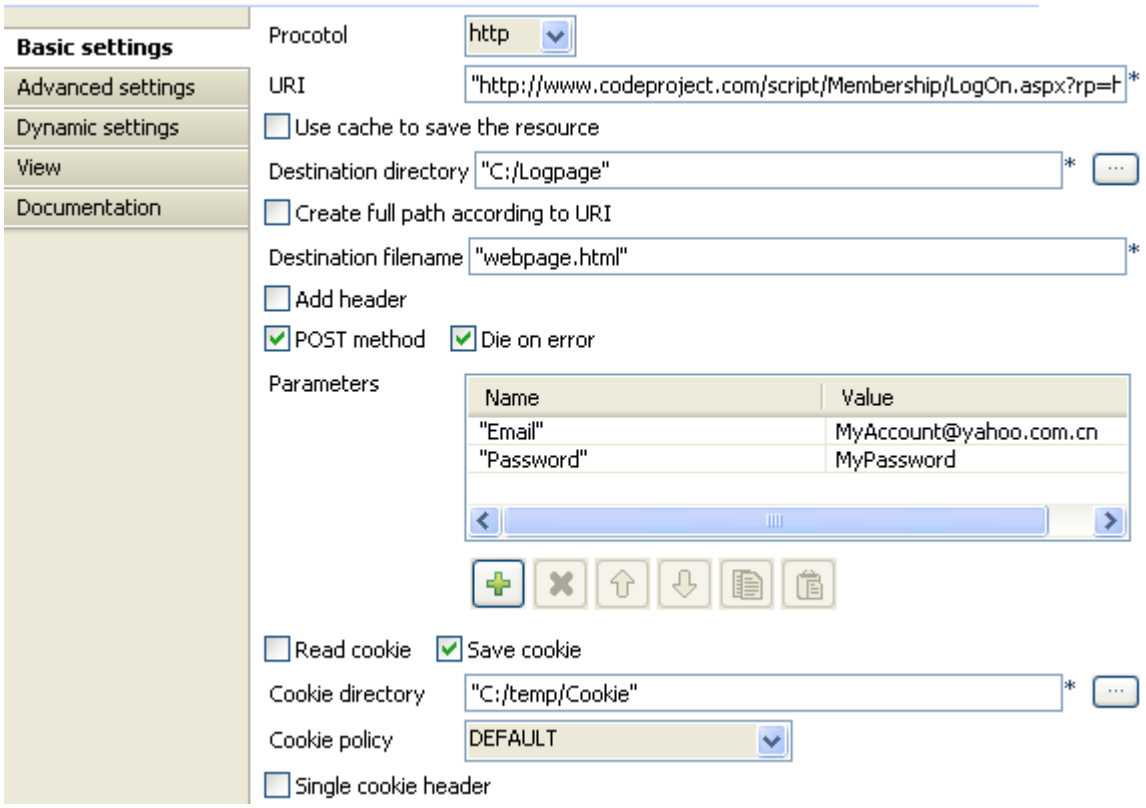
Dropping and linking components

1. Drop two **tFileFetch** components onto your design workspace.
2. Link the two components as subjobs using a **Trigger** > **On Subjob Ok** connection.

Configuring the components

Configuring the first subjob

1. Double click **tFileFetch_1** to open its component view.



2. In the **Procotol** field, select the protocol you want to use from the list. Here, we use the *HTTP* protocol.

3. In the **URI** field, type in the URI through which you can log in the website and fetch the web page accordingly. In this example, the URI is `http://www.codeproject.com/script/Membership/LogOn.aspx?rp=http%3a%2f%2fwww.codeproject.com%2fKB%2fcross-platform%2fjavacsharp.aspx&download=true`.

4. In the **Destination directory** field, browse to the folder where the fetched file is to be stored. This folder will be created on the fly if it does not exist. In this example, type in *C:/Logpage*.

5. In the **Destination Filename** field, type in a new name for the file if you want it to be changed. In this example, *webpage.html*.

6. Under the **Parameters** table, click the plus button to add two rows.

7. In the **Name** column of the **Parameters** table, type in a new name respectively for the two rows. In this example, they are *Email* and *Password*, which are required by the website you are logging in.

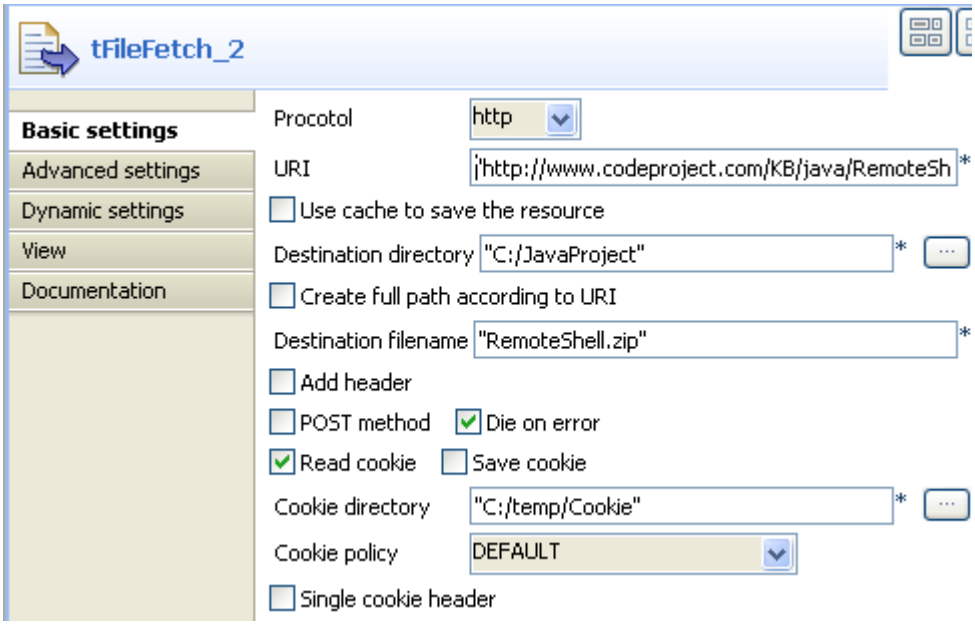
8. In the **Value** column, type in the authentication information.

9. Select the **Save cookie** check box to activate the **Cookie directory** field.

10. In the **Cookie directory** field, browse to the folder where you want to store cookie file and type in a name for the cookie to be saved. This folder must exist already. In this example, the directory is *C:/temp/Cookie*.

Configuring the second subjob

1. Double click **tFileFetch_2** to open its **Component** view.



2. In the **Procotol** list, select **http**.
3. In the **URI** field, type in the address from which you fetch the files of your interest. In this example, the address is `http://www.codeproject.com/KB/java/RemoteShell/RemoteShell.zip`.
4. In the **Destination directory** field, type in the directory or browse to the folder where you want to store the fetched files. This folder can be automatically created if it does not exist yet during the execution process. In this example, type in *C:/JavaProject*.
5. In the **Destination Filename** field, type in a new name for the file if you want it to be changed. In this example, *RemoteShell.zip*.
6. Clear the **Post method** check box to deactivate the **Parameter** table.
7. Select the **Read cookie** check box to activate the **Cookie directory** field.
8. In the **Cookie directory** field, type in the directory or browse to the cookie file you have saved and need to use. In this example, the directory is *C:/temp/Cookie*.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Then press **F6** to run the Job, and check each folder you have used to store the fetched files.

Related scenario

For an example of transferring data in streaming mode, see section *Scenario 2: Reading data from a remote file in streaming mode*

tFileInputJSON

tFileInputJSON belongs to two different component families: Internet and File. For further information, see section *tFileInputJSON*.

tFTPConnection

tFTPConnection properties

Component family	Internet/FTP	
Function	**tFTPConnection** opens an FTP connection in order that a transaction may be carried out.	
Purpose	**tFTPConnection** allows you to open an FTP connection to transfer files in a single transaction.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	The FTP server IP address.
	Port	The FTP server listening port number.
	Username and Password	FTP user authentication data.
	SFTP Support	When you select this check box, the **Authentication method** appears. It offers two means of authentication: **Public key**: Enter the access path to the public key. **Password**: Enter the password.
	FTPS Support	Select this check box to connect to an FTP server via an FTPS connection. Two fields appear: **Keystore file**: Enter the access path to the keystore file (password protected file containing several keys and certificates). **Keystore Password**: Enter your keystore password.
	Connect mode	Select the mode: **Active** or **Passive**
Usage	This component is typically used as a single-component sub-job. It is used along with other FTP components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

For a related scenario, see section *Scenario: Putting files on a remote FTP server*.

For a related scenario, see section *Scenario: Iterating on a remote directory*.

For a related scenario using a different protocol, see section *Scenario: Getting files from a remote SCP server*.

tFTPDelete

tFTPDelete properties

Component family	Internet/FTP	
Function	This component deletes specified files via an FTP connection.	
Purpose	**tFTPDelete** deletes files on a remote FTP server.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Host	FTP IP address
	Port	The FTP server listening port number.
	Username and *Password*	FTP user authentication data.
	Remote directory	Source directory where the files to be deleted are located.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Use Perl5 Regex Expression as Filemask	Select this check box if you want to use Perl5 regular expressions in the **Files** field as file filters. For information about Perl5 regular expression syntax, see Perl5 Regular Expression Syntax.
	Files	File name or path to the files to be deleted. You can specify multiple files in a line by using wildcards or a regular expression.
Usage	This component is typically used as a single-component sub-job but can also be used as an output or end object.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For **tFTPDelete** related scenario, see section *Scenario: Putting files on a remote FTP server*.

For **tFTPDelete** related scenario using a different protocol, see section *Scenario: Getting files from a remote SCP server*.

tFTPFileExist

tFTPFileExist properties

Component family	Internet/FTP	
Function	**tFTPFileExist** checks if a file exists on an FTP server.	
Purpose	**tFTPFileExist** allows you to check if a file exists on an FTP server.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. In this case, make sure that the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Host	FTP IP address.
	Port	The FTP server listening port number.
	Username and *Password* (or *Private key*)	User authentication information.
	Remote directory	Path to the remote directory.
	File Name	Name of the file you want to check exists.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Connection Mode	Select the SFTP connection mode you want to use: **Active**: You determine the connection port to use to allow data transfer. **Passive**: the FTP server determines the connection port to use to allow data transfer.
	Encoding Type	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
Advanced settings	Use Socks Proxy	Select this check box if you want to use a proxy. Then, set the **Host**, **Port**, **User** and **Password** proxy fields.
	Ignore Failure At Quit (FTP)	Select this check box to ignore library closing errors or FTP closing errors.

	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For **tFTPFileExist** related scenario, see section *Scenario: Putting files on a remote FTP server*.

For **tFTPFileExist** related scenario using a different protocol, see section *Scenario: Getting files from a remote SCP server*.

tFTPFileList

tFTPFileList properties

Component family	Internet/FTP	
Function	**tFTPFileList** iterates on files and/or folders of a given directory on a remote host.	
Objective	**tFTPFileList** retrieves files and /or folders based on a defined filemask pattern and iterates on each of them by connecting to a remote directory via an FTP protocol.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. For more information about **Dynamic settings**, see your studio user guide.
	Host	FTP IP address.
	Port	Listening port number of the FTP server.
	Username and *Password* (or *Private key*)	User authentication information.
	Remote directory	Path to the remote directory.
	File detail	Select this check box if you want to display the details of each of the files or folders on the remote host. These informative details include: type of rights on the file/folder, name of the author, name of the group of users that have a read-write rights, file size and date of last modification.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Files	Click the plus button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions.
	Connect Mode	Select the SFTP connection mode you want to use: **Active**: You determine the connection port to be used to allow data transfer.

		Passive: the FTP server determines the connection port to use to allow data transfer.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Iterating on a remote directory

The following scenario describes a three-component Job that connects to an FTP server, lists files held in a remote directory based on a filemask and finally recuperates and saves the files in a defined local directory.

Dropping and linking components

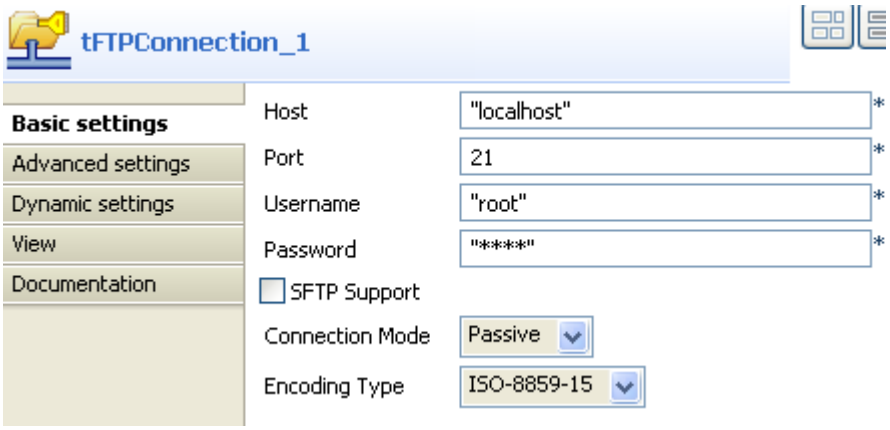
1. Drop the following components from the Palette to the design workspace: **tFTPConnection**, **tFTPFileList** and **tFTPGet**.

2. Link **tFTPConnection** to **tFTPFileList** using an **OnSubjobOk** connection and then **tFTPFileList** to **tFTPGet** using an **Iterate** connection.

Configuring the components

Configuring a connection to the FTP server

1. Double-click **tFTPConnection** to display its **Basic settings** view and define the component properties.

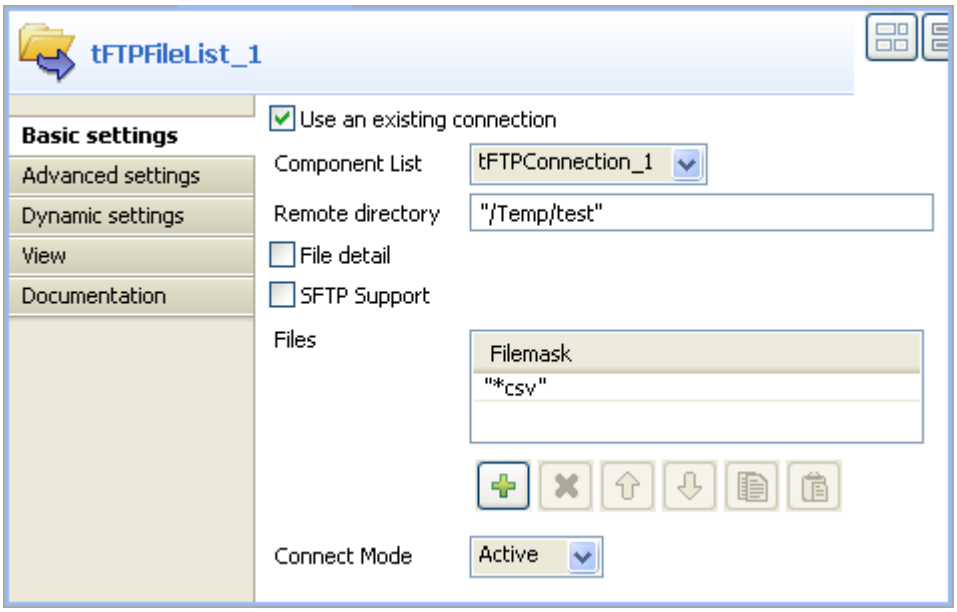


2. In the **Host** field, enter the IP address of the FTP server.
3. In the **Port** field, enter the listening port number.
4. In the **Username** and **Password** fields, enter your authentication information for the FTP server.

5. In the **Connect Mode** list, select the FTP connection mode you want to use, **Passive** in this example.

Configuring an FTP download list

1. Double-click **tFTPFileList** to open its **Basic settings** view and define the component properties.



2. Select the **Use an existing connection** check box and in the **Component list**, click the relevant FTP connection component, **tFTPConnection_1** in this scenario. Connection information are automatically filled in.

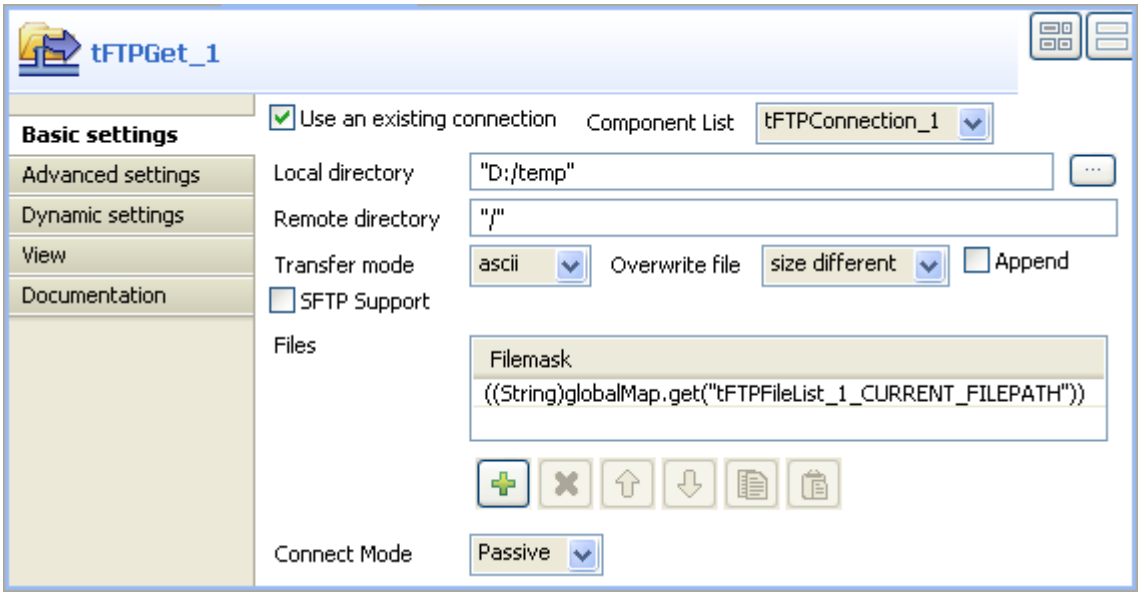
3. In the **Remote directory** field, enter the relative path of the directory that holds the files to be listed.

4. In the **Filemask** field, click the plus button to add one line and then define a file mask to filter the data to be retrieved. You can use special characters if need be. In this example, we want only to recuperate delimited files (**csv*).

5. In the **Connect Mode** list, select the FTP server connection mode you want to use, **Active** in this example.

Configuring file download

1. Double-click **tFTPGet** to display its **Basic settings** view and define the components properties.



2. Select the **Use an existing connection** check box and in the **Component list**, click the relevant FTP connection component, **tFTPConnection_1** in this scenario. Connection information are automatically filled in.

3. In the **Local directory** field, enter the relative path for the output local directory where you want to write the recuperated files.

4. In the **Remote directory** field, enter the relative path of the remote directory that holds the file to be recuperated.

5. In the **Transfer Mode** list, select the FTP transfer mode you want to use, **ascii** in this example.

6. In the **Overwrite file** field, select an option for you want to use for the transferred files.

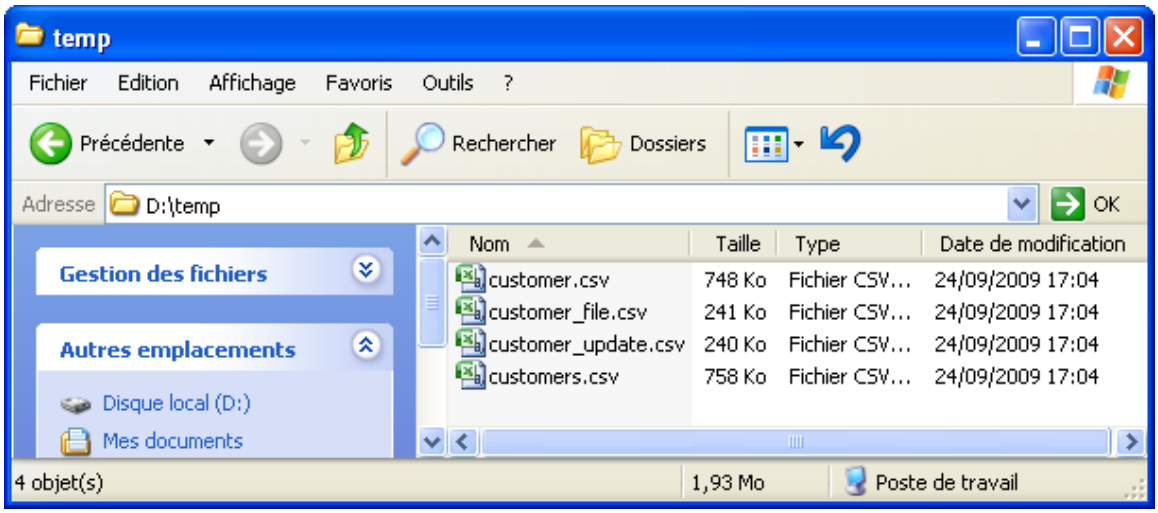
7. In the **Files** area, click the plus button to add a line in the **Filemask** list, then click in the added line and press**Ctrl+Space** to access the variable list. In the list, select the global variable `((String)globalMap.get("tFTPFileList_1_CURRENT_FILEPATH"))` to process all files in the remote directory.

8. In the **Connect Mode** list, select the connection mode to the FTP server you want to use.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.



All .csv files held in the remote directory on the FTP server are listed in the defined directory, as defined in the filemask. Then the files are retrieved and saved in the defined local output directory.

tFTPFileProperties

tFTPFileProperties Properties

Component family	Internet	
Function	**tFTPFileProperties** iterates on files and/or folders of a given directory on a remote host.	
Purpose	**tFTPFileProperties** retrieves files and /or folders based on a defined filemask pattern and iterates on each of them by connecting to a remote directory via an FTP protocol.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Host	FTP IP address
	Port	Listening port number of the FTP server.
	Username	FTP user name.
	Password	FTP password.
	Remote directory	Path to the source directory where the files can be fetched.
	File	Name or path to the file to be processed. Related topic: see *Talend Open Studio for Big Data User Guide*.
	SFTP Support and *Authentication method*	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, ensure that the key is added to the agent or that no passphrase (secret phrase) is required. If you do not select the check box, choose the connection mode you want to use: **Active**: You determine the connection port to use to allow data transfer. **Passive**: the FTP server determines the connection port to use to allow data transfer.
	Encoding	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Calculate MD5 Hash	Select this check box to check the of the downloaded file's MD5.
Advanced settings	Use Socks Proxy	Select this check box if you want to use a proxy. Then, set the **Host**, **Port**, **User** and **Password** proxy fields.
	Ignore Failure At Quit (FTP)	Select this check box to ignore library closing errors or FTP closing errors.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.

Usage	This component can be used as standalone component.
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Related scenario

For a related scenario, see section *Scenario: Displaying the properties of a processed file*

tFTPGet

tFTPGet properties

Component family	Internet/FTP	
Function	This component retrieves specified files via an FTP connection.	
Purpose	**tFTPGet** retrieves selected files from a defined remote FTP directory and cop them to a local directory.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	Select this check box and then choose the appropriate connection component from the **Component list** to reuse its connection parameters.
	Host	FTP IP address.
	Port	Listening port number of the FTP server.
	Username	FTP user name.
	Password	FTP password.
	Local directory	Path to where the file is to be saved locally.
	Remote directory	Path to source directory where the files can be fetched.
	Transfer mode	Different FTP transfer modes.
	Overwrite file	List of file transfer options. **Append**: Select this check box to append the data at the end of the file in order to avoid overwriting data.
	SFTP Support	When you select this check box, the **Overwrite file** and **Authentication method** appear. **Overwrite file**: Offers three options: **Overwrite**: Overwrite the existing file. **Resume**: Resume downloading the file from the point of interruption. **Append**: Add data to the end of the file without overwriting data. **Authentication** Offers two means of authentication: **Public key**: Enter the access path to the public key. **Password**: Enter the password.
	FTPS Support	Select this check box to connect to an FTP server via an FTPS connection. Two fields appear: **Keystore file**: Enter the access path to the keystore file (password protected file containing several keys and certificates). **Keystore Password**: Enter your keystore password.
	Use Perl5 Regex Expression as Filemask	Select this check box if you want to use Perl5 regular expressions in the **Files** field as file filters.

		For information about Perl5 regular expression syntax, see Perl5 Regular Expression Syntax.
	Files	File names or paths to the files to be transferred. You can specify multiple files in a line by using wildcards or a regular expression.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
	Print message	Select this check box to display in the Console the list of files downloaded.
Usage	This component is typically used as a single-component sub-job but can also be used as output or end object.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For an **tFTPGet** related scenario, see section *Scenario: Putting files on a remote FTP server*.

For an **tFTPGet** related scenario, see section *Scenario: Iterating on a remote directory*.

For an **tFTPGet** related scenario using a different protocol, see section *Scenario: Getting files from a remote SCP server*.

tFTPPut

tFTPPut properties

Component family	Internet/FTP	
Function	This component copies selected files via an FTP connection.	
Purpose	**tFTPPut** copies selected files from a defined local directory to a destination remote FTP directory.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	A connection needs to be open to allow the loop to check for FTP data on the defined DB. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. For more information about **Dynamic settings**, see your studio user guide.
	Host	FTP IP address.
	Port	FTP server listening port number.
	Username	FTP user name.
	Password	FTP password.
	Local directory	Path to the source location of the file(s).
	Remote directory	Path to the destination directory of the file(s).
	Transfer mode	Different FTP transfer modes.
	Overwrite file or Append	List of available options for the transferred file
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Files	Click the **[+]** button to add a new line, then fill in the columns. **Filemask**: file names or path to the files to be transferred. **New name**: name to give the FTP file after the transfer.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is typically used as a single-component sub-job but can also be used as output component.	

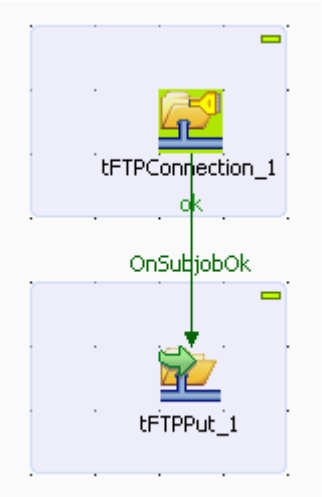
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.

Scenario: Putting files on a remote FTP server

This two-component Job allows you to open a connection to a remote FTP server in order to put specific files on the remote server in one transaction.

Dropping and linking components

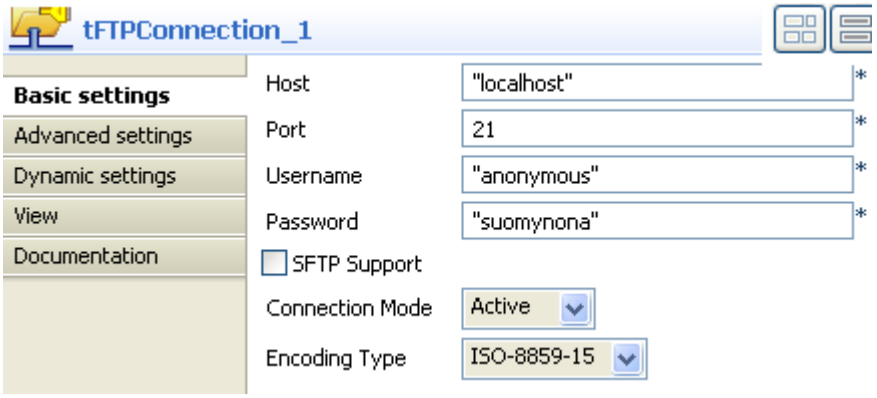
1. Drop **tFTPConnection** and **tFTPPut** from the **Palette** onto the design workspace. **tFTPConnection** allows you to perform all operations in one transaction.
2. Connect the two components together using an **OnSubJobOK** link.



Configuring the components

Configuring a connection to the FTP server

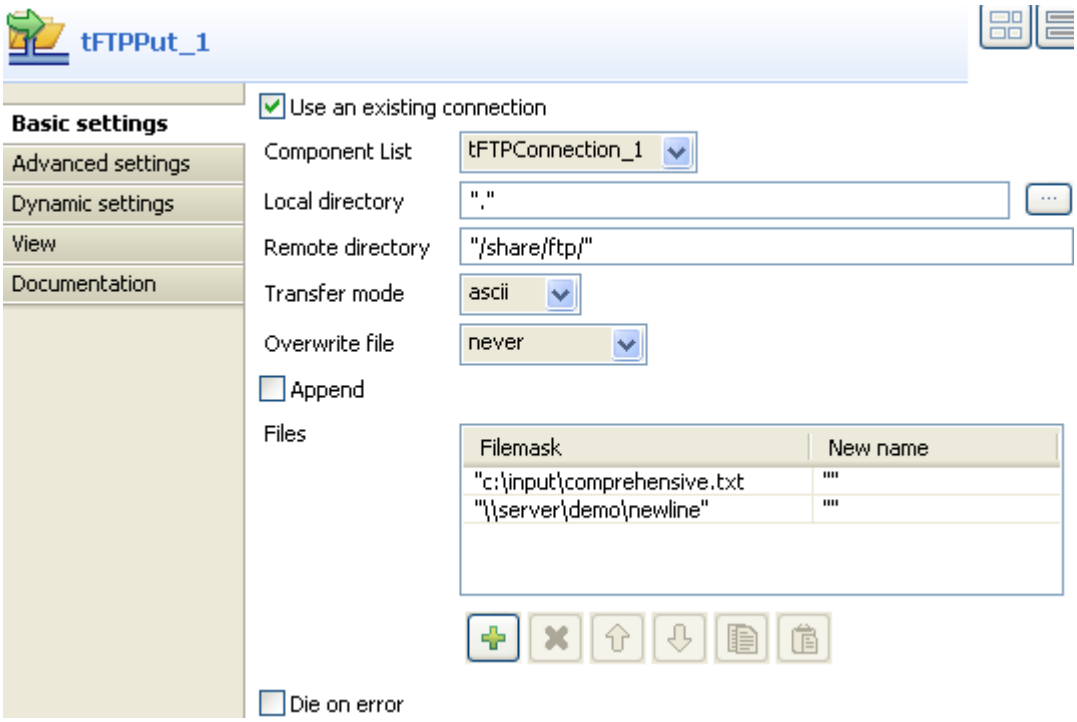
1. Double-click **tFTPConnection** to display its **Basic settings** view and define its properties.



2. In the **Host** field, enter the server IP address.
3. In the **Port** field, enter the listening port number.
4. In the **Username** and **Password** fields, enter your login and password for the remote server.
5. From the **Connect Mode** list, select the FTP connection mode you want to use, **Active** in this example.

Configuring file upload to the FTP server

1. In the design workspace, double-click **tFTPPut** to display its **Basic settings** view and define its properties.



2. Select the **Use an existing connection** check box and then select **tFTPConnection_1** from the **Component List**. The connection information is automatically filled in.
3. In the **Local directory** field, enter the path to the local directory containing the files, if all your files are in the same directory. If the files are in different directories, enter the path for each file in the **Filemask** column of the **Files** table.
4. In the **Remote directory** field, enter the path to the destination directory on the remote server.
5. From the **Transfer mode** list, select the transfer mode to be used.
6. From the **Overwrite file** list, select an option for the transferred file.
7. In the **Files** table, click twice the plus button to add two lines to the **Filemask** column and then fill in the filemasks of all files to be copied onto the remote directory.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

   The files specified in the **Filemask** column are copied to the remote server.

tFTPRename

tFTPRename Properties

Component Family	Internet/FTP	
Function	**tFTPRename** renames the selected files via an FTP connection.	
Purpose	**tFTPRename** renames files selected from a local directory towards a distant FTP directory.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. For more information about **Dynamic settings**, see your studio user guide.
	Host	FTP IP address.
	Port	FTP server listening port number.
	Username	Connection login to the FTP server.
	Password	Connection password to the FTP server.
	Remote directory	Path to the remote directory.
	Overwrite file	List of available options for the transferred file. **Append**: Select this check box to write the data at the end of the record, to not delete it.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Files	Click the **[+]** button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions. **New name**: name to give to the FTP file after the transfer.
	Connection Mode	Select the SFTP connection mode you want to use: **Active**: You determine the connection port to use to allow data transfer. **Passive**: the FTP server determines the connection port to use to allow data transfer.

	Encoding type	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Die on error	This check box is selected by default. Clear the check box to skip the row in error and complete the process for error-free rows.
Advanced settings	Use Socks Proxy	Select this check box if you want to use a proxy. Then, set the **Host**, **Port**, **User** and **Password** proxy fields.
	Ignore Failure At Quit (FTP)	Select this check box to ignore library closing errors or FTP closing errors.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as a subjob with one component, but it can also be used as an output or end component..	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Putting files on a remote FTP server* .

tFTPTruncate

tFTPTruncate properties

Component family	Internet/FTP	
Function	**tFTPTruncate** truncates the selected files via an FTP connection.	
Objective	**tFTPTruncate** truncates the selected files of a defined local directory via a distant FTP directory.	
Basic settings	*Property type*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally.
	Use an existing connection/ Component List	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can use **Dynamic settings** to share the intended connection. For more information about **Dynamic settings**, see your studio user guide.
	Host	FTP IP address.
	Port	Listening port number of the FTP server.
	Username and *Password* (or *Private key*)	User authentication information.
	Remote directory	Path to the remote directory.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it. If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Use Perl5 Regex Expression as Filemask	Select this check box if you want to use Perl5 regular expressions in the **Files** field as file filters. For information about Perl5 regular expression syntax, see Perl5 Regular Expression Syntax.
	Files	Click the plus button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcards (*) or regular expressions.
	Connection Mode	Select the SFTP connection mode you want to use: **Active**: You determine the connection port to use to allow data transfer. **Passive**: the FTP server determines the connection port to use to allow data transfer.

	Encoding type	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
Advanced settings	*Use Socks Proxy*	Select this check box if you want to use a proxy. Then, set the **Host**, **Port**, **User** and **Password** proxy fields.
	Ignore Failure At Quit (FTP)	Select this check box to ignore library closing errors or FTP closing errors.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Putting files on a remote FTP server*.

tHttpRequest

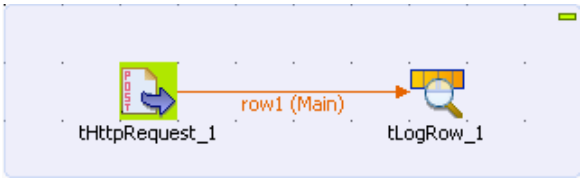
tHttpRequest properties

Component family	Internet	
Function	This component sends an HTTP request to the server end and gets the corresponding response information from the server end.	
Purpose	The **tHttpRequest** component allows you to send an HTTP request to the server and output the response information locally.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Sync columns	Click this button to retrieve the schema from the preceding component.
	URI	Type in the Uniform Resource Identifier (URI) that identifies the data resource on the server. A URI is similar to a URL, but more general.
	Method	Select an HTTP method to define the action to be performed: **Post**: Sends data (e.g. HTML form data) to the server end. **Get**: Retrieves data from the server end.
	Write response content to file	Select this check box to save the HTTP response to a local file. You can either type in the file path in the input field or click the three-dot button to browse to the file path.
	Headers	Type in the name-value pair(s) for HTTP headers to define the parameters of the requested HTTP operation. **Key**: Fill in the name of the header field of an HTTP header. **Value**: Fill in the content of the header field of an HTTP header. For more information about definition of HTTP headers, please refer to: en.wikipedia.org/wiki/List_of_HTTP_headers.
	Need authentication	Select this check box to fill in a user name and a password in the corresponding fields if authentication is needed: **user**: Fill in the user name for the authentication. **password**: Fill in the password for the authentication.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level and at each component level.
Usage	This component can be used in sending HTTP requests to server and saving the response information. This component can be used as a standalone component.	
Limitation	N/A	

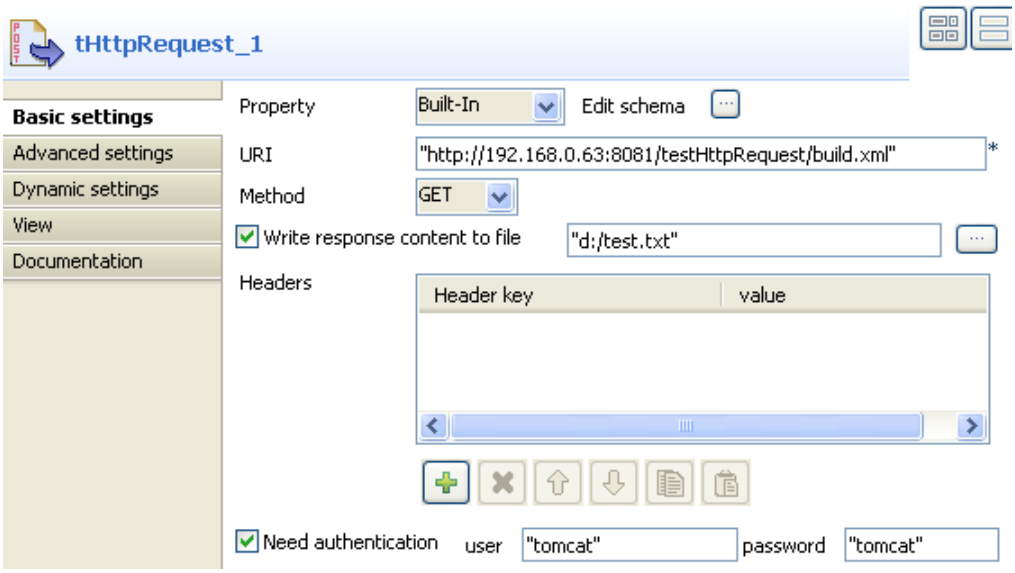
Scenario: Sending a HTTP request to the server and saving the response information to a local file

This java scenario describes a two-component Job that uses the **GET** method to retrieve information from the server end and writes the response to a local file as well as to the console.

- Drop the following components from the **Palette** onto the design workspace: **tHttpRequest** and **tLogRow**.



- Connect the **tHttpRequest** component to the **tLogRow** component using a **Row** > **Main** connection.
- Double-click the **tHttpRequest** component to open its **Basic settings** view and define the component properties.



- Fill in the **URI** field with *"http://192.168.0.63:8081/testHttpRequest/build.xml"*. Note that this URI is for demonstration purpose only and it is not a live address.
- Select **GET** from the **Method** list.
- Select the **Write response content to file** check box and fill in the input field on the right with the file path by manual entry, *D:/test.txt* for this use case.
- Select the **Need authentication** check box and fill in the user and password, both *tomcat* in this use case.
- Double-click the **tLogRow** component to open its **Basic settings** view and select **Table** in the **Mode** area.
- Save your Job and press **F6** to execute it.

  Then the response information from the server is saved and displayed.

```
Starting job test at 15:31 07/03/2011.

[statistics] connecting to socket on port 4011
[statistics] connected
.---------------.
|   tLogRow_1   |
|=--------------=|
|ResponseContent|
|=--------------=|
|hello world!   |
'---------------'

[statistics] disconnected
Job test ended at 15:31 07/03/2011. [exit
code=0]
```

tJMSInput

tJMSInput properties

Component Family	Internet	
Function	**tJMSInput** creates an interface between a Java application and a Message-Oriented middle ware system.	
Purpose	Using a JMS server, **tJMSInput** makes it possible to have loosely coupled, reliable, and asynchronous communication between different components in a distributed application.	
Basic settings	*Module List*	Select the library to be used from the list.
	Context Provider	Type in the context URL, for example "com.tibco.tibjms.naming.TibjmsInitialContext Factory". However, be careful, the syntax can vary according to the JMS server used.
	Server URL	Type in the server URL, respecting the syntax, for example "tibjmsnaming://localhost:7222".
	Connection Factory JDNI Name	Type in the JDNI name.
	Use Specified User Identity	If you have to log in, select the check box and type in your login and password.
	Message Type	Select the message type, either: **Topic** or **Queue**.
	Message From	Type in the message source, exactly as expected by the server; this must include the type and name of the source. e.g.: queue/A or topic/testtopic Note that the field is case-sensitive.
	Timeout for Next Message (in sec)	Type in the number of seconds before passing to the next message.
	Maximum Messages	Type in the maximum number of messages to be processed.
	Message Selector Expression	Set your filter.
	Processing Mode	Select the processing mode for the messages. **Raw Message** or **Message Content**
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The **tJMSInput** schema is read-only. It is made of only one column: **Message**
Advanced settings	*Properties*	Click the plus button underneath the table to add lines that contains username and password required for user authentication.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as an input component. It must be linked to an output component.	
Limitation	Make sure the JMS server is launched. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

No scenario is available for this component yet.

tJMSOutput

tJMSOutput properties

Component Family	Internet	
Function	**tJMSOutput** creates an interface between a Java application and a Message-Oriented middle ware system.	
Purpose	Using a JMS server, **tJMSOutput** makes it possible to have loosely coupled, reliable, and asynchronous communication between different components in a distributed application.	
Basic settings	*Module List*	Select the library to be used from the list.
	Context Provider	Type in the context URL, for example "com.tibco.tibjms.naming.TibjmsInitialContext Factory". However, be careful, the syntax can vary according to the JMS server used.
	Server URL	Type in the server URL, respecting the syntax, for example "tibjmsnaming://localhost:7222".
	Connection Factory JDNI Name	Type in the JDNI name.
	Use Specified User Identity	If you have to log in, select the check box and type in your login and password.
	Message Type	Select the message type, either: **Topic** or **Queue**.
	To	Type in the message target, as expected by the server.
	Processing Mode	Select the processing mode for the messages. **Raw Message** or **Message Content**
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The **tJMSOutput** schema is read-only. It is made of one column: **Message**
Advanced settings	*Delivery Mode*	Select a delivery mode from this list to ensure the quality of data delivery: **Not Persistent**: This mode allows data loss during the data exchange. **Persistent**: This mode ensures the integrity of message delivery.
	Properties	Click the plus button underneath the table to add lines that contains username and password required for user authentication.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as an output component. It must be linked to an input component.	
Limitation	Make sure the JMS server is launched. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenarios

No scenario is available for this component yet.

tMicrosoftMQInput

tMicrosoftMQInput Properties

Component family	Internet/MOM and JMS	
Function	This component retrieves the first message in a given Microsoft message queue (only support String).	
Purpose	This component allows you to fetch messages one by one in the ID sequence of these messages from the Microsoft message queue. Each execution retrieves only one message.	
Basic settings	*PROPERTY*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually
	Host	Type in the Host name or IP address of the host server.
	Queue	Enter the queue name you want to retrieve messages from.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as a start component of a Job or Subjob. It must be linked to an output component.	
Connections		Outgoing links (from one component to another): **Row**: Main, Iterate **Trigger**: Run if; On Subjob Ok, On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if, On Subjob Ok, On Component Ok, On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	This component supports only String type. Also, it only works with the Windows systems. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Writing and fetching queuing messages from Microsoft message queue

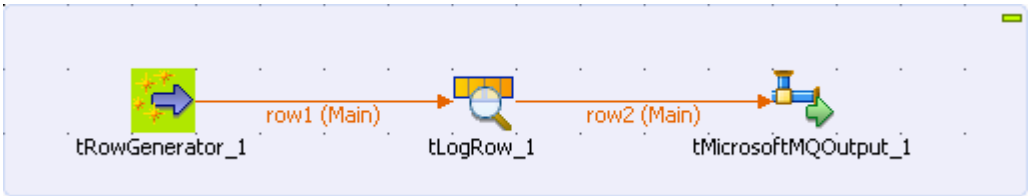
This scenario is made of two Jobs. The first Job posts messages on a Microsoft message queue and the second Job fetches the message from the server.

Posting messages on a Microsoft message queue

In the first Job, a string message is created using a **tRowGenerator** and put on a Microsoft message queue using a **tMicrosoftMQOutput**. An intermediary **tLogRow** component displays the flow being passed.

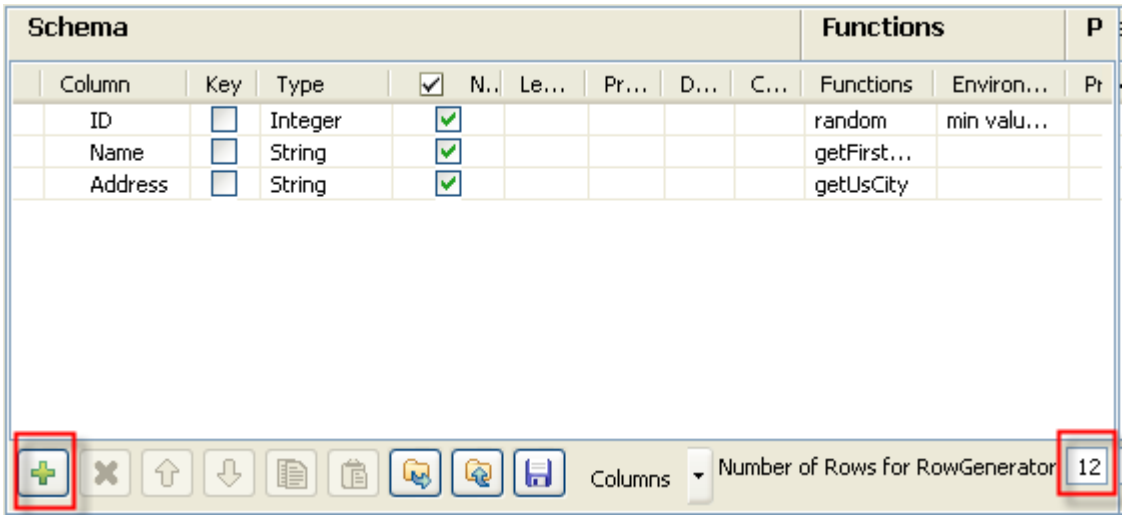
Dropping and linking components

1. Drop the three components required for the first Job from the **Palette** onto the design workspace.
2. Connect the components using a **Row** > **Main** link.



Configuring the components

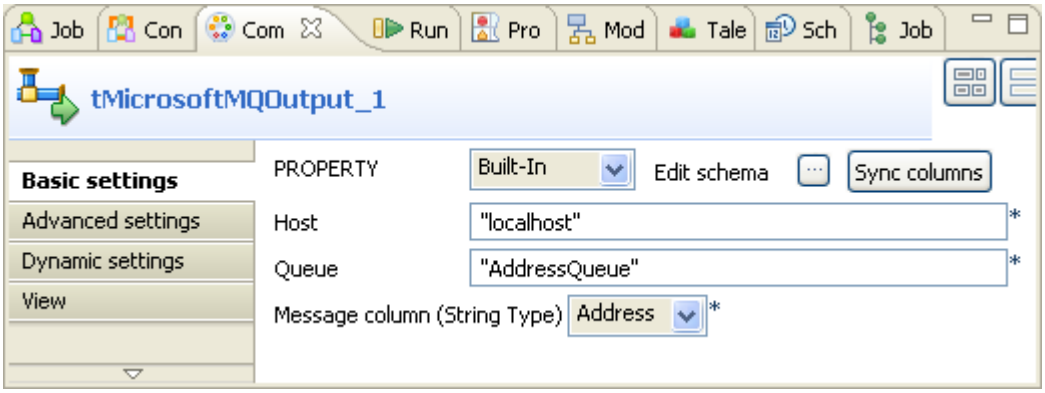
1. Double-click **tRowGenerator** to open its editor.



2. Click the plus button to add three rows into the schema table.
3. In the **Column** column, type in a new name for each row to rename it. Here, we type in *ID*, *Name* and *Address*.
4. In the **Type** column, select **Integer** for the *ID* row from the drop-down list and leave the other rows as **String**.
5. In the **Functions** column, select `random` for the *ID* row, `getFirstName` for the *Name* row and `getUsCity` for the *Address* row.
6. In the **Number of Rows for RowGenerator** field on the right end of the toolbar, type in *12* to limit the number of rows to be generated. Then, Click **Ok** to validate this editing.

In real case, you may use an input component to load the data of your interest, instead of the **tRowGenerator** component.

7. Double click the **tMicrosoftMQOutput** component to open its **Component** view.



8. In the **Host** field, type in the host address. In this example, it is *localhost*.
9. In the **Queue** field, type in the queue name you want to write message in. In this example, name it *AddressQueue*.
10. In **Message column (String Type)** field, select *Address* from the drop-down list to determine the message body to be written.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

```
Starting job tMicrosoftMQ at 14:03 18/11/2010.

[statistics] connecting to socket on port 3852
[statistics] connected
Queue open failure: Cannot open queue. (hr=MQ_ERROR_QUEUE_NOT_FOUND)
OpenQueueWithAccess (DIRECT=OS:localhost\private$\AddressQueue)
open: fmtname(DIRECT=OS:localhost\private$\AddressQueue) accessmode(1)
sharemode(0)
.....
.....attempting to create queue with
name= '.\private$\AddressQueue', label='Created by
org.talend.msmq.MsmqUtil.java'
OpenQueueWithAccess (DIRECT=OS:.\private$\AddressQueue)
open: fmtname(DIRECT=OS:.\private$\AddressQueue) accessmode(1) sharemode(0)
open: fmtname(DIRECT=OS:.\private$\AddressQueue) accessmode(2) sharemode(0)
7|Theodore|Atlanta
8|Andrew|Saint Paul
7|Benjamin|Oklahoma City
3|Abraham|Providence
4|Rutherford|Lincoln
6|Chester|Little Rock
11|Richard|Lincoln
8|Bill|Boston
9|Herbert|Jackson
10|James|Honolulu
5|Jimmy|Montpelier
9|Woodrow|Nashville
[statistics] disconnected
Job tMicrosoftMQ ended at 14:03 18/11/2010. [exit code=0]
```

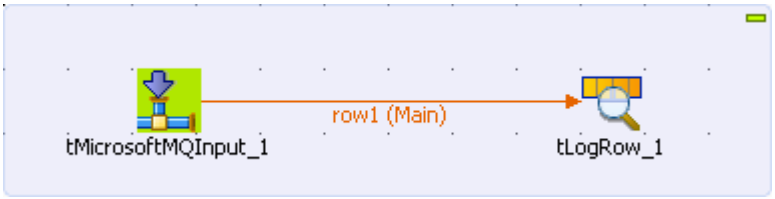
You can see that this queue has been created automatically and that the messages have been written.

Fetching the first queuing message from the message queue

Now set the second Job in order to fetch the first queuing message from the message queue.

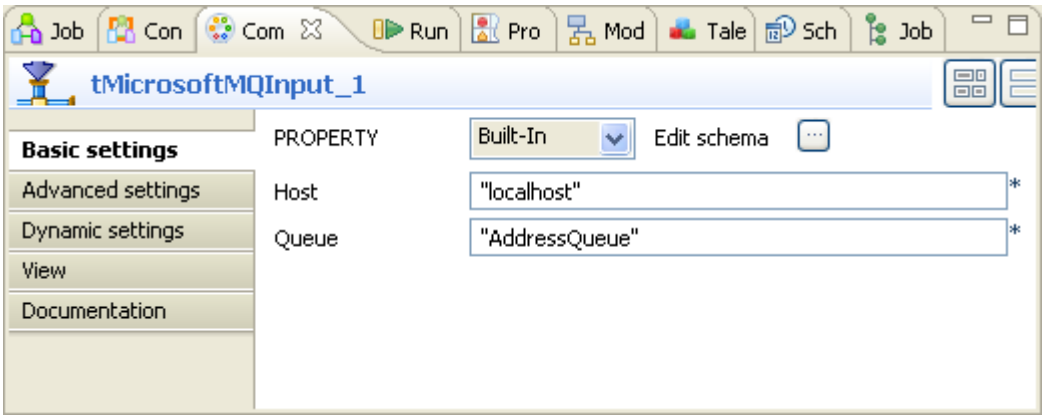
Dropping and linking components

1. Drop **tMicrosoftMQInput** and **tLogRow** from the **Palette** to the design workspace.
2. Connect these two components using a **Row** > **Main** link.



Configuring the components

1. Double-click the **tMicrosoftMQInput** to open its **Component** view.



2. In the **Host** field, type in the host name or address. Here, we type in *localhost*.
3. In the **Queue** field, type in the queue name from which you want to fetch the message. In this example, it is *AddressQueue*.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** on the **Run** tab to execute the Job.

```
Starting job MQInput at 16:57 18/11/2010.

[statistics] connecting to socket on port 3719
[statistics] connected
OpenQueueWithAccess
(DIRECT=OS:localhost\private$\AddressQueue)
open:
fmtname(DIRECT=OS:localhost\private$\AddressQueue)
accessmode(1) sharemode(0)
open:
fmtname(DIRECT=OS:localhost\private$\AddressQueue)
accessmode(2) sharemode(0)
Atlanta
[statistics] disconnected
Job MQInput ended at 16:57 18/11/2010. [exit code=0]
```

The message body *Atlanta* fetched from the queue is displayed on the console.

tMicrosoftMQOutput

tMicrosoftMQOutput Properties

Component family	Internet/MOM and JMS	
Function	This component writes a defined column of given inflow data to Microsoft message queue (only support String type).	
Purpose	This component makes it possible to write messages to Microsoft message queue.	
Basic settings	*PROPERTY*	In *Talend Open Studio for Big Data*, the Property type is always **Built-in**.
		Built-in: No property data stored centrally. Enter properties manually
	Host	Type in the Host name or the IP address of the host server.
	Queue	Type in the name of the queue which you want write a given message in. This queue can be created automatically on the fly if it does not exist then.
	Message column	Select the column as message to be written to Microsoft message queue. The selected column must be of String type.
Usage	This component must be linked to an input or intermediary component.	
Connections		Outgoing links (from one component to another): **Row**: Main, Iterate **Trigger**: Run if, On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; Iterate; **Trigger**: Run if, On Subjob Ok, On Subjob Error; On Component Ok, On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	The message to be output cannot be null. Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related scenario

For a related scenario, see section *Scenario: Writing and fetching queuing messages from Microsoft message queue*

tPOP

tPOP properties

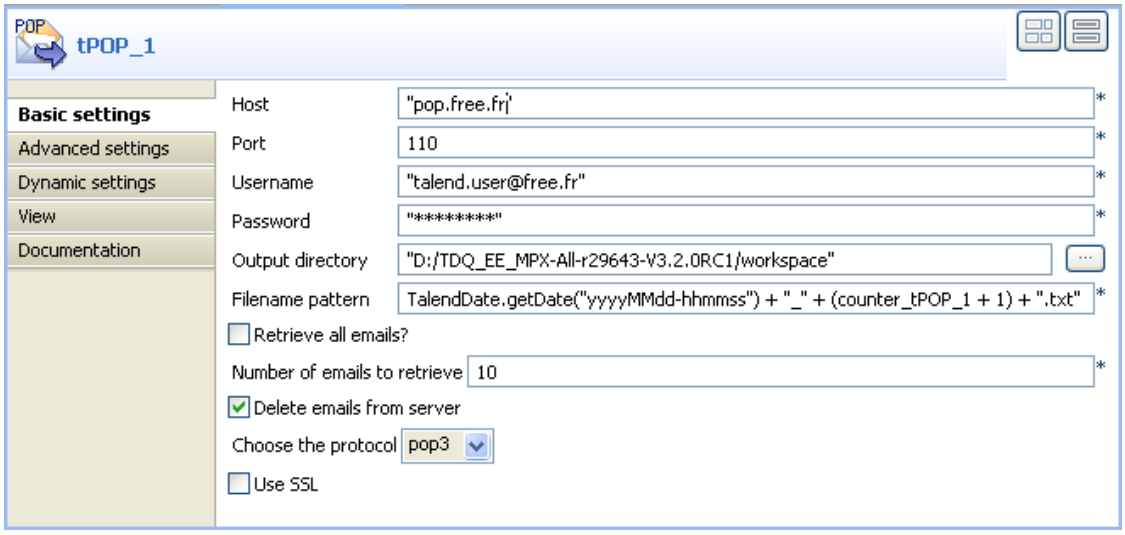
Component family	Internet	
Function	The **tPOP** component fetches one or more email messages from a server using the POP3 or IMAP protocol.	
Purpose	The **tPOP** component uses the POP or IMAP protocol to connect to a specific email server. Then it fetches one or more email messages and writes the recovered information in specified files. Parameters in the **Advanced settings** view allows you to use filters on your selection.	
Basic settings	*Host*	IP address of the email server you want to connect to.
	Port	Port number of the email server.
	Username and *Password*	User authentication data for the email server. **Username**: enter the username you use to access your email box. **Password**: enter the password you use to access your email box.
	Output directory	Enter the path to the file in which you want to store the email messages you retrieve from the email server, or click the three-dot button next to the field to browse to the file.
	Filename pattern	Define the syntax of the names of the files that will hold each of the email messages retrieved from the email server, or press **Ctrl +Space** to display the list of predefined patterns.
	Retrieve all emails?	By default, all email messages present on the specified server are retrieved. To retrieve only a limited number of these email messages, clear this check box and in the **Number of emails to retrieve**.field, enter the number of messages you want to retrieve. email messages are retrieved starting from the most recent.
	Delete emails from server	Select this check box if you do not want to keep the retrieved email messages on the server. For Gmail servers, this option does not work for the pop3 protocol. Select the imap protocol and ensure that the Gmail account is configured to use imap.
	Choose the protocol	From the list, select the protocol to be used to retrieve the email messages from the server. This protocol is the one used by the email server. If you choose the imap protocol, you will be able to select the folder from which you want to retrieve your emails.
	Use SSL	Select this check box if your email server uses this protocol for authentication and communication confidentiality. This option is obligatory for users of Gmail.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
	Filter	Click the plus button to add as many lines as needed to filter email messages and retrieve only a specific selection:
		Filter item: select one of the following filter types from the list: **From**: email messages are filtered according to the sender email address.

		To: email messages are filtered according to the recipient email address. **Subject**: email messages are filtered according to the message subject matter. **Before date**: email messages are filtered by the sending or receiving date. All messages before the set date are retrieved. **After date**: email messages are filtered by the sending or receiving date. All messages after the set date are retrieved.
		Pattern: press **Ctrl+Space** to display the list of available values. Select the value to use for each filter.
	Filter condition relation	Select the type of logical relation you want to use to combine the specified filters: **and**: the conditions set by the filters are combined together, the research is more restrictive. **or**: the conditions set by the filters are independent, the research is large.
Usage	This component does not handle data flow, it can be used alone.	
Limitation	n/a	

Scenario: Retrieving a selection of email messages from an email server

This Java scenario is a one-component Job that retrieves a predefined number of email messages from an email server.

- Drop the **tPOP** component from the **Palette** to the design workspace.
- Double click **tPOP** to display the **Basic settings** view and define the component properties.
- Enter the email server IP address and port number in the corresponding fields.
- Enter the username and password for your email account in the corresponding fields. In this example, the email server is called *Free*.



- In the **Output directory** field, enter the path to the output directory manually, or click the three-dot button next to the field and browse to the output directory where the email messages retrieved from the email server are to be stored.

- In the **Filename pattern** field, define the syntax you want to use to name the output files that will hold the messages retrieved from the email server, or press **Ctrl+Space** to display a list of predefined patterns. The syntax used in this example is the following: `TalendDate.getDate("yyyyMMdd-hhmmss") + "_" + (counter_tPOP_1 + 1) + ".txt"`.

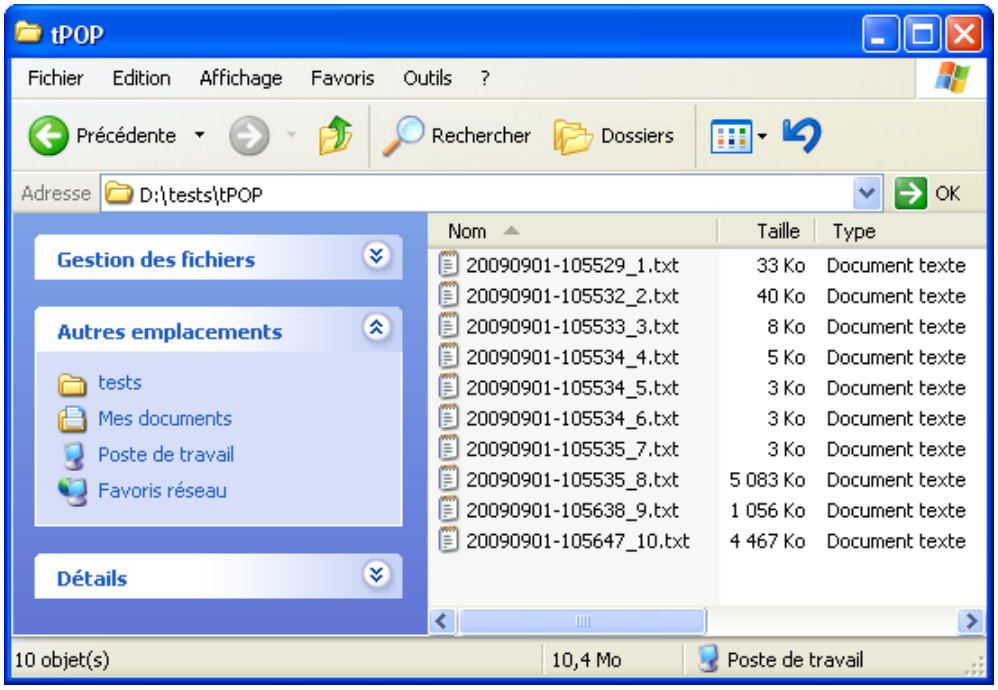
  The output files will be stored as .txt files and are defined by date, time and arrival chronological order.

- Clear the **Retrieve all emails?** field and in the **Number of emails to retrieve** field, enter the number of email messages you want to retrieve, 10 in this example.

- Select the **Delete emails from server** check box to delete the email messages from the email server once they are retrieved and stored locally.

- In the **Choose the protocol** field, select the protocol type you want to use. This depends on the protocol used by the email server. Certain email suppliers, like *Gmail*, use both protocols. In this example, the protocol used is *pop3*.

- Save your Job and press **F6** to execute it.



The **tPOP** component retrieves the 10 recent messages from the specified email server.

In the *tPOP* directory stored locally, a .txt file is created for each retrieved message. Each file holds the metadata of the email message headings (sender's address, recipient's address, subject matter) in addition to the message content.

tREST

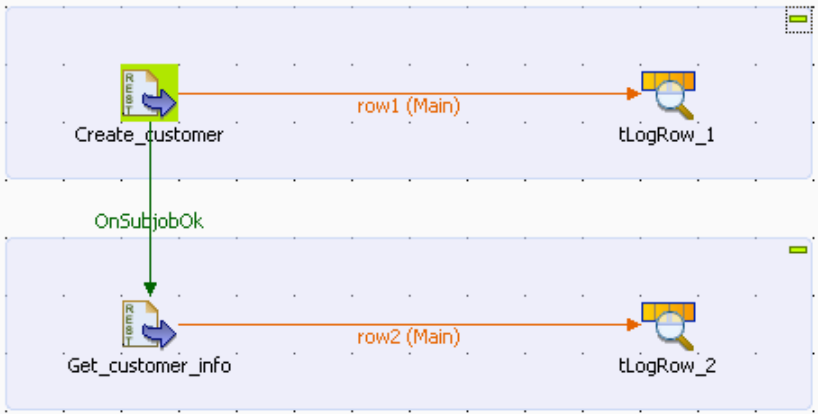
tREST properties

Component family	Internet	
Function	The **tREST** component sends HTTP requests to a REpresentational State Transfer (REST) Web service provider and gets responses correspondingly.	
Purpose	The **tREST** component serves as a REST Web service client that sends HTTP requests to a REST Web service provider and gets the responses.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. This component always uses a built-in, read-only schema that contains two columns: - **Body**: stores the result from the server end. - **ERROR_CODE**: stores the HTTP status code from the server end when an error occurs during the invocation process. The specific meanings of the errors codes are subject to definitions of your Web service provider. For reference information, visit en.wikipedia.org/wiki/List_of_HTTP_status_codes. Click **Edit Schema** to view the schema structure. *Changing the schema type may result in loss of the schema structure and therefore failure of the component.*
	URL	Type in the URL address of the REST Web server to be invoked.
	HTTP Method	From this list, select an HTTP method that describes the desired action. The specific meanings of the HTTP methods are subject to definitions of your Web service provider. Listed below are the generally accepted HTTP method definitions: - **GET**: retrieves data from the server end based on the given parameters. - **POST**: creates and uploads data based on the given parameters. - **PUT**: updates data based on the given parameters, or if the data does not exist, creates it. - **DELETE**: removes data based on the given parameters.
	HTTP Headers	Type in the name-value pair(s) for HTTP headers to define the parameters of the requested HTTP operation. For the specific definitions of HTTP headers, consult your REST Web service provider. For reference information, visit en.wikipedia.org/wiki/List_of_HTTP_headers.
	HTTP Body	Type in the payload to be uploaded to the server end when the **POST** or **PUT** action is selected.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	Use this component as a REST Web service client to communicate with a REST Web service provider. It must be linked to an output component.	
Limitation	JRE 1.6 must be running for this component to work properly.	

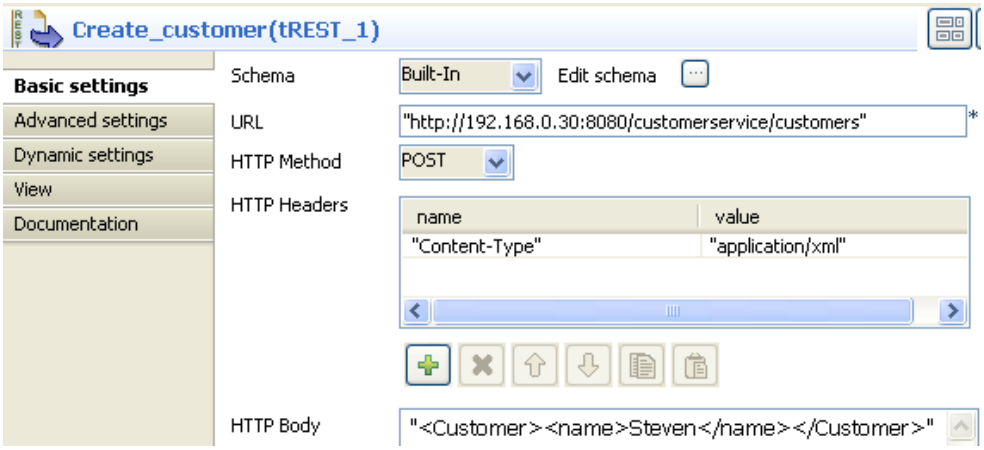
Scenario: Creating and retrieving data by invoking REST Web service

This scenario describes a simple Job that invokes a REST Web service to create a new customer record on the server end and then retrieve the customer information. When executed, the Job displays relevant information on the **Run** console.

- Drop the following components from the **Palette** onto the design workspace: two **tREST** components and two **tLogRow** components, and label the two **tREST** components to best describe the actions to perform.
- Connect each **tREST** to one **tLogRow** using a **Row** > **Main** connection.
- Connect the first **tREST** to the second **tREST** using a **Trigger** > **OnSubjobOK** connection.



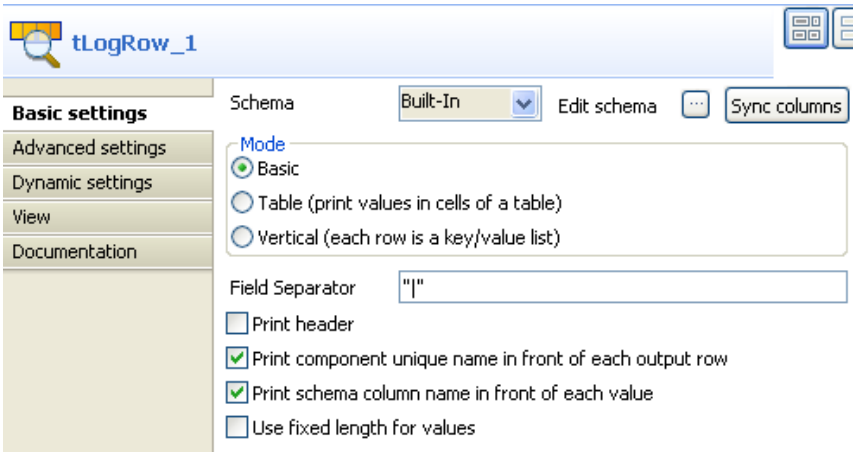
- Double click the first **tREST** component to open its **Basic settings** view.



- Fill the **URL** field with the URL of the Web service you are going to invoke. Note that the URL provided in this use case is for demonstration purpose only and is not a live address.
- From the **HTTP Method** list, select **POST** to send an HTTP request for creating a new record.
- Click the plus button to add a line in the **HTTP Headers** table, and type in the appropriate name-value key pair, which is subject to definition of your service provider, to indicate the media type of the payload to send to the server end. In this use case, type in *Content-Type* and *application/xml*. For reference information about Internet media types, visit www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.7.
- Fill the **HTTP Body** field with the payload to be uploaded to the server end. In this use case, type in *<Customer><name>Steven</name></Customer>* to create a record for a new customer named *Steven*.

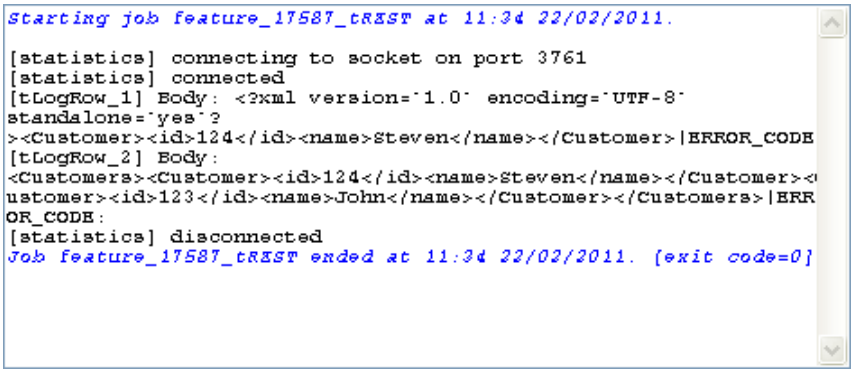
If you want to include double quotation marks in your payload, be sure to use a backslash escape character before each of the quotation marks. In this use case, for example, type in *<Customer><name>\"Steven\"</name></Customer>* if you want to enclose the name *Steven* in a pair of double quotation marks.

- Double click the second **tREST** component to open its **Basic settings** view.
- Fill the **URL** field with the same URL.
- From the **HTTP Method** list, select **GET** to send an HTTP request for retrieving the existing records.
- In the **Basic settings** view of each **tLogRow**, select the **Print component unique name in front of each output row** and **Print schema column name in front of each value** check boxes for better identification of the output flows.



- Save your Job and press **F6** to launch it.

  The console shows that the first **tREST** component sends an HTTP request to the server end to create a new customer named *Steven*, and the second **tREST** component successfully reads data from the server end, which includes the information of the new customer you just created.



tRSSInput

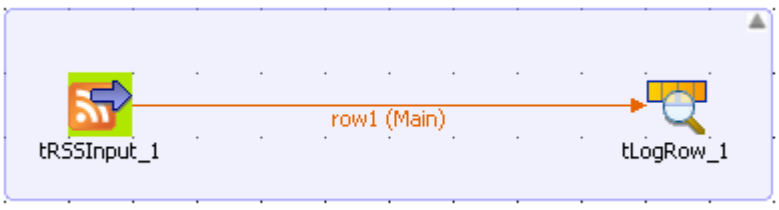
tRSSInput Properties

Component family	Internet	
Function	**tRSSInput** reads RSS-Feeds using URLs.	
Purpose	**tRSSInput** makes it possible to keep track of blog entries on websites to gather and organize information for quick and easy access.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. The **tRSSInput** component has a read-only schema that is made of four columns: **TITLE**, **DESCRIPTION**, **PUBDATE**, and **Link**.
	RSS URL	Enter the URL for the RSS_Feed to read.
	Read articles from	If selected, **tRSSInput** reads articles on the RSS_Feed from the date set through the three-dot **[...]** button next to the **date time** field.
	Max number of articles	If selected, **tRSSInput** reads as many articles as the number entered in the **max amount** field.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
Usage	This component is generally used as an input component. It requires an output component.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

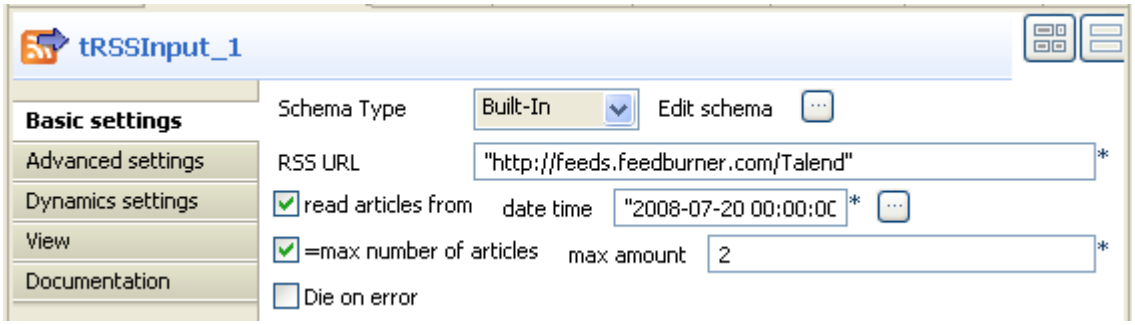
Scenario: Fetching frequently updated blog entries.

This two-component scenario aims at retrieving frequently updated blog entries from a **Talend** local news RSS feed using the **tRSSInput** component.

1. Drop the following components from the **Palette** onto the design workspace: **tRSSInput** and **tLogRow**.
2. Right-click to connect them using a **Row** > **Main** link.



3. In the design workspace, select **tRSSInput**, and click the **Component** tab to define the basic settings for **tRSSInput**.



4. Enter the URL for the RSS_Feed to access. In this scenario, **tRSSInput** links to the **Talend** RSS_Feed: *http://feeds.feedburner.com/Talend*.

5. Select/clear the other check boxes as required. In this scenario, we want to display the information about two articles dated from July 20, 2008.

6. In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow** properties, see section *tLogRow properties*.

7. Save the Job and press **F6** to execute it.

```
Starting job RSSInput at 15:17 07/08/2008.
.-------------------------------------------------------------------------------
|                                    #1. tLogRow_1
+-------------+-----------------------------------------------------------------
| key         | value
+-------------+-----------------------------------------------------------------
| TITLE       | Welcoming Jean-Luc Solans
| DESCRIPTION | Jean-Luc Solans joins Talend as VP of Strategy and Busines
| PUBDATE     | 24 Jul 2008 19:40:11 GMT
| LINK        | http://feeds.feedburner.com/~r/Talend/~3/344920575/
+-------------+-----------------------------------------------------------------

.-------------------------------------------------------------------------------
|                                  #2. tLogRow_1
+-------------+-----------------------------------------------------------------
| key         | value
+-------------+-----------------------------------------------------------------
| TITLE       | Talend Open Profiler gets rave reviews
| DESCRIPTION | An interview and a product review of Talend Open Profiler.
| PUBDATE     | 23 Jul 2008 21:12:53 GMT
| LINK        | http://feeds.feedburner.com/~r/Talend/~3/343928056/
+-------------+-----------------------------------------------------------------

Job RSSInput ended at 15:17 07/08/2008. [exit code=0]
```

The **tRSSInput** component accessed the RSS feed of **Talend** website on your behalf and organized the information for you.

Two blog entries are displayed on the console. Each entry has its own title, description, publication date, and the corresponding RSS feed URL address. Blogs show the last entry first, and you can scroll down to read earlier entries.

tRSSOutput

tRSSOutput Properties

Component family	Internet	
Function	**tRSSOutput** writes RSS_Feed or Atom_Feed XML files.	
Purpose	**tRSSOutput** makes it possible to create XML files that hold RSS or Atom feeds.	
Basic settings	*File name*	Name or path to the output XML file. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Encoding	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Append	Select this check box to add the new rows to the end of the file.
	Mode	Select between **RSS** or **ATOM** according to the feed you want to generate.
	Channel (in RSS mode)	The information to be typed in here concerns your entire input data, site etc, rather than a particular item. **Title**: Enter a meaningful title. **Description**: Enter a description that you think will describe your content. **Publication date**: Enter the relevant date. **Link**: Enter the relevant URL.
	Feed (in ATOM mode)	**Title**: Enter a meaningful title. **Link**: Enter the relevant URL. **Id**: Enter the valid URL corresponding to the **Link**. **Update date**: Enter the relevant date . **Author name**: Enter the relevant name.
	Optionnal Channel Elements	Click the [+] button below the table to add new lines and enter the information relative to the RSS flow metadata: **Element Name**: name of the metadata. **Element Value**: content of the metadata.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. By default, the schema of **tRSSOutput** is made of five read-only columns: **id**, **title**, **link**, **updated**, and **summary**. You can add new columns or click **Syn columns** to retrieve the schema structure from the preceding component.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component must be linked to an input or intermediary component.	
Limitation	n/a	

Scenario 1: Creating an RSS flow and storing files on an FTP server

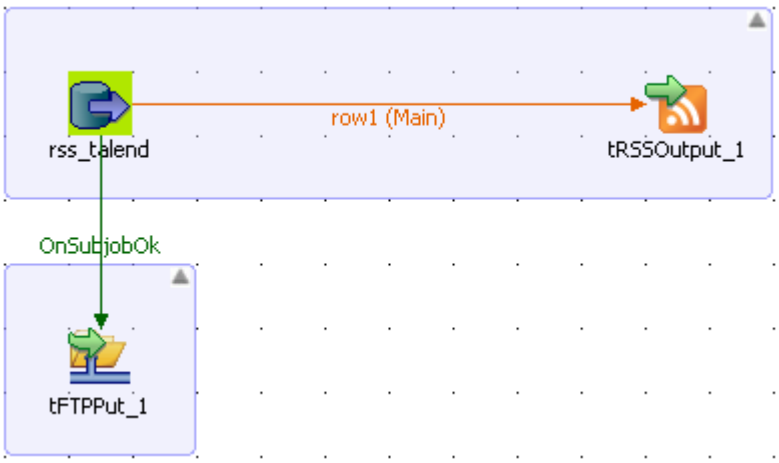
In this scenario we:

- create an RSS flow for files that you would like to share with other people, and
- store the complete files on an FTP server.

This scenario writes an RSS feed XML file about a Mysql table holding information about books. It adds links to the files stored on an FTP server in case users want to have access to the complete files.

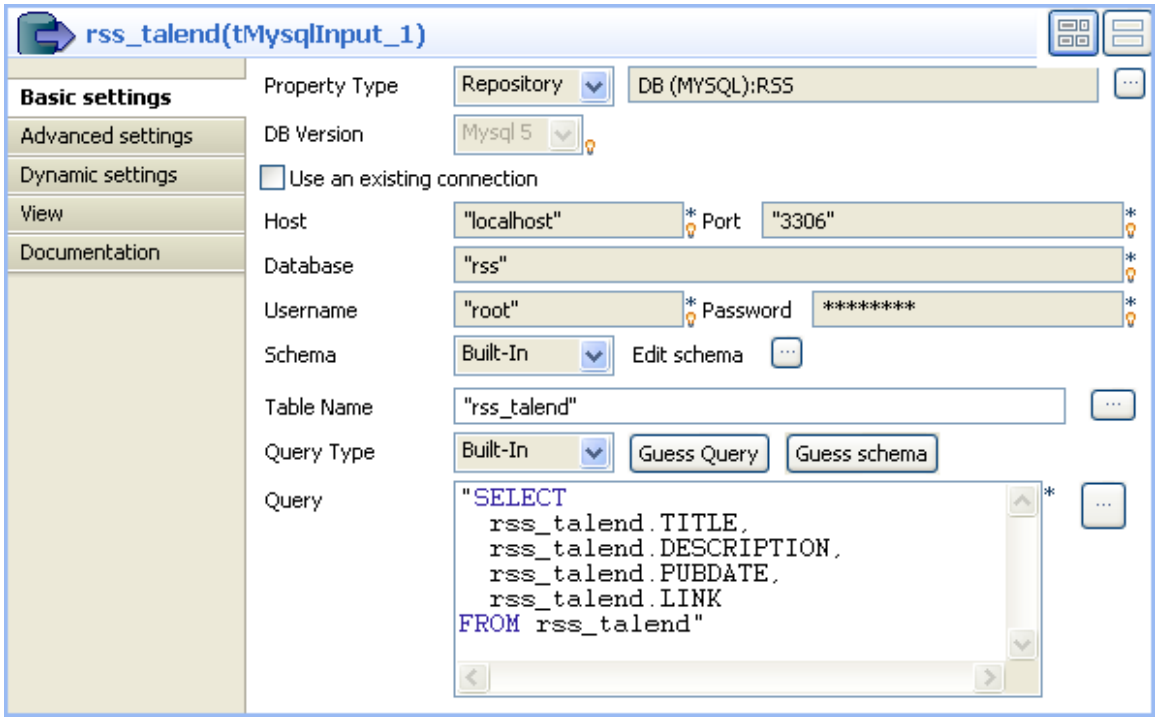
Dropping and linking components

1. Drop the following components from the **Palette** onto the design workspace: **tMysqlInput**, **tRSSOutput**, and **tFTPPut**.
2. Right-click **tMysqlInput** and connect it to **tRSSOutput** using a **Row** > **Main** link.
3. Right-click **tMysqlInput** and connect it to **tFTPPut** using a **Trigger** > **OnSubjobOk** link.



Defining the data source

1. In the design workspace, select **tMysqlInput**, and click the **Component** tab to define the basic settings for **tMysqlInput**.



*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

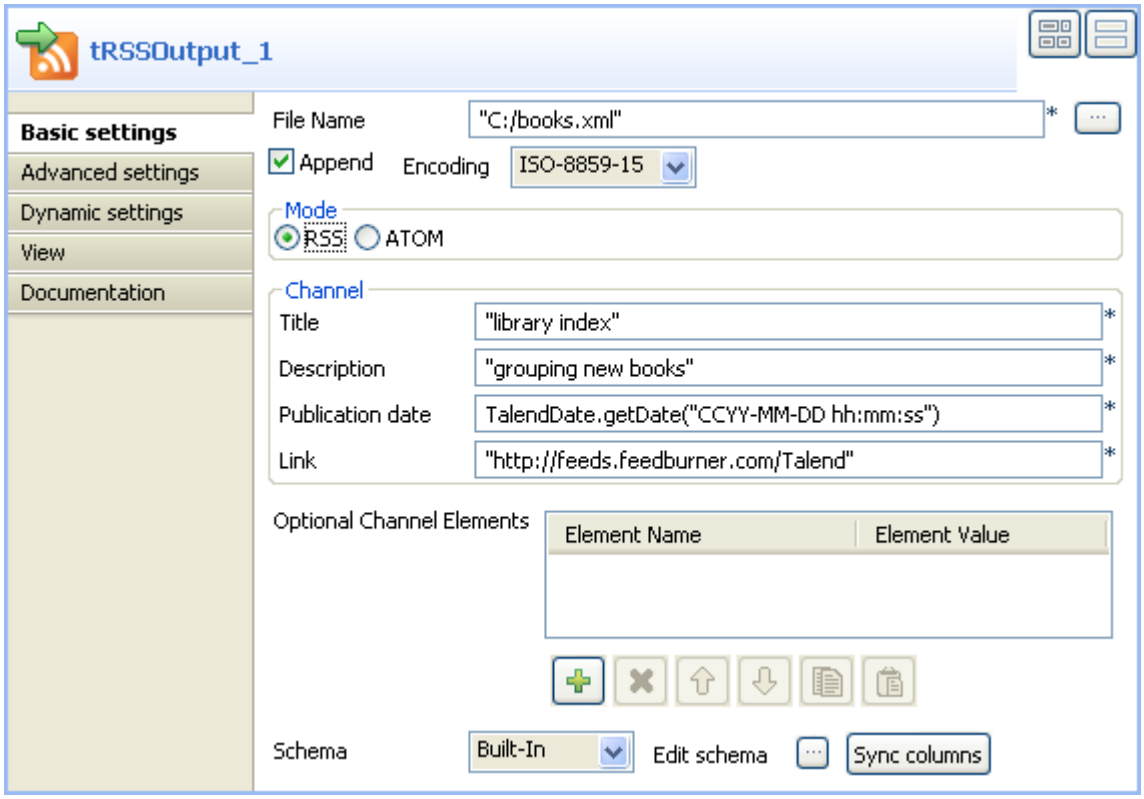
2. Define the database connection details and the input schema.

3. In the **Table Name** field, either type your table name or click the three dots button **[...]** and select your table name from the list. In this scenario, the Mysql input table is called “rss_talend” and the schema is made up of four columns, *TITLE*, *Description*, *PUBDATE*, and LINK.

4. In the **Query** field, enter your DB query paying particular attention to properly sequence the fields in order to match the schema definition, or click **Guess Query**.

Creating an RSS flow

1. In the design workspace, select **tRSSOutput**, and click the **Component** view to define the basic settings for **tRSSOutput**.



2. In the **File name** field, use the by default file name and path, or browse to set your own for the output XML file.

3. Select the encoding type on the **Encoding Type** list.

4. In the **Mode** area, select **RSS**.

5. In the **Channel** panel, enter a title, a description, a publication date, and a link to define your input data as a whole.

6. Click **Edit Schema** to modify the schema if necessary.

   You can click **Sync columns** to retrieve the generated schema from the preceding component.

7. Save your Job and press **F6** to execute this first part.

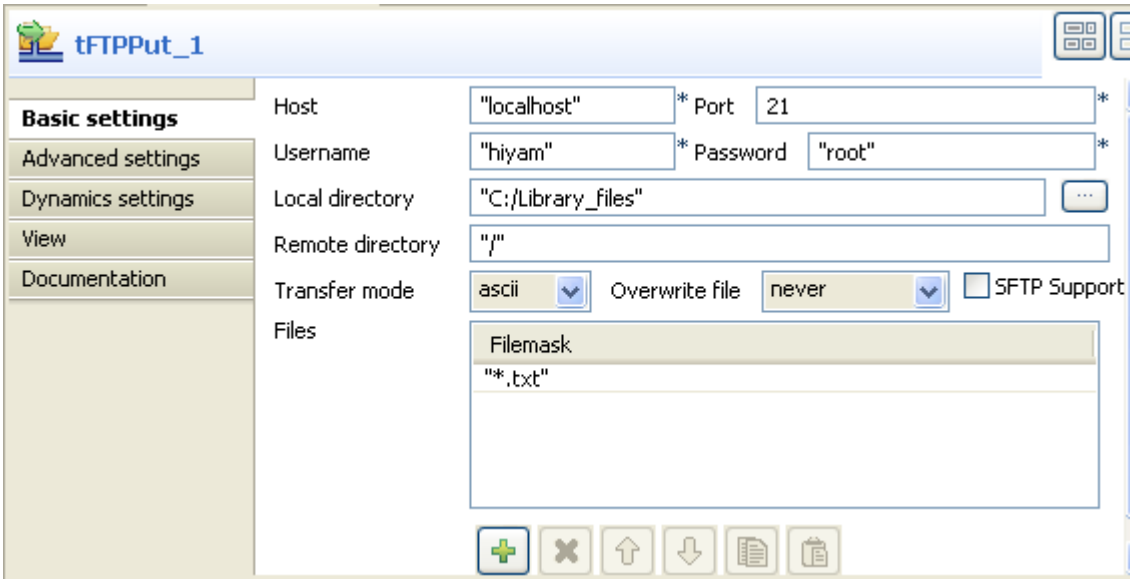
```
<?xml version="1.0" encoding="UTF-8" ?>
- <rss version="2.0">
  - <channel>
      <title>library index</title>
      <description>grouping new books</description>
      <pubdate>2008-08-16 14:55:29</pubdate>
      <link>http://feeds.feedburner.com/Talend</link>
    - <item>
        <title>Book 1</title>
        <description>Summary of book1</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_1.txt</link>
      </item>
    - <item>
        <title>Book 2</title>
        <description>Summary of book2</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_2.txt</link>
      </item>
    - <item>
        <title>Book 3</title>
        <description>Summary of book3</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_3.txt</link>
      </item>
    </channel>
</rss>
```

The **tRSSOutput** component created an output RSS flow in an XML format for the defined files.

Writing the complete files to an FTP server

To store the complete files on an FTP server:

1. In the design workspace, select **FTPPut**, and click the **Component** tab to define the basic settings for **tFTPPut**.



2. Enter the host name and the port number in their corresponding fields.
3. Enter your connection details in the corresponding **Username** and **Password** fields.
4. Browse to the local directory, or enter it manually in the **Local directory** field.
5. Enter the details of the remote server directory.
6. Select the transfer mode from the **Transfer mode** list.
7. On the **Files** panel, click on the plus button to add new lines and fill in the filemasks of all files to be copied onto the remote directory. In this scenario, the files to be saved on the FTP server are all text files.
8. Save your Job and press **F6** to execute it.

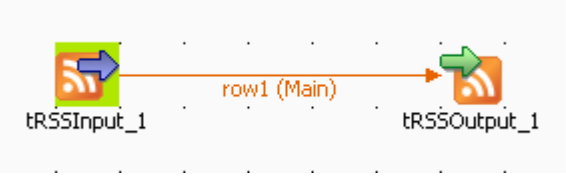
   Files defined in the Filemask are copied on the remote server.

Scenario 2: Creating an RSS flow that contains metadata

This scenario describes a two-component Job that creates an RSS flow that holds metadata and then redirects the obtained information in an XML file of the output RSS flow.

Dropping and linking components

1. Drop **tRSSInput** and **tRSSOutput** from the **Palette** to the design workspace.
2. Connect the two components together using a **Row** > **Main** link.

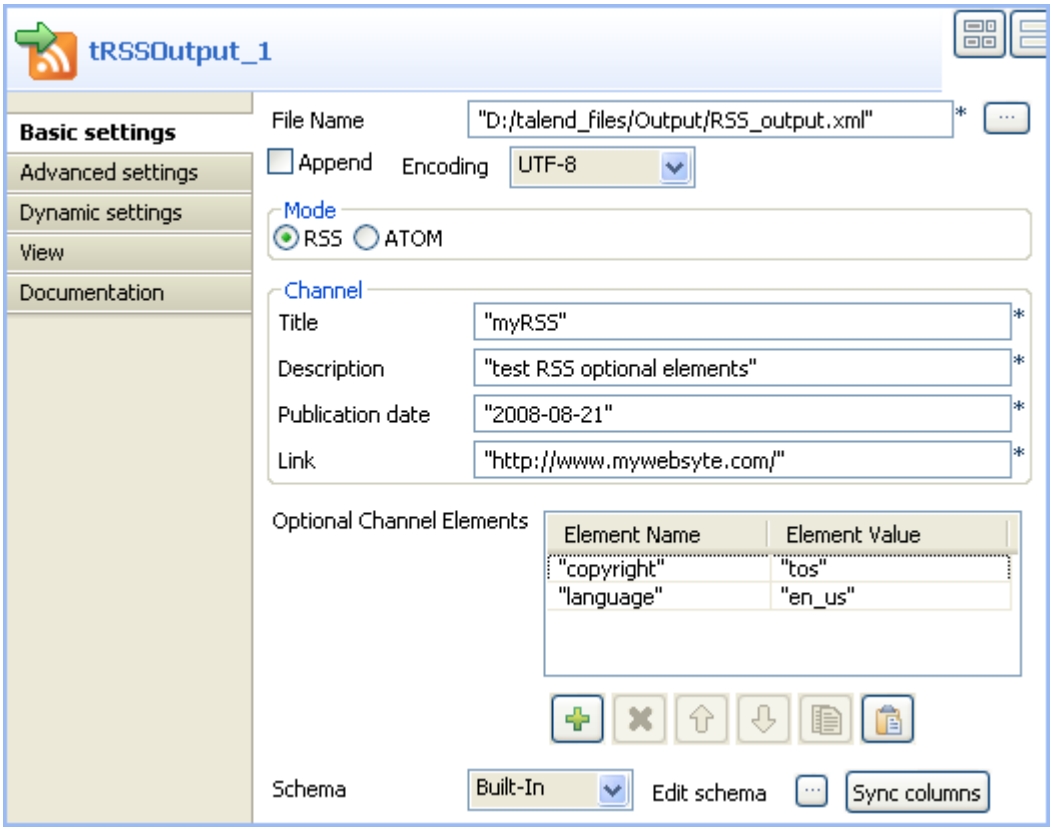


Configuring the components

1. Double-click **tRSSInput** to open its **Basic settings** view and define the component properties.

2. Enter the URL for the RSS_Feed to access. In this scenario, **tRSSInput** links to the **Talend** RSS_Feed: *http://feeds.feedburner.com/Talend*.

3. In the design workspace, double-click **tRSSOutput** to display its **Basic settings** view and define the component properties.



4. In the **File name** field, use the by default file name and path, or browse to set your own for the output XML file.

5. Select the encoding type on the **Encoding Type** list.

6. In the **Mode** area, select **RSS**.

7. In the **Channel** panel, enter a title, a description, a publication date and a link to define your input data as a whole.

8. In the **Optional Channel Element**, define the RSS flow metadata. In this example, the flow has two metadata: *copyright*, which value is *tos*, and *language* which value is *en_us*.

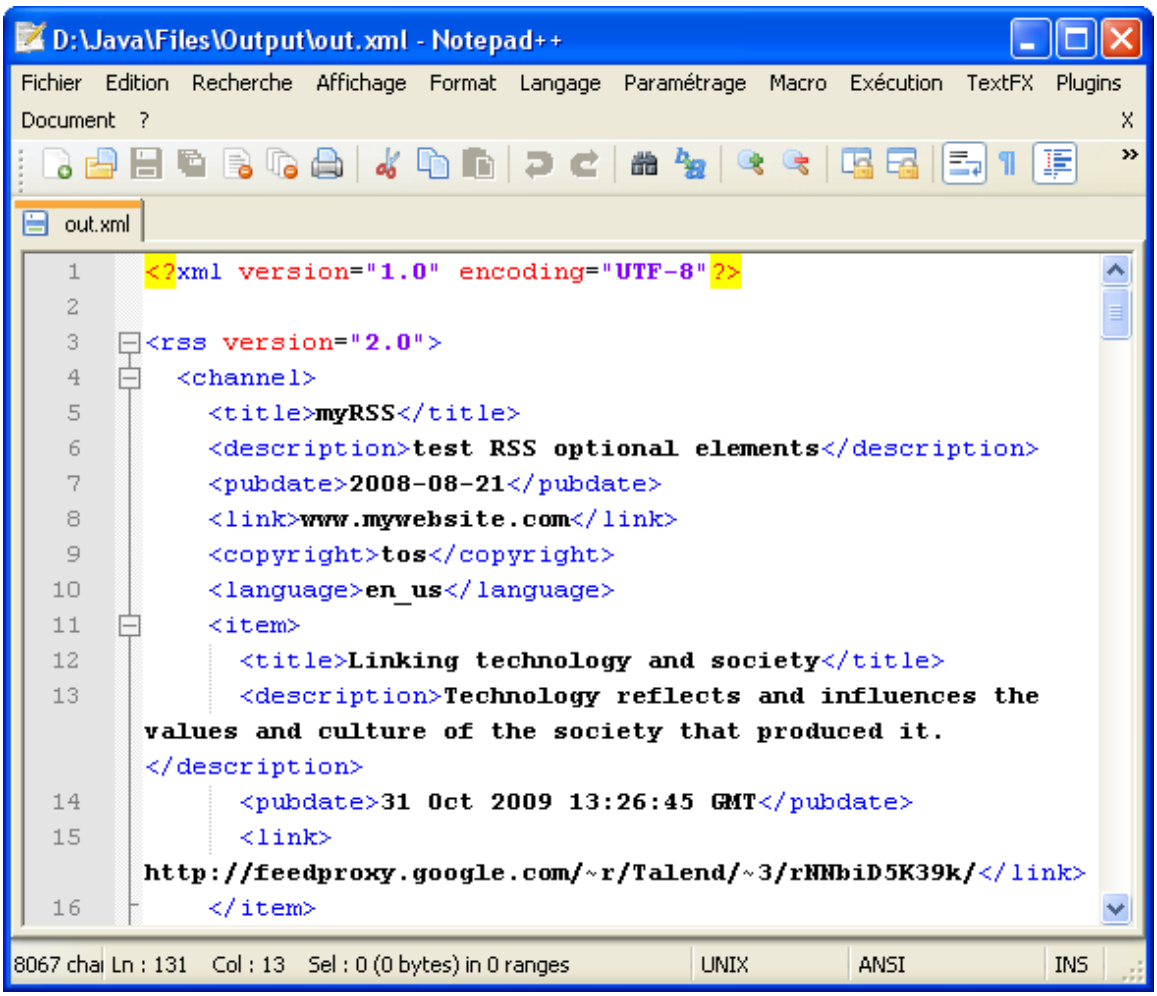
9. Click **Edit Schema** to modify the schema if necessary.

You can click **Sync Column** to retrieve the generated schema from the preceding component.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.



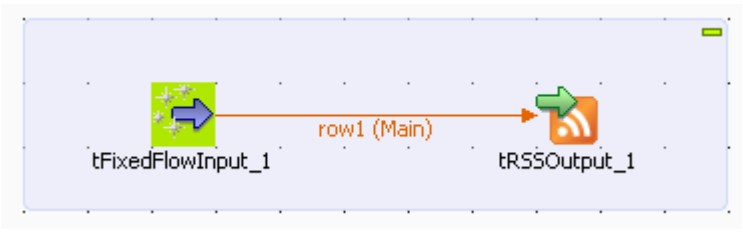
The defined files are copied in the output XML file and the metadata display under the <*channel*> node above the information about the RSS flow.

Scenario 3: Creating an ATOM feed XML file

This scenario describes a two component Job that generates data and writes them in an ATOM feed XML file.

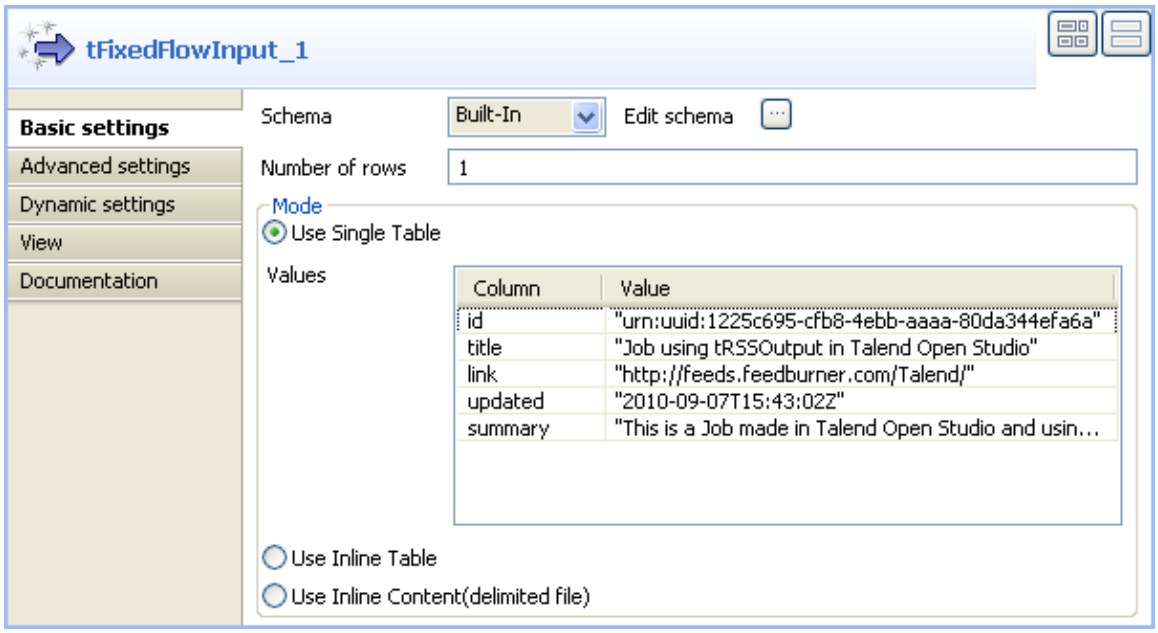
Dropping and linking components

1. Drop the following components from the **Palette** onto the deisgn workspace: **tFixedFlowInput** of the **Misc** component group and **tRSSOutput** of the **Internet** component group.
2. Right-click **tFixedFlowInput** and connect it to **tRSSOutput** using a **Row Main** link.
3. When asked whether you want to pass on the schema of **tRSSOutput** to **tFixedFlowInput**, click **Yes**.



Configuring the components

1. In the design workspace, double-click **tFixedFlowInput** to display its corresponding **Component** view and define its basic settings.

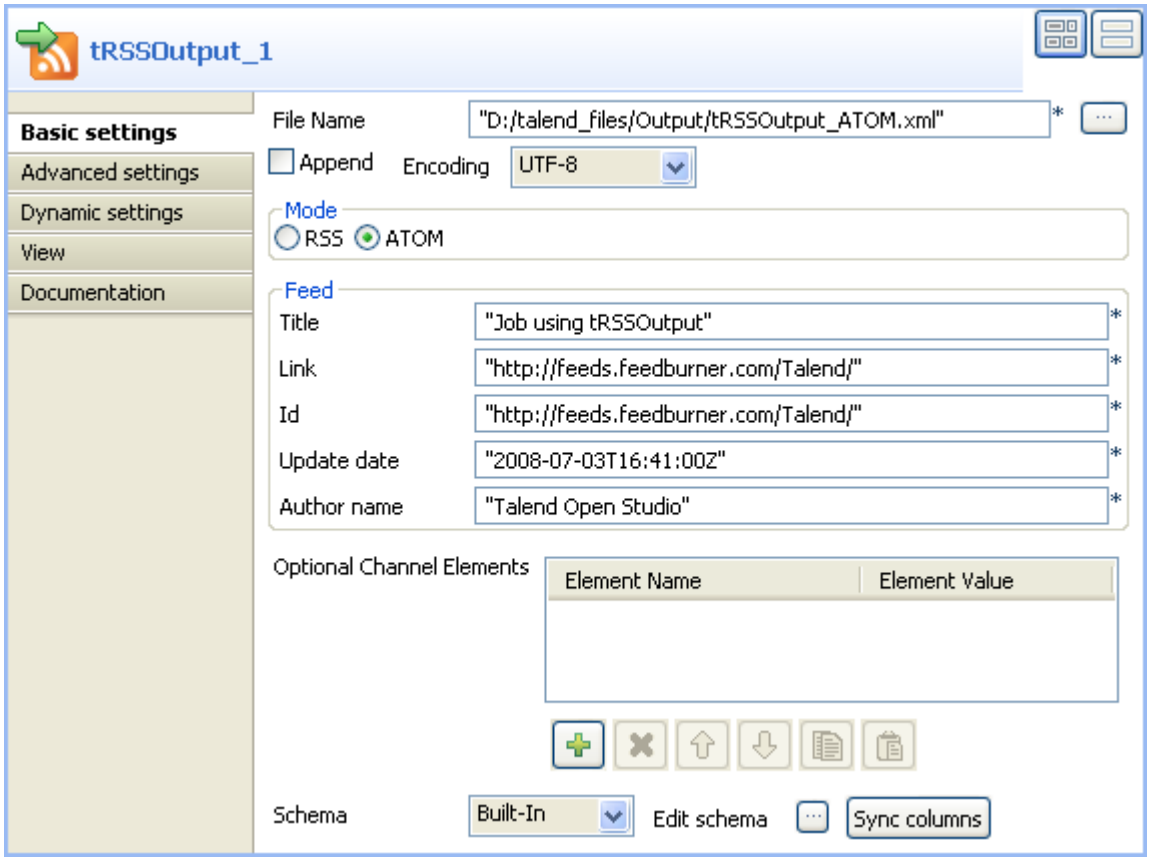


2. In the **Number of rows** field, leave the default setting to **1** to only generate one line of data.

3. In the **Mode** area, leave the **Use Single Table** option selected and fill in the **Values** table. Note that the **Column** field of the **Values** table is filled in by the columns of the schema defined in the component.

4. In the **Value** field of the **Values** table, type in the data you want to be sent to the following component.

5. In the design workspace, double-click **tRSSOutput** to display its corresponding **Component** view and define its basic settings.



6. Click the **[...]** button next to the **File Name** field to set the output XML file directory and name.

7. In the **Mode** area, select **ATOM** to generate an ATOM feed XML file.

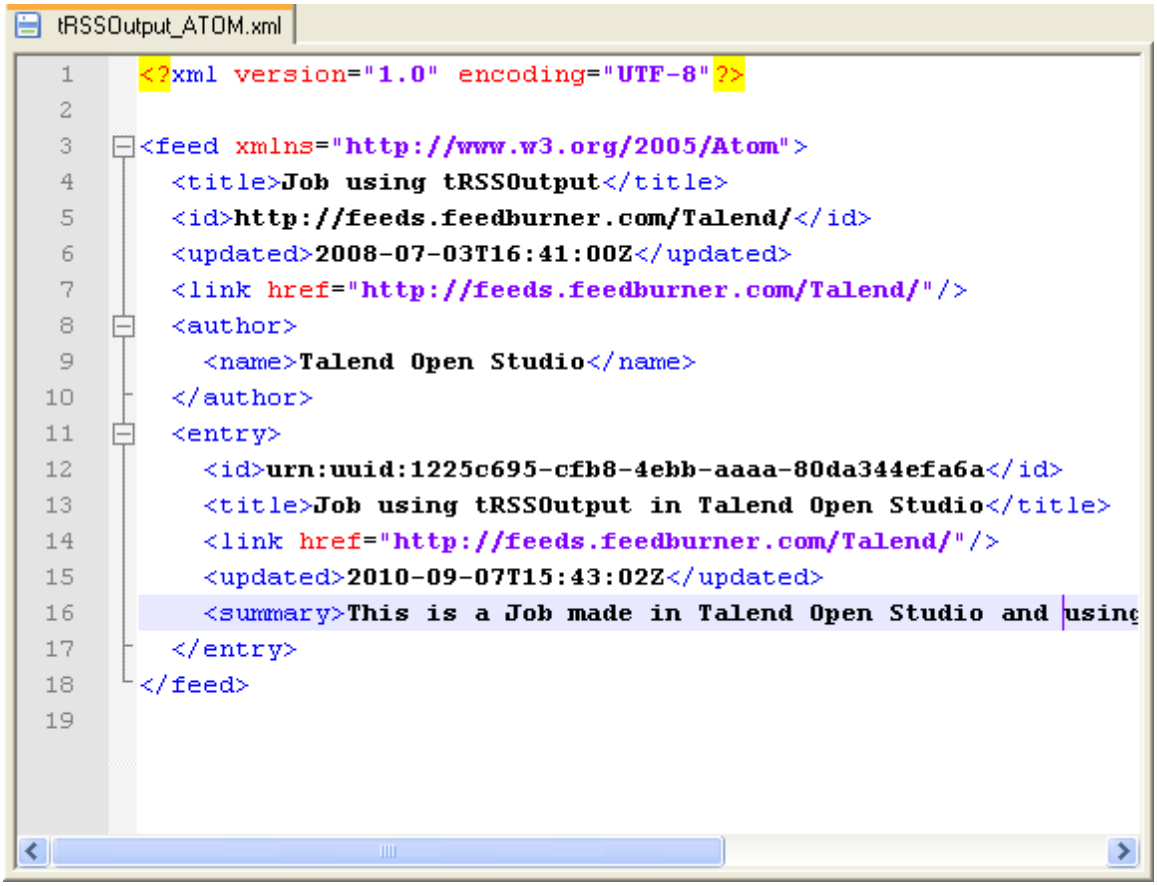
   *As the ATOM feed format is strict, some default information is required to create the XML file. So, the schema **tRSSOutput** contains default columns that will contain those information. Those default columns are greyed out to indicate that they must not be modified. If you choose to modify the schema of the component, the ATOM XML file created will not be valid.*

8. In the **Feed** area, enter a title, link, id, update date, author name to define your input data as a whole.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** or click **Run** on the **Run** tab to execute the Job.



The **tRSSOutput** component creates an output ATOM flow in an XML format.

tSCPClose

tSCPClose Properties

Componant family	Internet/SCP	
Function	**tSCPClose** closes a connection to a fully encrypted channel.	
Purpose	This component closes a connection to an SCP protocol.	
Basic settings	*Component list*	If there is more than one connection in the current Job, select **tSCPConnection** from the list.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level
Usage	**tSCPClose** is generally used as a start component. It requires an output component.	
Limitation	n/a	

Related scenario

This component is closely related to **tSCPConnection** and **tSCPRollback**. It is generally used with **SCPConnection** as it allows you to close a connection for the transaction which is underway.

For a related scenario see section *tMysqlConnection*.

tSCPConnection

tSCPConnection properties

Component family	Internet/SCP	
Function	**tSCPConnection** opens an SCP connection for the current transaction.	
Purpose	**tSCPConnection** allows you to open an SCP connection to transfer files in one transaction.	
Basic settings	*Host*	IP address of the SCP server.
	Port	Number of listening port of the SCP server.
	Username	User name for the SCP server.
	Authentication method	SCP authentication method.
	Password	User password for the SCP server.
Usage	This component is typically used as a single-component sub-job. It is used along with other SCP components.	
Limitation	n/a	

Related scenarios

For a related scenario, see section *Scenario: Putting files on a remote FTP server*.

For a related scenario using a different protocol, see section *Scenario: Getting files from a remote SCP server*.

tSCPDelete

tSCPDelete properties

Component family	Internet/SCP	
Function	This component deletes files from remote hosts over a fully encrypted channel.	
Purpose	**tSCPDelete** allows you to remove a file from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Filelist	File name or path to the files to be deleted.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPDelete** related scenario, see section *Scenario: Getting files from a remote SCP server*.

For **tSCPDelete** related scenario using a different protocol, see section *Scenario: Putting files on a remote FTP server*.

tSCPFileExists

tSCPFileExists properties

Component family	Internet/SCP	
Function	This component checks, over a fully encrypted channel, if a file exists on a remote host.	
Purpose	**tSCPFileExists** allows you to verify the existence of a file on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	File path on the remote directory.
	Filename	Name of the file to check.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPFileExists** related scenario, see section *Scenario: Getting files from a remote SCP server*.

For **tSCPFileExists** related scenario using a different protocol, see section *Scenario: Putting files on a remote FTP server*.

tSCPFileList

tSCPFileList properties

Component family	Internet/SCP	
Function	This component iterates, over a fully encrypted channel, on files of a given directory on a remote host.	
Purpose	**tSCPFileList** allows you to list files from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Command separator	The character used to separate multiple commands.
	Filelist	Directory name or path to the directory holding the files to list.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPFileList** related scenario, see section *Scenario: Getting files from a remote SCP server*.

For **tSCPFileList** related scenario using a different protocol, see section *Scenario: Putting files on a remote FTP server*.

tSCPGet

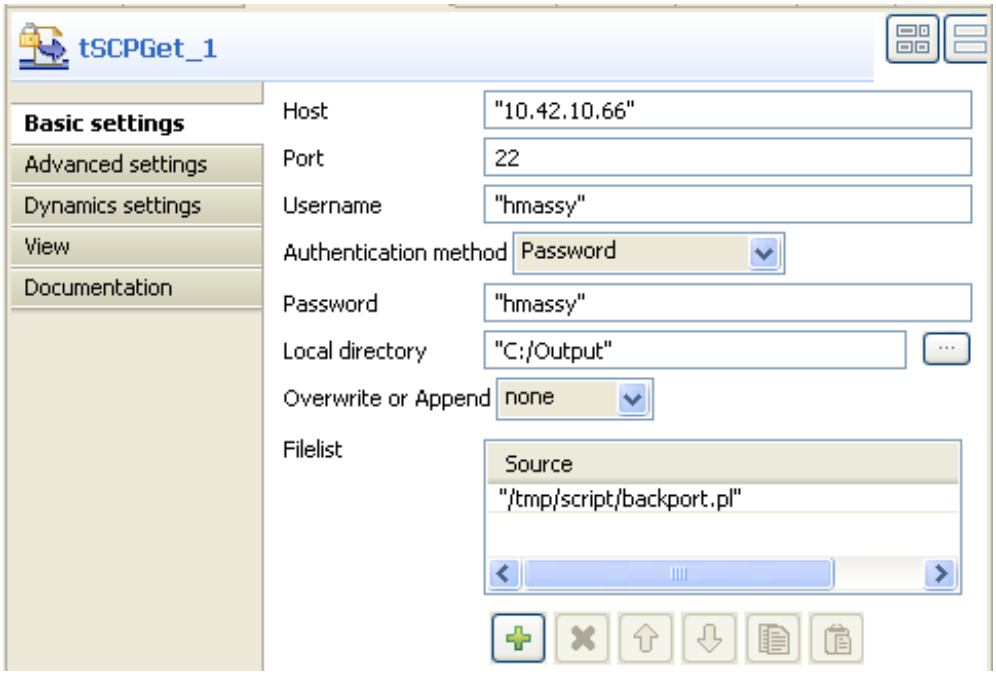
tSCPGet properties

Component family	Internet/SCP	
Function	This component transfers defined files via an SCP connection over a fully encrypted channel.	
Purpose	**tSCPGet** allows you to copy files from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Local directory	Path to the destination folder.
	Overwrite or Append	List of available options for the transferred files.
	Filelist	File name or path to the file(s) to copy.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Scenario: Getting files from a remote SCP server

This scenario creates a single-component Job which gets the defined file from a remote SCP server.

- Drop a **tSCPGet** component from the **Palette** onto the design workspace.
- In the design workspace, select **tSCPGet** and click the **Component** tab to define its basic settings.



- Fill in the **Host** IP address, the listening **Port** number, and the user name in the corresponding fields.
- On the **Authentication method** list, select the appropriate authentication method.

  Note that the field to follow changes according to the selected authentication method. The authentication form used in this scenario is password.
- Fill in the local directory details where you want to copy the fetched file.
- On the **Overwrite or Append** list, select the action to be carried out.
- In the **Filelist** area, click the plus button to add a line in the **Source** list and fill in the path to the given file on the remote SCP server.

  In this scenario, the file to copy from the remote SCP server to the local disk is *backport*.
- Save the Job and press **F6** to execute it.

The given file on the remote server is copied on the local disk.

tSCPPut

tSCPPut properties

Component family	Internet/SCP	
Function	This component copies defined files to a remote SCP server over a fully encrypted channel.	
Purpose	**tSCPPut** allows you to copy files to the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	Path. to the destination folder.
	Filelist	File name or path to the file(s) to copy.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPPut** related scenario, see section *Scenario: Getting files from a remote SCP server*.

For **tSCPut** related scenario using a different protocol, see section *Scenario: Putting files on a remote FTP server*.

tSCPRename

tSCPRename properties

Component family	Internet/SCP	
Function	This component renames files on a remote SCP server.	
Purpose	**tSCPRename** allows you to rename file(s) on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	File to rename	Enter the name or path to the file you want to rename.
	Rename to	Enter the file new name.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPRename** related scenario, see section *Scenario: Getting files from a remote SCP server*.

tSCPTruncate

tSCPRename properties

Component family	Internet/SCP	
Function	This component removes all the data from a file via an SCP connection.	
Purpose	**tSCPTruncate** allows you to remove data from file(s) on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	Path. to the destination file.
	Filelist	File name or path to the file(s) to truncate.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPTruncate** related scenario, see section *Scenario: Getting files from a remote SCP server*.

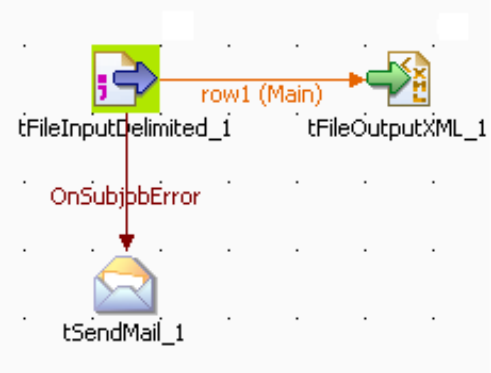
tSendMail

tSendMail Properties

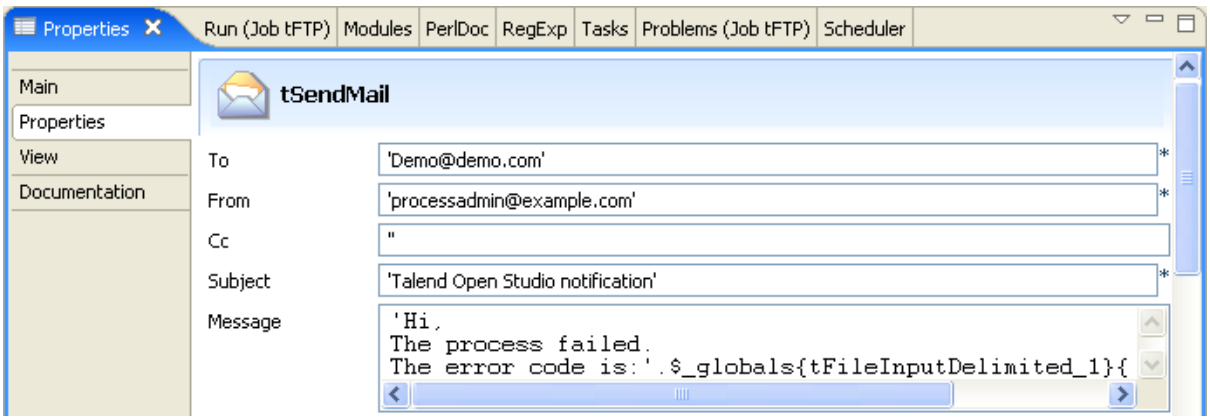
Component family	Internet	
Function	**tSendMail** sends emails and attachments to defined recipients.	
Purpose	**tSendMail** purpose is to notify recipients about a particular state of a Job or possible errors.	
Basic settings	*To*	Main recipient email address.
	From	Sending server email address.
	Show sender's name	Select this check box if you want the sender name to show in the messages.
	Cc	Email addresses of secondary recipients of the email message directed to another.
	Bcc	Email addresses of secondary recipients of the email message. Recipients listed in the **Bcc** field receive a copy of the message but are not shown on any other recipient's copy.
	Subject	Heading of the mail.
	Message	Body message of the email. Press **Ctrl+Space** to display the list of available variables.
	Die if the attachment file doesn't exist	This check box is selected by default. Clear this check box if you want the message to be sent even if there are no attachments.
	Attachments / File and Content Transfer Encoding	Click the plus button to add as many lines as needed where you can put filemask or path to the file to be sent along with the mail, if any. Two options are available for content transfer encoding, i.e. Default and Base64.
	Other Headers	Click the plus button to add as many lines as needed where you can type the **Key** and the corresponding **Value** of any header information that does not belong to the standard header.
	SMTP Host and Port	IP address of SMTP server used to send emails.
	SSL Support	Select this check box to authenticate the server at the client side via an SSL protocol.
	STARTTLS Support	Select this check box to authenticate the server at the client side via a STARTTLS protocol.
	Importance	Select in the list the priority level of your messages.
	Need authentication / Username and Password	Select this check box and enter a username and a password in the corresponding fields if this is necessary to access the service.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*MIME subtype from the 'text' MIME type*	Select in the list the structural form for the text of the message.
	Encoding type	Select the encoding from the list or select **Custom** and define it manually.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is typically used as one sub-job but can also be used as output or end object. It can be connected to other components with either **Row** or **Iterate** links.	
Limitation	n/a	

Scenario: Email on error

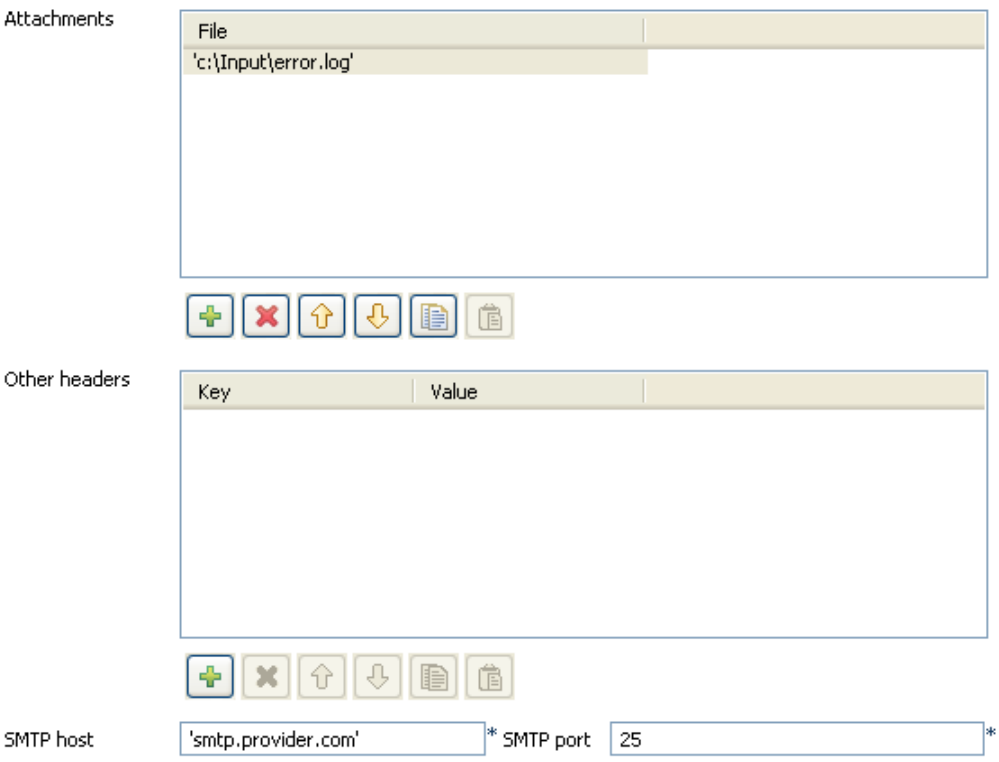
This scenario creates a three-component Job which sends an email to defined recipients when an error occurs.



- Drop the following components from your **Palette** to the design workspace: **tFileInputDelimited**, **tFileOutputXML**, **tSendMail**.
- Define **tFileInputdelimited** properties. Related topic: section *tFileInputDelimited*.
- Right-click on the **tFileInputDelimited** component and select *Row* > *Main*. Then drag it onto the **tFileOutputXML** component and release when the plug symbol shows up.
- Define **tFileOutputXML** properties.
- Drag a **Run on Error** link from **tFileDelimited** to **tSendMail** component.
- Define the **tSendMail** component properties:



- Enter the recipient and sender email addresses, as well as the email subject.
- Enter a message containing the error code produced using the corresponding global variable. Access the list of variables by pressing **Ctrl+Space**.
- Add attachments and extra header information if any. Type in the SMTP information.



In this scenario, the file containing data to be transferred to XML output cannot be found. **tSendmail** runs on this error and sends a notification email to the defined recipient.

tSetKerberosConfiguration

tSetKerberosConfiguration properties

Component family	Internet	
Function	**tSetKerberosConfiguration** is designed to configure Kerberos authentication for enhanced security of network communications. For more information on the Kerberos protocol, go to http://www.kerberos.org.	
Purpose	**tSetKerberosConfiguration** allows you to enter the relevant information for Kerberos authentication.	
Basic settings	*KDC Server*	Address of the Key Distribution Center (KDC) server.
	Realm	Name of the Kerberos realm.
	Username and Password	Kerberos authentication credentials.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is typically used as a sub-job by itself and is used along with **tSoap**.	
Limitation	n/a	

Related scenarios

No scenario is available for this component.

tSetKeystore

tSetKeystore properties

Component family	Internet	
Function	**tSetKeystore** submits authentication data of a truststore with or without keystore to validation for the SSL connection.	
Purpose	This component allows you to set the authentication data type between **PKCS 12** and **JKS**.	
Basic settings	*TrustStore type*	Select the type of the TrustStore to be used. It may be **PKCS 12** or **JKS**.
	TrustStore file	Type in the path, or browse to the certificate TrustStore file (including filename) that contains the list of certificates that the client trusts.
	TrustStore password	Type in the password used to check the integrity of the TrustStore data.
	Need Client authentication	Select this check box to validate the keystore data. Once doing so, you need complete three fields: - **KeyStore type**: select the type of the keystore to be used. It may be **PKCS 12** or **JKS**. - **KeyStore file**: type in the path, or browse to the file (including filename) containing the keystore data. - **KeyStore password**: type in the password for this keystore.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is used standalone.	
Connections		Outgoing links (from one component to another): **Trigger**: Run if; On Subjob Ok, On Subjob Error, On Component Ok; On Component Error. Incoming links (from one component to another): **Trigger**: Run if, On Subjob Ok, On Component Ok, On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a.	

Scenario: Extracting customer information from a private WSDL file

This scenario describes a three-component Job that connects to a private WSDL file in order to extract customer information.

The WSDL file used in this Job accesses the corresponding web service under the SSL protocol. For this purpose, the most relative code in this file reads as follows :

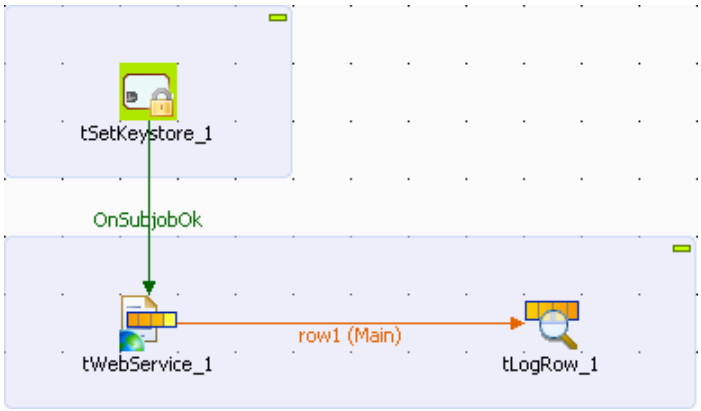
```
<wsdl:port name="CustomerServiceHttpSoap11Endpoint"
binding="ns:CustomerServiceSoap11Binding">
           <soap:address location="https://192.168.0.22:8443/axis2/services/
CustomerService.CustomerServiceHttpSoap11Endpoint/"/>
       </wsdl:port>
```

Accordingly, we enter the following code in the *server.xml* file of Tomcat:

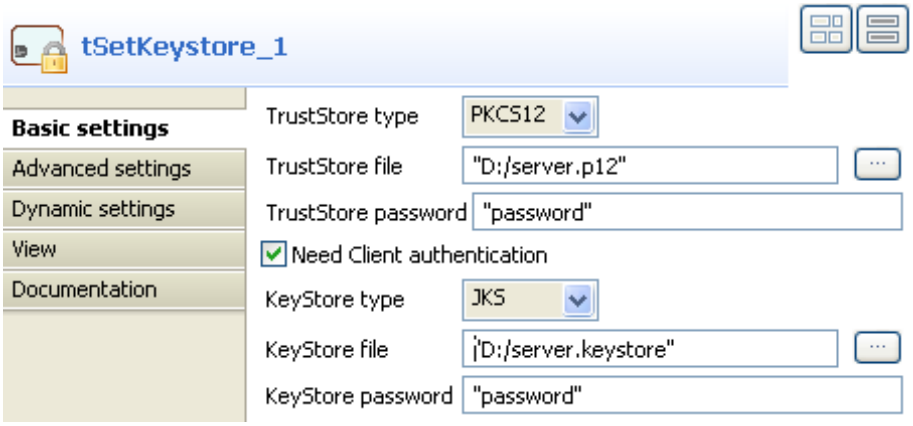
```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="true" sslProtocol="TLS"
    keystoreFile="D:/server.keystore" keystorePass="password"
    keystoreType="JKS"
    truststoreFile="D:/server.p12" truststorePass="password"
    truststoreType="PKCS12"
   />
```

So we need keystore files to connect to this WSDL file. To replicate this Job, proceed as follows:

- Drop the following components from the **Palette** onto the design workspace: **tSetKeystore**, **tWebService**, and **tLogRow**.

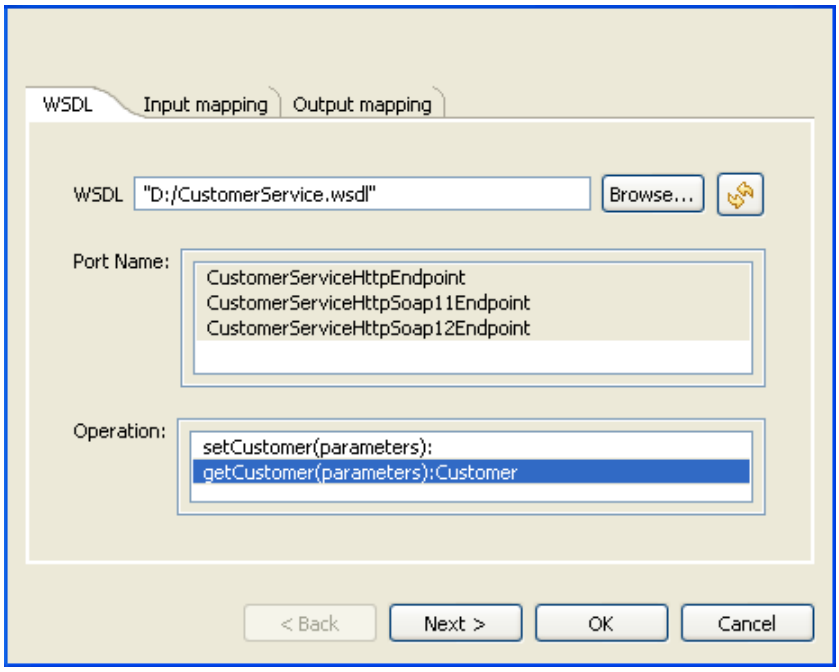


- Right-click **tSetKeystore** to open its contextual menu.
- In this menu, select **Trigger** > **On Subjob Ok** to connect this component to **tWebService**.
- Right-click **tWebService** to open its contextual menu.
- In this menu, select **Row** > **Main** to connect this component to **tLogRow**.
- Double-click **tSetKeystore** to open its **Basic settings** view and define the component properties.

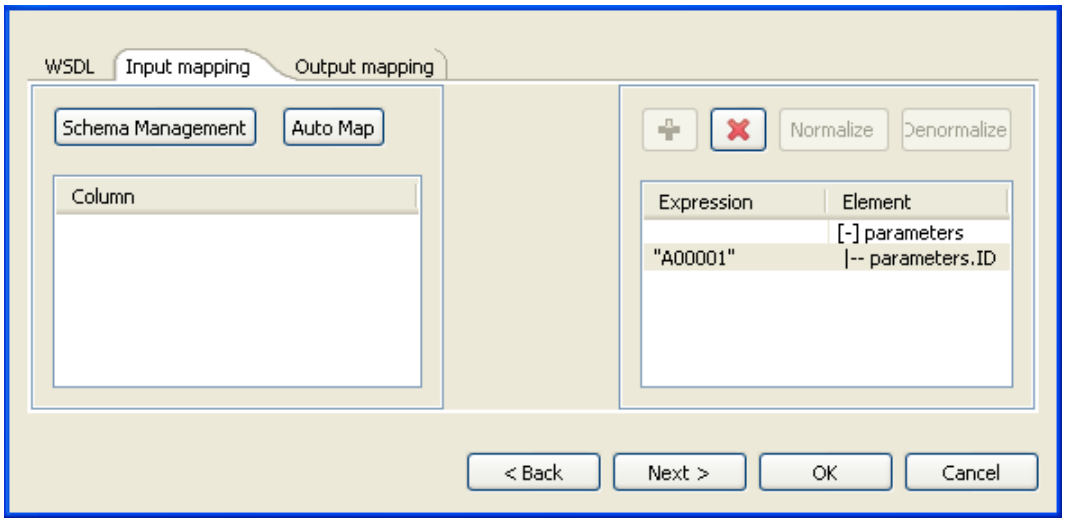


- In the **TrustStore type** field, select **PKCS12** from the drop-down list.
- In the **TrustStore file** field, browse to the corresponding truststore file. Here, it is *server.p12*.

- In the **TrustStore password** field, type in the password for this truststore file. In this example, it is *password*.
- Select the **Need Client authentication** check box to activate the keystore configuration fields.
- In the **KeyStore type** field, select **JKS** from the drop-down list.
- In the **KeyStore file** field, browse to the corresponding keystore file. Here, it is *server.keystore*.
- Double-click **tWebService** to open the component editor, or select the component in the design workspace and in the **Basic settings** view, click the three-dot button next to **Service configuration**.



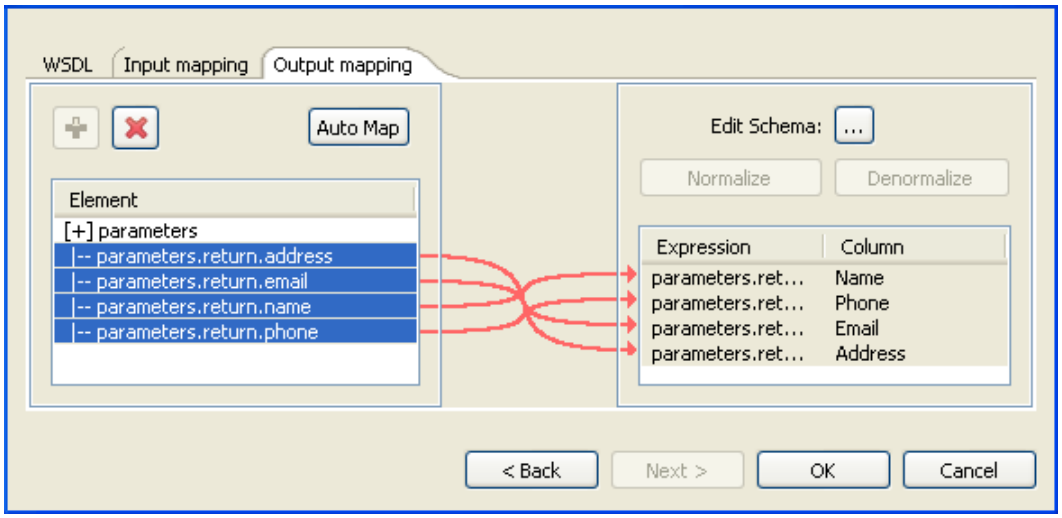
- In the **WSDL** field, browse to the private WSDL file to be used. In this example, it is *CustomerService.wsdl*.
- Click the refresh button next to the **WSDL** field to retrieve the WSDL description and display it in the fields that follow.
- In the **Port Name** list, select the port you want to use, *CustomerServiceHttpSoap11Endpoint* in this example.
- In the **Operation** list, select the service you want to use. In this example the selected service is *getCustomer(parameters):Customer*.
- Click **Next** to open a new view in the editor.



In the panel to the right of the **Input mapping** view, the input parameter of the service displays automatically. However, you can add other parameters if you select **[+] parameters** and then click the plus button on top to display the **[Parameter Tree]** dialog box where you can select any of the listed parameters.

The Web service in this example has only one input parameter, **ID**.

- In the **Expression** column of the *parameters.ID* row, type in the customer ID of your interest between quotation marks. In this example, it is *A00001*.
- Click **Next** to open a new view in the editor.

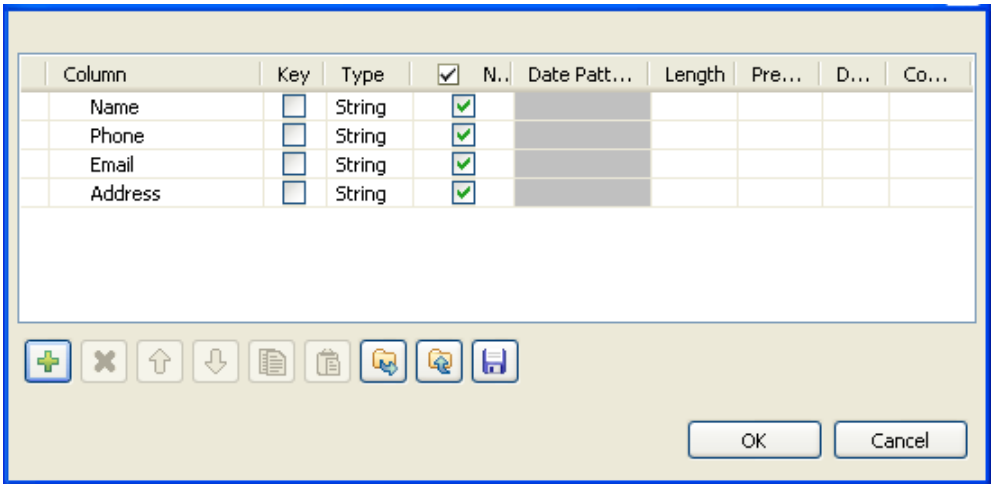


In the **Element** list to the left of the view, the output parameter of the web service displays automatically. However, you can add other parameters if you select **[+] parameters** and then click the plus button on top to display the **[Parameter Tree]** dialog box where you can select any of the parameters listed.

The Web service in this example has four output parameter: *return.address*, *return.email*, *return.name* and *return.phone*.

You now need to create a connection between the output parameter of the defined Web service and the schema of the output component. To do so:

- In the panel to the right of the view, click the three-dot button next to **Edit Schema** to open a dialog box in which you can define the output schema.



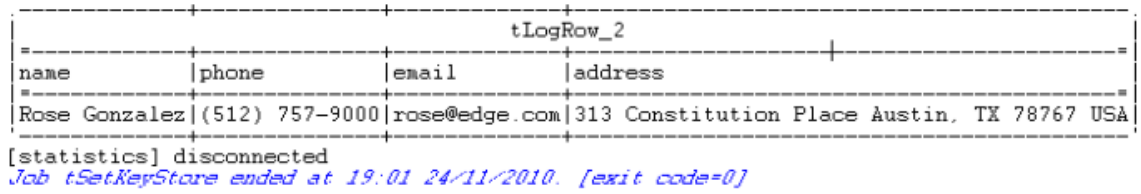
- In the schema editing dialog box, click the plus button to add four columns to the output schema.

- Click in each column and type in the new names, *Name*, *Phone*, *Email* and *Address* in this example. This will retrieve the customer information of your interest.
- Click **OK** to validate your changes and to close the schema editing dialog box.
- In the **Element** list to the right of the editor, drag each parameter to the field that corresponds to the column you have defined in the schema editing dialog box.

If available, use the **Auto map!** button, located at the bottom left of the interface, to carry out the mapping operation automatically.

- Click **OK** to validate your changes and to close the editor.
- In the design workspace, double-click **tLogRow** to open its **Basic settings** view and define its properties.
- Click **Sync columns** to retrieve the schema from the preceding component.
- Save your Job and press **F6** to execute it.

The information of the customer with ID *A00001* is returned and displayed in the console of *Talend Open Studio for Big Data*.



tSetProxy

tSetProxy properties

Component family	Internet	
Function	**tSetProxy** is designed to for proxy setup.	
Purpose	**tSetProxy** allows you to enter the relevant information for proxy setup.	
Basic settings	*Proxy type*	List of proxy protocols.
	Proxy host	Address of the proxy server.
	Proxy port	Number of the proxy port.
	Proxy user	Username for proxy authentication.
	Proxy password	Password for proxy authentication.
	None proxy hosts	The hosts which should be connected to directly and not through the proxy server. The value can be a list of hosts, each separated by a |, and in addition a wildcard character (*) can be used for matching, e.g. "192.168.0.* | localhost".
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Typically used as a sub-job by itself, **tSetProxy** is deployed along with other Internet components.	
Limitation	n/a	

Related scenarios

No scenario is available for this component.

tSocketInput

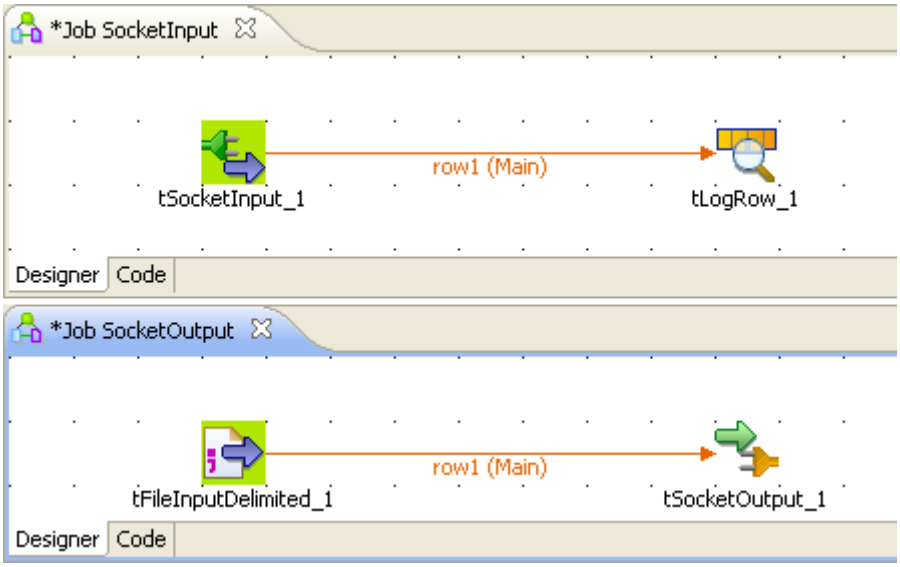
tSocketInput properties

Component family	Internet	
Function	**tSocketInput** component opens the socket port and listens for the incoming data.	
Purpose	**tSocketInput** component is a listening component, allowing to pass data via a defined port	
JAVA Basic settings	*Host name*	Name or IP address of the Host server
	Port	Listening port to open
	Timeout	Number of seconds for the port to listen before closing.
	Uncompress	Select this check box to unzip the data if relevant
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Rejects** link.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Escape Char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Encoding type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
JAVA Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component opens a point of access to a workstation or server. This component starts a Job and only stops after the time goes out.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Passing on data to the listening port

The following scenario describes two Jobs aiming at passing data via a listening port. The first Job (*SocketInput*) opens the listening port and waits for the data to be sent over. The second Job (*SocketOutput*) passes delimited data from a file to a defined port number corresponding to the listening port.

Another application for the Socket components would be to allow controlled communication between servers which cannot communicate directly.



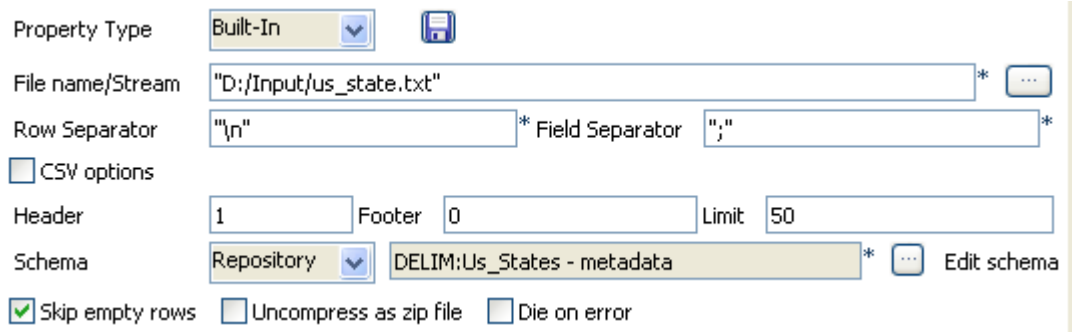
Dropping and linking components

1. For the first Job, drop a **tSocketInput** component and a **tLogRow** component from the **Palette** to the design workspace, and link them using a **Row** > **Main** connection.

2. For the second Job, drop a **tFileInputDelimited** component and a **tSocketOutput** component from the **Palette** to the design workspace, and link them using a **Row** > **Main** connection.

Configuring the Jobs

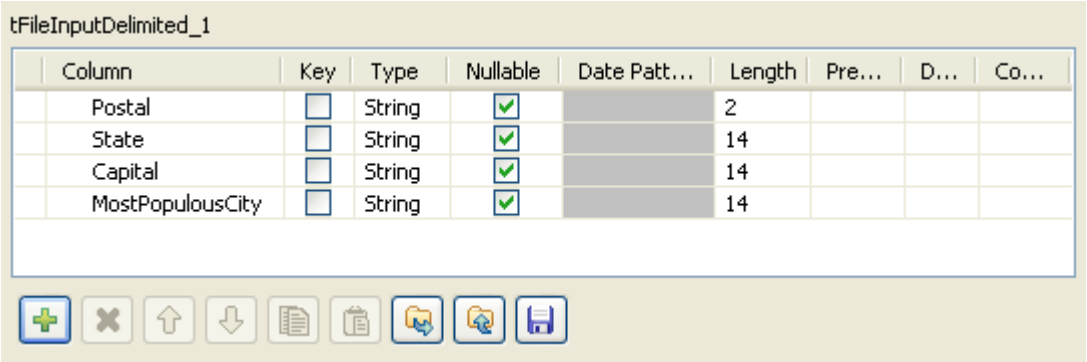
1. On the second Job, select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.



*For Talend Open Studio for Big Data, the **Schema** of components is always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

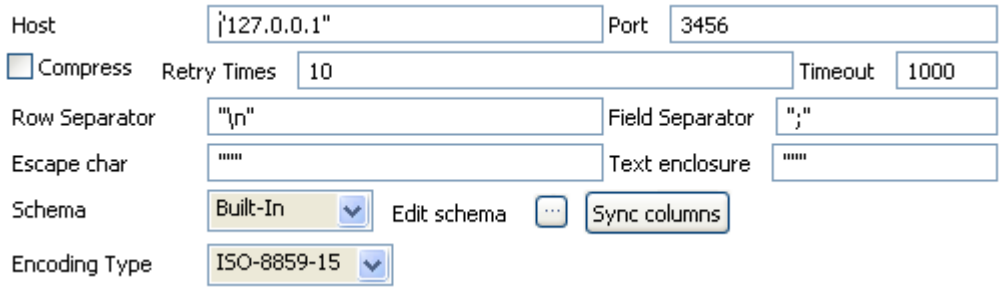
2. In **File Name**, browse to the file, and fill the **Row**, **Field separators**, and **Header** fields according to the input file used.

3. Describe the **Schema** of the data to be passed on to the **tSocketOutput** component.

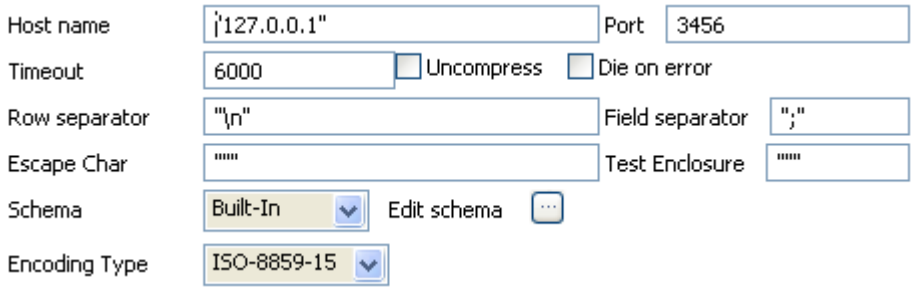


The schema should be propagated automatically to the output component.

4. Select the **tSocketOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.



5. Define the **Host** IP address and the **Port** number where the data will be passed on to.
6. Set the number of retries in the **Retry** field and the amount of time (in seconds) after which the Job will time out.
7. Now on the other Job (*SocketInput*) design, define the parameters of the **tSocketInput** component.



8. Define the **Host** IP address and the listening **Port** number where the data are passed on to.
9. Set the amount of time (in seconds) after which the Job will time out.
10. Edit the schema and set it to reflect the whole or part of the other Job's schema.

Executing the Jobs

1. Press **F6** to execute this Job (SocketInput) first, in order to open the listening port and prepare it to receive the passed data.
2. Before the time-out, launch the other Job (SocketOutput) to pass on the data.

   The result displays on the **Run** view, along with the opening socket information.

```
Starting job SocketInput at 17:53 04/02/2008.
socket connected
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
```

tSocketOutput

tSocketOutput properties

Component family	Internet	
Function	**tSocketOutput** component writes data to a listening port.	
Purpose	**tSocketOutput** sends out the data from the incoming flow to listening socket port.	
Basic settings	*Host name*	Name or IP address of the Host server
	Port	Listening port to open
	Compress	Select this check box to zip the data if relevant.
	Retry times	Number of retries before the Job fails.
	Timeout	Number of seconds for the port to listen before closing.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Escape Char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Encoding type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component opens a point of access to a workstation or server. This component starts a Job and only stops after the time goes out.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Related Scenario

For use cases in relation with **tSocketOutput**, see section *Scenario: Passing on data to the listening port*

tSOAP

tSOAP properties

Component family	Internet	
Function	**tSOAP** sends the defined SOAP message with the given parameters to the invoked Web service and returns the value as defined, based on the given parameters.	
Purpose	This component calls a method via a Web service in order to retrieve the values of the parameters defined in the component editor.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. This component always uses a built-in, read-only schema. By default, the schema contains three **String** type columns: - **Header**: stores the SOAP message header of the response from the server end. - **Body**: stores the SOAP message body of the response from the server end. - **Fault**: stores the error information when an error occurs during the SOAP message processing. If the **Output in Document** check box is selected, the schema then contains only one **Document** type column named **Soap**, which stores the whole response SOAP message in the XML format. Click **Edit schema** to view the schema structure. *Changing the schema type may result in loss of the schema structure and therefore failure of the component.*
	Use NTLM	Select this check box if you want to use the NTLM authentication protocol. **Domain**: Name of the client domain.
	Need authentication	Select this check box and enter a user name and a password in the corresponding fields if this is necessary to access the service.
	Use http proxy	Select this check box if you are using a proxy server and fill in the necessary information.
	Trust server with SSL	Select this check box to validate the server certificate to the client via an SSL protocol and fill in the corresponding fields: **TrustStore file**: enter the path (including filename) to the certificate TrustStore file that contains the list of certificates that the client trusts. **TrustStore password**: enter the password used to check the integrity of the TrustStore data.
	ENDPOINT	Type in the URL address of the invoked Web server.
	SOAP Action	Type in the URL address of the SOAPAction HTTP header field to be used to identify the intent of the SOAP HTTP request.
	SOAP version	Select the version of the SOAP system you are using.

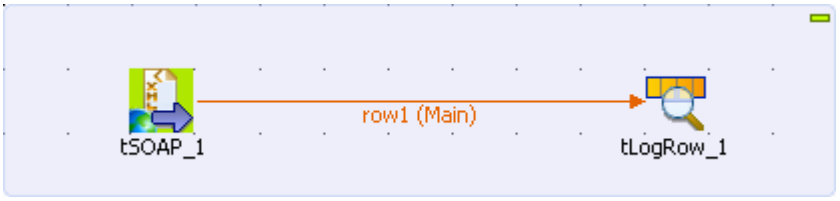
		The required SOAP Envelope varies among versions.
	Use a message from the input schema	Select this check box to read a SOAP message from the preceding component to send to the invoked Web service. When this check box is selected, the **SOAP message** field becomes a drop-down list allowing you to select a **Document** type column to read an input XML file. *This option makes sense only when the **tSOAP** component is connected with an input component the schema of which contains a **Document** type column to read a valid SOAP message.*
	Output in Document	Select this check box to output the response message in XML format.
	SOAP message	Type in the SOAP message to be sent to the invoked Web service. The global and context variables can be used when you write a SOAP message. For further information about the context variables, see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Use Kerberos*	Select this check box to choose a **tSetKerberosConfiguration** component from the **Kerberos configuration** list. The **OnSubjobOk** trigger of **tSetKerberosConfiguration** should be used for connection with **tSoap**.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as an input or as an intermediate component.	
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Component Ok; On Component Error. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	N/A	

Scenario 1: Extracting the weather information using a Web service

This scenario describes a two-component Job that uses a Web service to retrieve the weather information of a given American city.

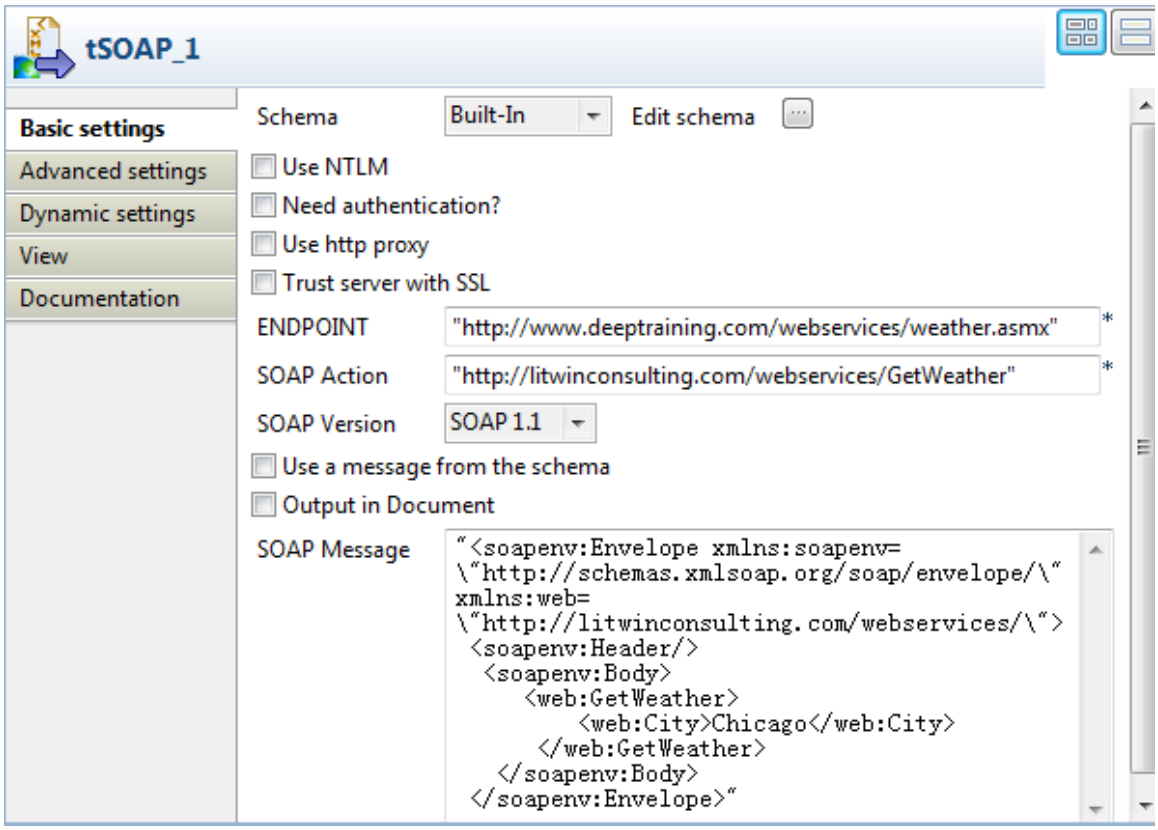
The Web service to be used is http://www.deeptraining.com/webservices/weather.asmx.

1. Drop the following components from the **Palette** onto the design workspace: **tSOAP** and **tLogRow**.



2. Right click **tSOAP**, select **Row > Main** from the contextual menu, and click **tLogRow** to connect the components together using a **Main Row** link.

3. Double-click **tSOAP** to open its **Basic settings** view and define the component properties.



4. In **ENDPOINT** field, type in or copy-paste the URL address of the Web service to be used between the quotation marks: *"http://www.deeptraining.com/webservices/weather.asmx"*.

5. In the **SOAP Action** field, type in or copy-paste the URL address of the SOAPAction HTTP header field that indicates that you want to retrieve the weather information: *http://litwinconsulting.com/webservices/GetWeather*.

You can see this address by looking at the WSDL for the Web service you are calling. For the Web service of this example, in a web browser, append `?wsdl` on the end of the URL of the Web service used in the **ENDPOINT** field, open the corresponding web page, and then see the SOAPAction defined under the operation node:

```
<wsdl:operation name="GetWeather">
<soap:operation soapAction="http://litwinconsulting.com/webservices/GetWeather"
style="document"/>
```

6. In the **SOAP version** field, select the version of the SOAP system being used. In this scenario, the version is *SOAP 1.1*.

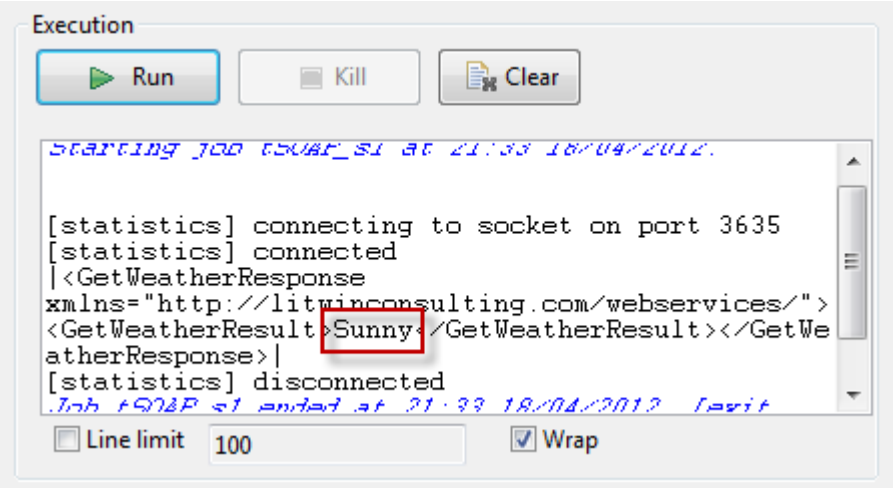
7. In the **SOAP message** field, enter the XML-format message used to retrieve the weather information from the invoked Web service. In this example, the weather information of Chicago is needed, so the message is:

```
"<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
```

```
xmlns:web=\"http://litwinconsulting.com/webservices/\">
 <soapenv:Header/>
  <soapenv:Body>
     <web:GetWeather>
         <web:City>Chicago</web:City>
      </web:GetWeather>
   </soapenv:Body>
 </soapenv:Envelope>"
```

8. Save your Job and press **F6** to execute it.

   The weather of Chicago is returned and displayed in the console of the **Run** view.



Scenario 2: Using a SOAP message from an XML file to get weather information and saving the information to an XML file

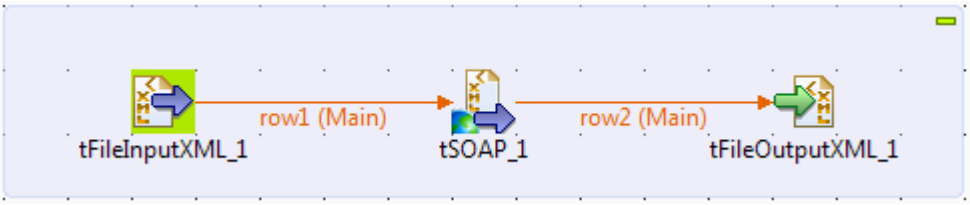
This scenario describes a three-component Job that uses a SOAP message from an input XML file to invoke a Web service for weather information of Paris, and writes the response to an XML file.

As in the previous scenario, the Web service to be used is http://www.deeptraining.com/webservices/weather.asmx.

Dropping and linking the components

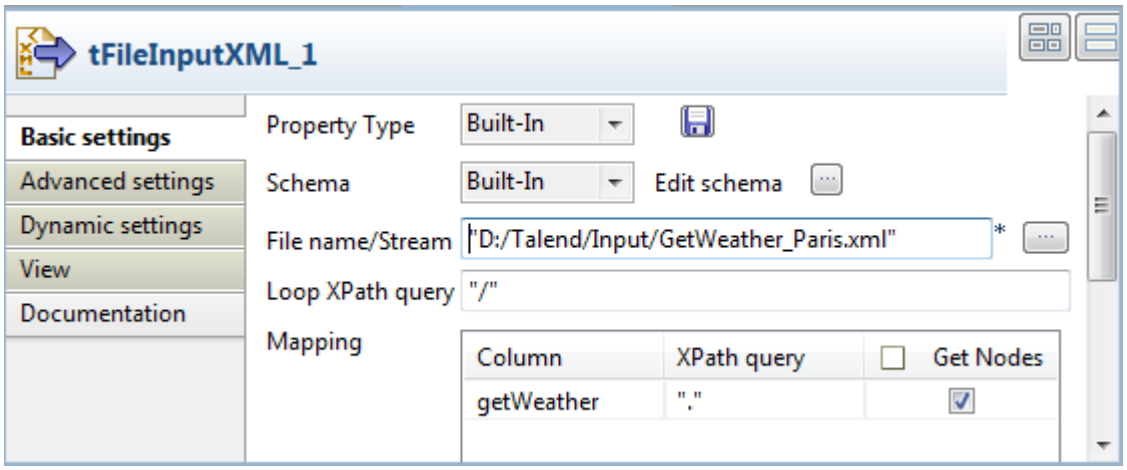
1. Drop the following components from the **Palette** onto the design workspace: **tFileInputXML**, **tSOAP**, and **tFileOutputXML**.

2. Connect the components using **Main** > **Row** links.

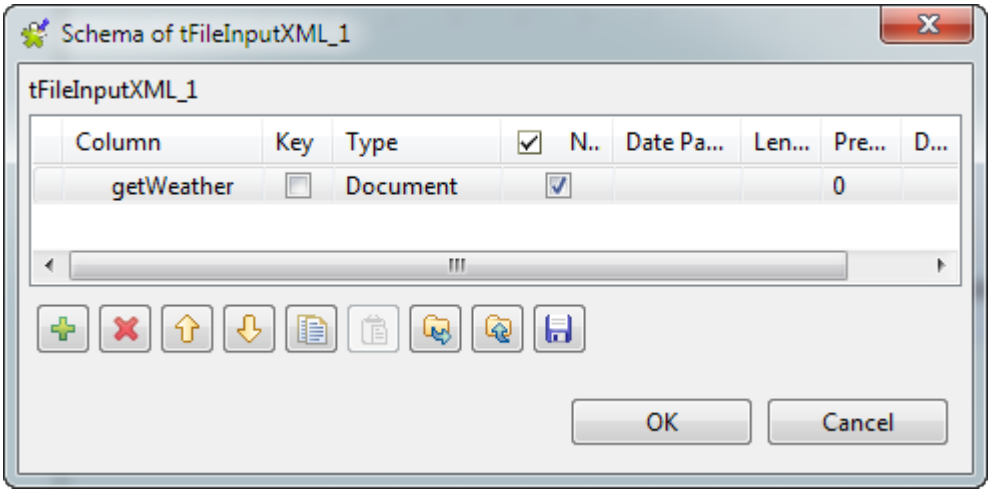


Configuring the input component

1. Double-click the **tFileInputXML** component to open its **Basic settings** view and define the component properties.



2. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box.



3. Click the **[+]** button to add a column, give it a name, *getWeather* in this example, and select **Document** from the **Type** list. Then, click **OK** to close the dialog box.

4. In the **File name/Stream** field, enter the path to the input XML file that contains the SOAP message to be used, or browse to the path by clicking the **[...]** button.

   The input file contains the following SOAP message:

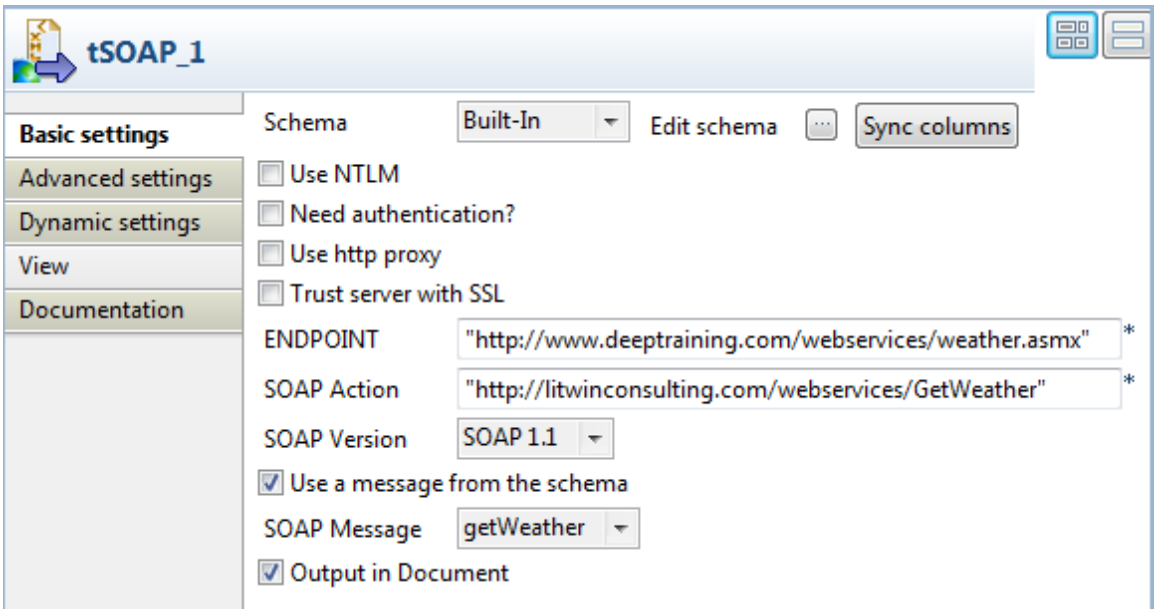
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:web="http://litwinconsulting.com/webservices/">
    <soapenv:Header/>
    <soapenv:Body>
        <web:GetWeather>
            <web:City>Paris</web:City>
        </web:GetWeather>
    </soapenv:Body>
</soapenv:Envelope>
```

5. In the **Loop XPath query** field, enter `"/"` to define the root as the loop node of the input file structure.

6. In the **Mapping** table, fill the **XPath query** column with `"."` to extract all data from context node of the source, and select the **Get Nodes** check box to build a **Document** type data flow.

Configuring the Web service via the tSOAP component

1. Double-click the **tSOAP** component to open its **Basic settings** view and define the component properties.



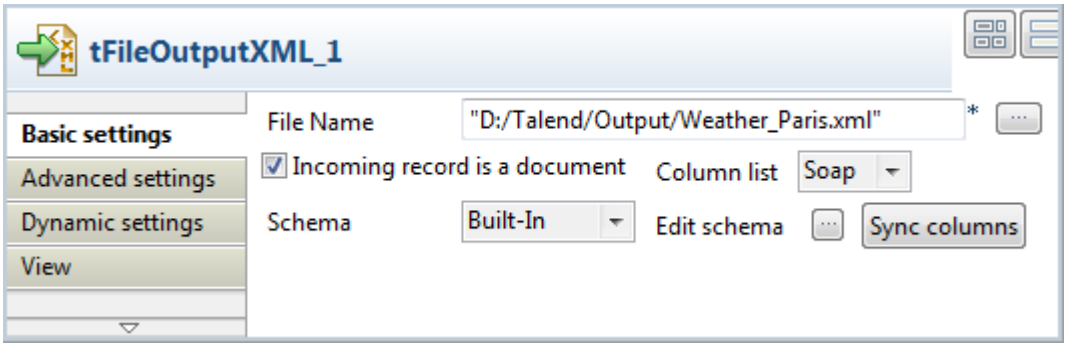
2. In **ENDPOINT** field, enter or copy-paste the URL address of the Web service to be used between the quotation marks: *"http://www.deeptraining.com/webservices/weather.asmx"*.

3. In the **SOAP Action** field, enter or copy-paste the URL address of the SOAPAction HTTP header field that indicates that you want to retrieve the weather information: *http://litwinconsulting.com/webservices/GetWeather*.

4. Select the **Use a message from the input schema** check box, and select a **Document** type column from the **SOAP Message** list to read the SOAP message from the input file to send to the Web service. In this example, the input schema has only one column, *getWeather*.

Configuring the output component

1. Double-click the **tFileOutputXML** component to open its **Basic settings** view.



2. In the **File Name** field, enter the path to the output XML file.

3. Select the **Incoming record is a document** check box to retrieve the incoming data flow as an XML document. Note that a **Column list** appears allowing you choose a column to retrieve data from. In this example, the schema contains only one column.

Executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6**, or click **Run** on the **Run** tab to execute the Job.

   The weather of Paris is returned and the information is saved in the defined XML file.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
- <soap:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    - <soap:Body>
        - <GetWeatherResponse xmlns="http://litwinconsulting.com/webservices/">
            <GetWeatherResult>Rain</GetWeatherResult>
          </GetWeatherResponse>
      </soap:Body>
  </soap:Envelope>
```

tWebService

This component requires an Oracle JDK to be functional.

tWebService properties

Component family	Internet	
Function	**tWebservice** calls the defined method from the invoked Web service and returns the class as defined, based on the given parameters.	
Purpose	This component calls a method via a Web service in order to retrieve the values of the parameters defined in the component editor.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally. Enter properties manually
		Repository: Select the repository file where properties are stored. The fields that come after are pre-filled in using the fetched data.
	Service configuration	Click the three-dot button next to the field to open the component editor. In this editor, you can: -select the Web service you want to use, -configure the input parameters of the Web service, -configure the output parameters of the Web service. These parameters will be used to retrieve and output specific data.
	Mapping links display as	**Auto**: By default, the links between the input and output schemas and the Web service parameters are in the form of curves. **Curves**: Links between the schema and the Web service parameters are in the form of curve. **Lines**: Links between the schema and the Web service parameters are in the form of straight lines. This option slightly optimizes performance.
	Connection Time out	Set a value in seconds for Web service connection time out.
	Receive Time out	Set a value in seconds for server answer.
	Input schema	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. This field is used to process the input schema. The option for this schema may be - **Built-in**: No property data stored centrally. - **Repository**: Select the Repository file where properties are stored. The following fields are pre-filled in using fetched data. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.

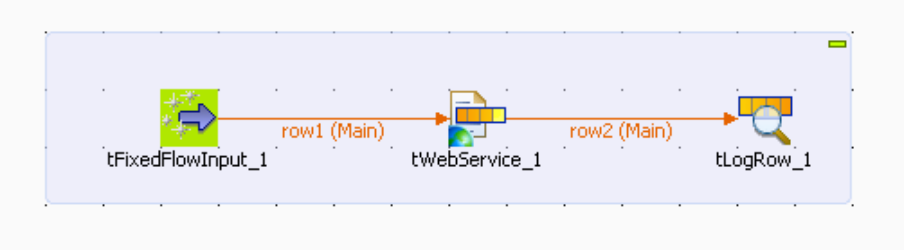
	Edit Schema	Click the three-dot button to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
	Sync columns	This button is available when an input link has been created. Click this button to retrieve the schema from the previous component connected in the Job.
	Output schema	This field is used to process the output schema. The schema is either built-in or remote in the Repository and is configured the same way as the input schema is. *The input schema is not necessarily identical with the output schema.* If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
	Use NTLM	Select this check box if you want to use the NTLM authentication protocol. **Domain**: Name of the client domain, **Host**: Client IP address.
	Need authentication	Select this check box and enter a username and a password in the corresponding fields if this is necessary to access the service.
	Use http proxy	Select this check box if you are using a proxy server and fill in the necessary information.
	Trust server with SSL	Select this check box to validate the server certificate to the client via an SSL protocol and fill in the corresponding fields: **TrustStore file**: enter the path (including filename) to the certificate TrustStore file that contains the list of certificates that the client trusts. **TrustStore password**: enter the password used to check the integrity of the TrustStore data.
	Die on error	Clear this check box to skip the rows in error and to complete the process for the error free rows.
Advanced settings	*Temporary folder (for wsdl2java)*	Set or browse to a temporary folder that you configured in order to store the wsdl files.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
	Enable parallel execution	Select this check box to perform high-speed data processing, that is treating multiple data flows simultaneously. For further information, see *Talend Open Studio for Big Data User Guide*.
Usage	This component can be used as an input or as an intermediate component. It must be linked to an output component.	
Limitation	A JDK is required for this component to operate.	

Scenario: Extracting a name list using a Web service

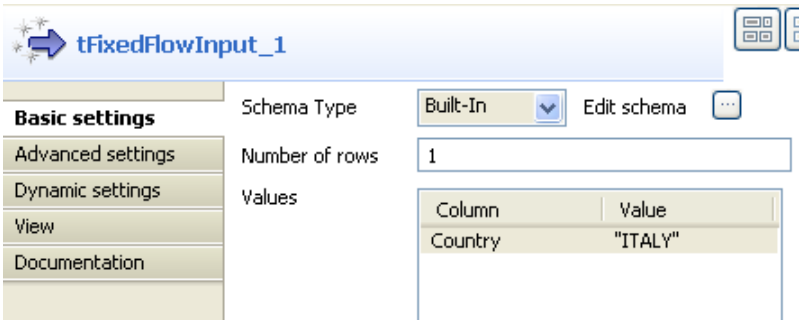
This scenario describes a three-component Job that uses a Web service to retrieve a list of all of the defenders who play for the football team of a defined country.

You can also call a Job exported as a Web service. For more information, see section *Scenario 2: Buffering output data on the webapp server* and section *Scenario 4: Calling a Job exported as Webservice in another Job*.

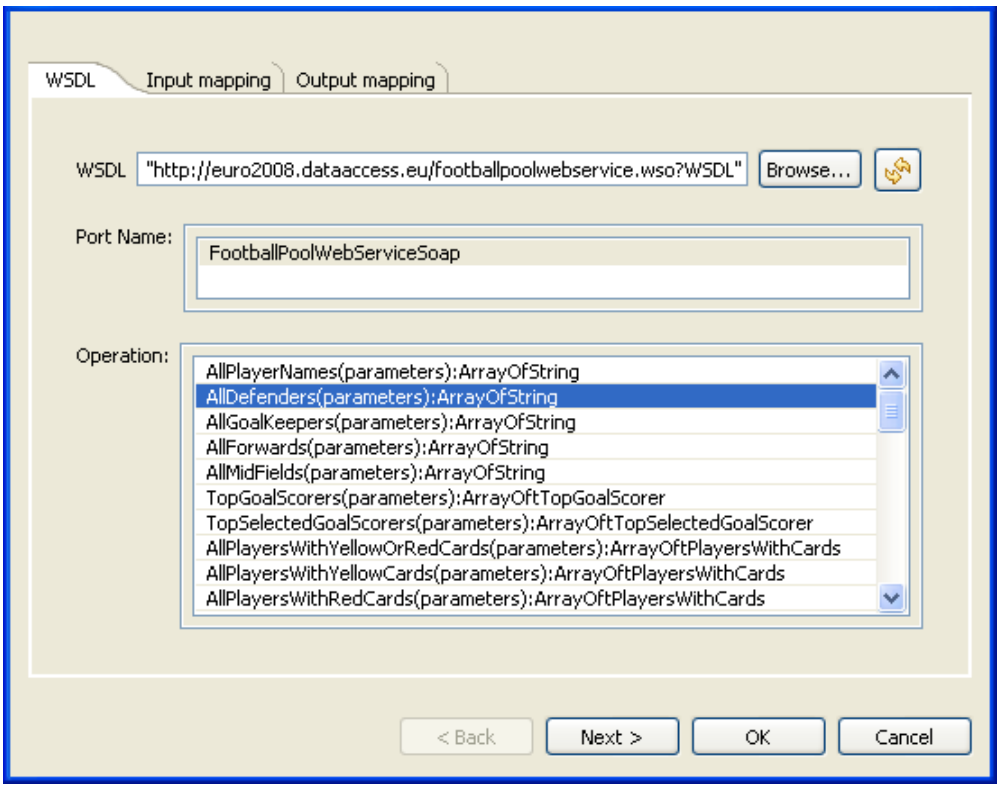
- Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput**, **tWebService**, and **tLogRow**.



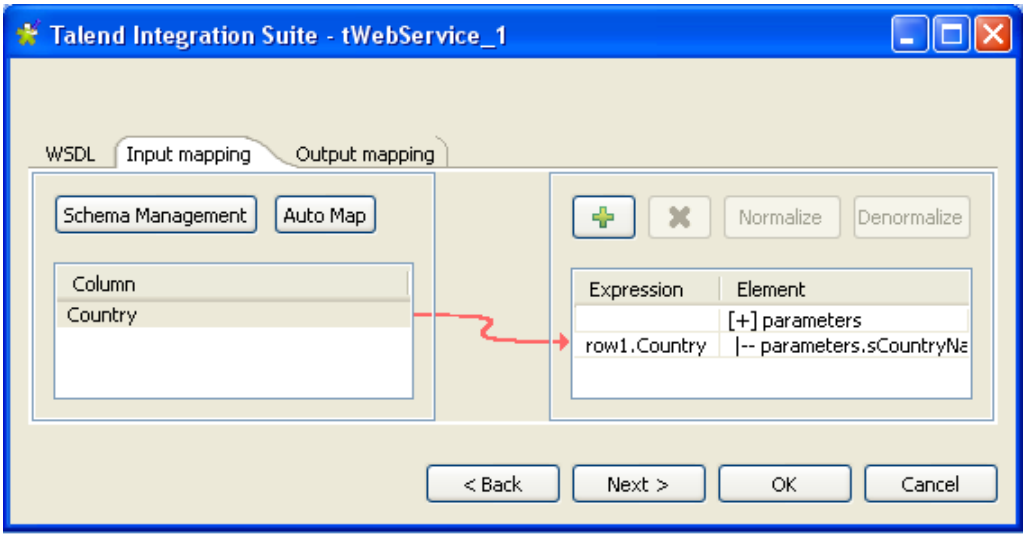
- Connect the components together using **Main Row** links.
- Double-click **tFixedFlowInput** to open its **Basic settings** view and define the component properties.



- Click the three-dot button next to the **Edit schema** field to open a dialog box where you can define the input schema.
- In the open dialog box, click the plus button to add a column to the schema.
- Click in the column and type in a name, *Country* in this example.
- Click **OK** to close the schema definition dialog box. The *Country* column displays in the **Values** table in the component **Basic settings** view.
- In the **Values** table, click in the **Value** column and enter the value of the *Country* column, *ITALY* in this example. This will retrieve the list of defenders of the Italian football team.
- Double-click **tWebService** to open the component editor, or select the component in the design workspace and in the **Basic settings** view, click the three-dot button next to **Service configuration**.



- Connect to the Web service you want to use in the Job.
- In the **WSDL** field, enter the Web service address or browse to it, if the WSDL is locally stored, by clicking the **[Browse...]** button.
- Click the refresh button next to the **WSDL** filed to retrieve the WSDL description and display it in the fields that follow.
- In the **Port Name** list, select the port you want to use, **FootballPoolWebServiceSoap** in this example.
- In the **Operation** list, select the service you want to use. In this example the selected service is **AllDefenders(parameters):ArrayOfString** .
- Click **Next** to open a new view in the editor.



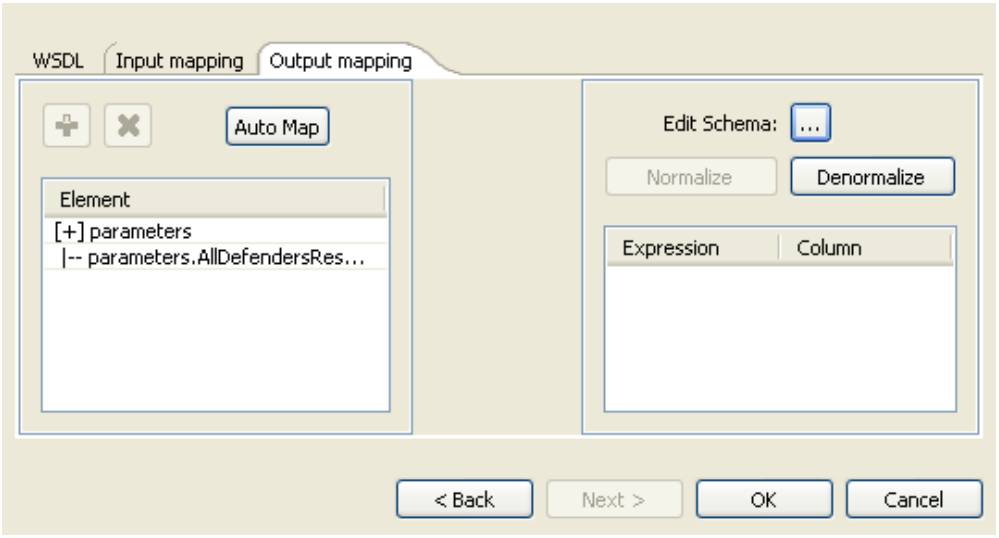
In the panel to the right of the **Input mapping** view, the input parameter of the service displays automatically. However, you can add other parameters if you select **[+] parameters** and then click the plus button on top to display the **[Parameter Tree]** dialog box where you can select any of the listed parameters.

The Web service in this example has only one input parameter, **sCountryName**.

If available, use the **Auto map!** button, located at the bottom left of the interface, to carry out the mapping operation automatically.

You now need to create a connection between the input schema and the input parameter of the defined Web service. To do so:

- In the **Column** list, drag the column in the input schema you want to link to the input parameter of the Web service to the corresponding parameter in the panel to the right.
- Click **Next** to open a new view in the editor.

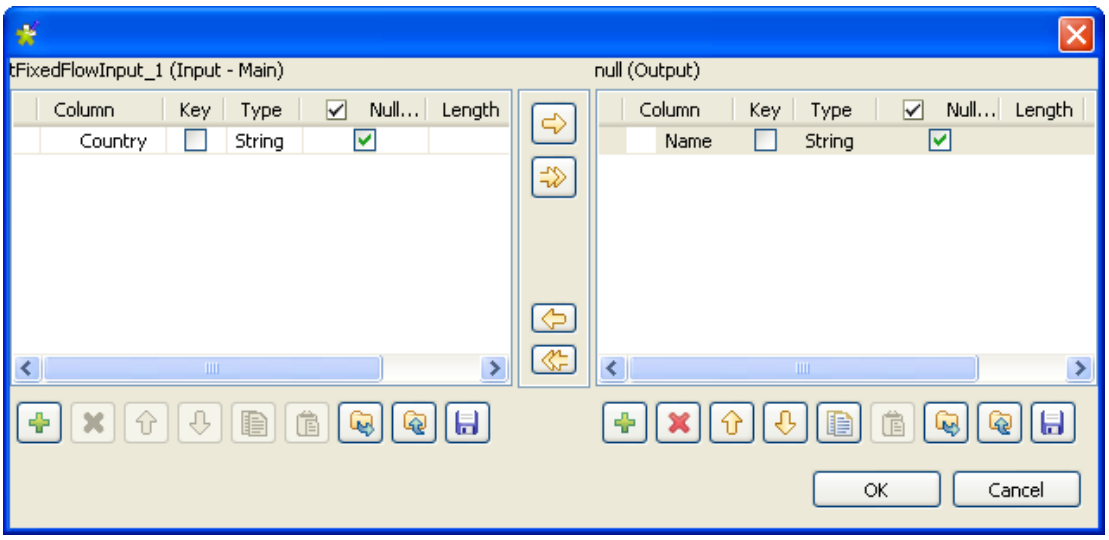


In the **Element** list to the left of the view, the output parameter of the web service displays automatically. However, you can add other parameters if you select **[+] parameters** and then click the plus button on top to display the **[Parameter Tree]** dialog box where you can select any of the parameters listed.

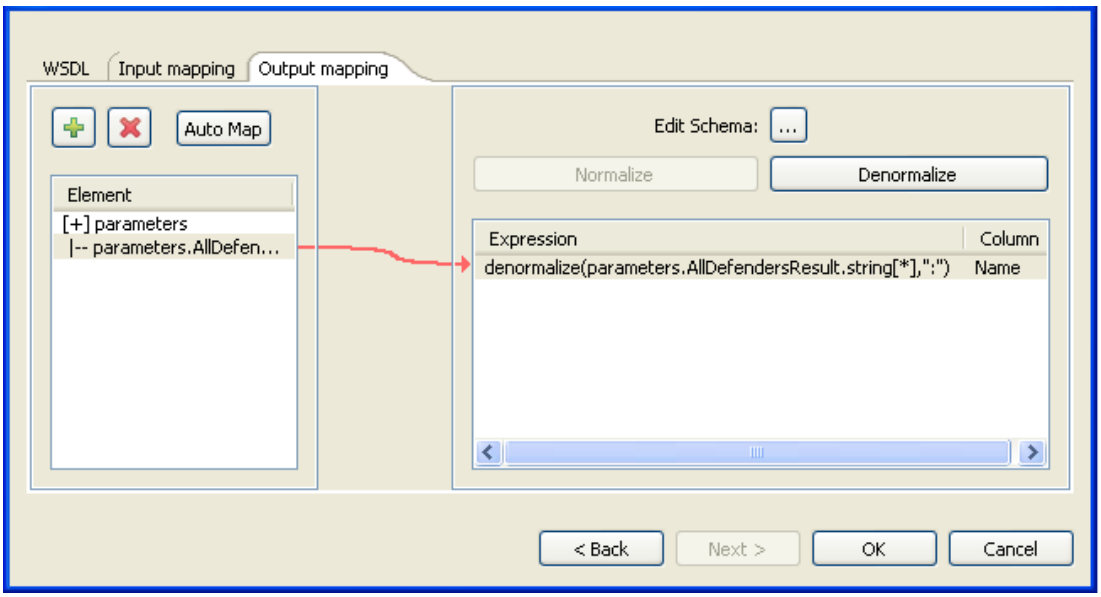
The Web service in this example has only one output parameter: **AllDefendersResult.string**.

You now need to create a connection between the output parameter of the defined Web service and the schema of the output component. To do so:

- In the panel to the right of the view, click the three-dot button next to **Edit Schema** to open a dialog box in which you can define the output schema.



- In the **Output** list to the right of the dialog box, click the plus button to add a column to the output schema.
- Click in the column and type in a name, *Name* in this example. This will retrieve the names of the defenders.
- Click **OK** to validate your changes and to close the schema definition dialog box.
- In the **Element** list to the right of the editor, drag **parameters.AllDefendersResult.string** to the field that corresponds to the **Name** column to the right of the editor.



If available, use the **Auto map!** button, located at the bottom left of the interface, to carry out the mapping operation automatically.

- Select this row in the panel to the right and click **Denormalize** in order to denormalize the returned data.
- Add `[*]` after the parameter in order to have the following code: `denormalize(parameters.AllDefendersResult.string[*],":")`. This will retrieve all data separated by a colon `":"`.
- Click **OK** to validate your changes and to close the editor.
- In the design workspace, double-click **tLogRow** to open its **Basic settings** view and define its properties.
- Click **Sync columns** to retrieve the schema from the preceding component.

- Save your Job and press **F6** to execute it.

The names of all defenders of the Italian football team are returned and displayed in the console of *Talend Open Studio for Big Data*.

tWebServiceInput

tWebServiceInput Properties

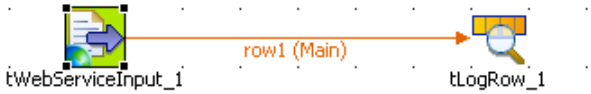
Component family	Internet	
Function	Calls the defined method from the invoked Web service, and returns the class as defined, based on the given parameters.	
Purpose	Invokes a Method through a Web service. To handle complex hierarchical data, use the advanced features of **tWebServiceInput** and provide Java code directly in the **Code** field of the **Advanced Settings** view.	
Basic settings	*Property type*	Either **Built-in** or **Repository**. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: No property data stored centrally.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	WSDL	Description of Web service bindings and configuration.
	Need authentication / Username and Password	Select this check box and: -enter a username and a password in the corresponding fields if this is necessary to access the service. Or, -select the **Windows authentication** check box and enter the windows domain in the corresponding field if this is necessary to access the service.
	Use http proxy	Select this check box if you are using a proxy server and fill in the necessary information.
	Trust server with SSL	Select this check box to validate the server certificate to the client via an SSL protocol and fill in the corresponding fields: **TrustStore file**: enter the path (including filename) to the certificate TrustStore file that contains the list of certificates that the client trusts. **TrustStore password**: enter the password used to check the integrity of the TrustStore data.
	Time out (second)	Set a value in seconds for Web service connection time out.
	Method Name	Enter the exact name of the Method to be invoked.

		The Method name MUST match the corresponding method described in the Web Service. The Method name is also case-sensitive.
	Parameters	Enter the parameters expected and the sought values to be returned. Make sure that the parameters entered fully match the names and the case of the parameters described in the method.
	Advanced Use	Select this check box to display the fields dedicated for the advanced use of **tWebServiceInput**: **WSDL2java**: click the three-dot button to generate **Talend** routines that hold the Java code necessary to connect and query the Web service. **Code**: replace the generated model Java code with the code necessary to connect and query the specified Web service using the code in the generated **Talend** routines. **Match Brackets**: select the number of brackets to be used to close the `for` loop based on the number of open brackets.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is generally used as a Start component. It must be linked to an output component.	
Limitation	n/a	

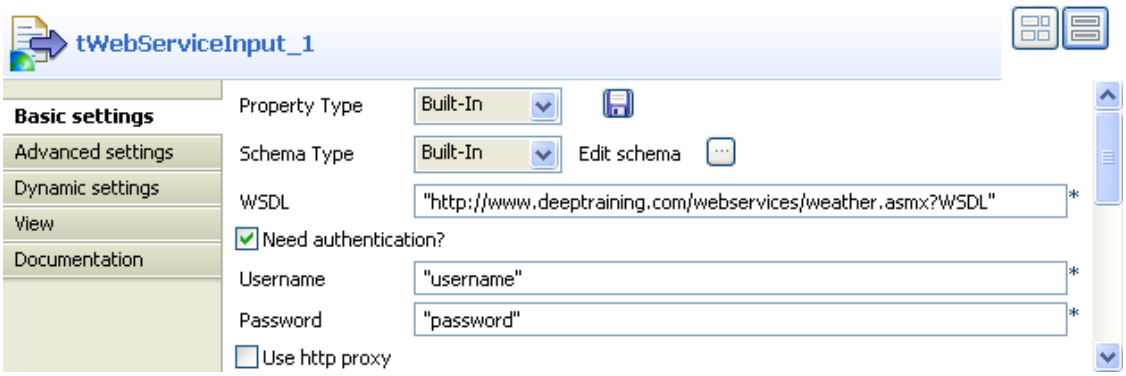
Scenario 1: Extracting images through a Web service

This scenario describes a two-component Job which uses a Web service method and displays the output on the **Run** console view.

The method retrieves a full URL as an input string and returns a string array of images from a given Web page.

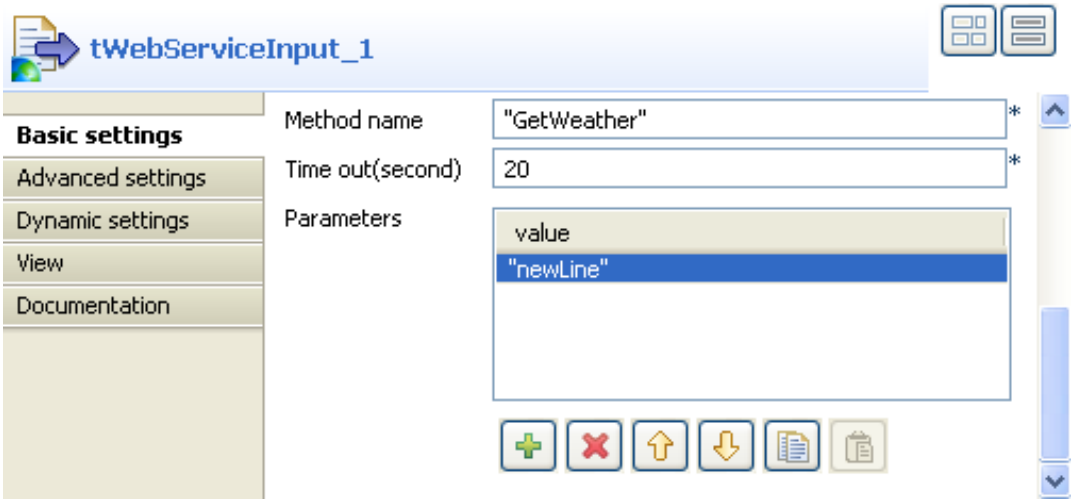


- Drop a **tWebServiceInput** component and a **tLogRow** component from the **Palette** onto the design workspace.
- On the **Component** view of the **tWebServiceInput** component, define the WSDL specifications, such as **End Point URI**, **WSDL** and **SOAPAction URI** where required.
- If the Web service you invoke requires authentication details, select the **Need authentication** check box and provide the relevant authentication information.



- If you are using a proxy server, select the **Use http proxy** check box and enter the necessary connection information.

- In the **Method Name** field, enter the method name as defined in the Web Service description. The name and the case of the method entered must match the corresponding Web service method exactly.



- In the **Parameters** area, click the plus **[+]** button to add a row to the table, then enter the exact name of the parameters which correspond to the method.
- In the **Value** column, type in the URL of the Website that the images are to be extracted from.
- Link the **tWebServiceInput** component to the standard output component, **tLogRow**.
- Then save your Job and press **F6** to execute it.

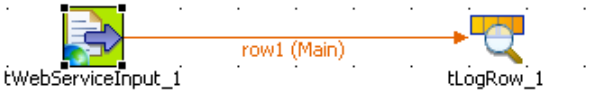
```
Starting job WebService at 16:30 06/06/2007.
<img src="http://us.a1.yimg.com/us.yimg.com/i/ww/beta/y3.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/webs.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/allys.gif"></img>
Job WebService ended at 16:30 06/06/2007. [exit code=0]
```

All of the images extracted from the Web site are returned as a list of URLs on the **Run** view.

Scenario 2: Reading the data published on a Web service using the tWebServiceInput advanced features

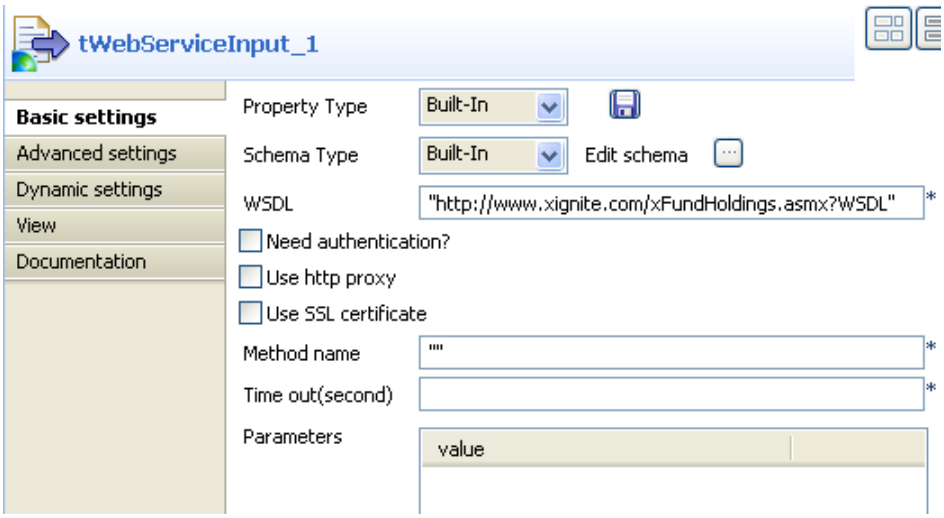
This scenario describes a two-component Job that retrieves a list of funds published by a financial Web service (distributed by *www.xignite.com*) and displays the output on the standard console (the **Run** view).

This scenario is designed for advanced users with basic knowledge of Java. Since the aim of this Job is to retrieve complex hierarchical data, you need to code the necessary functions in Java.



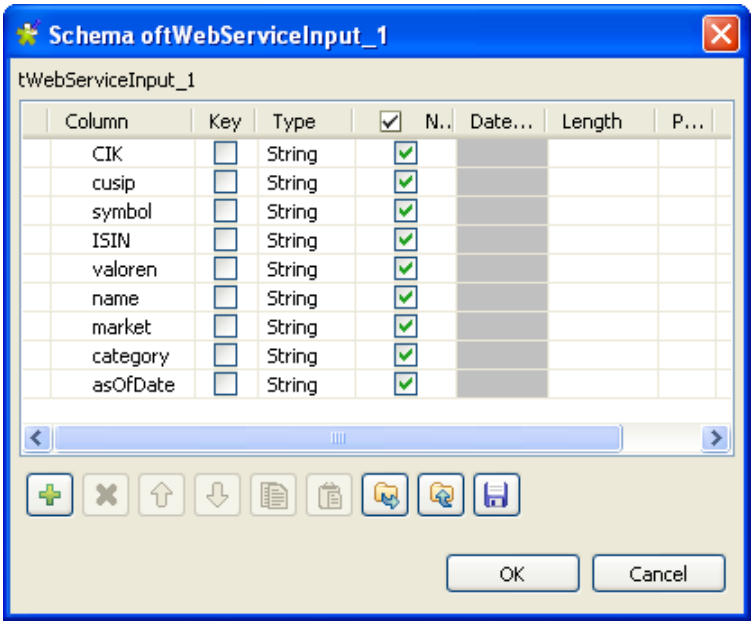
- Drop the following components from the **Palette** onto the design workspace: **tWebServiceInput** and **tLogRow**.

- Link the two components together using a **Row Main** connection.
- Double-click **tWebServiceInput** to show the **Component** view and set the component properties:



In the **Basic settings** view:

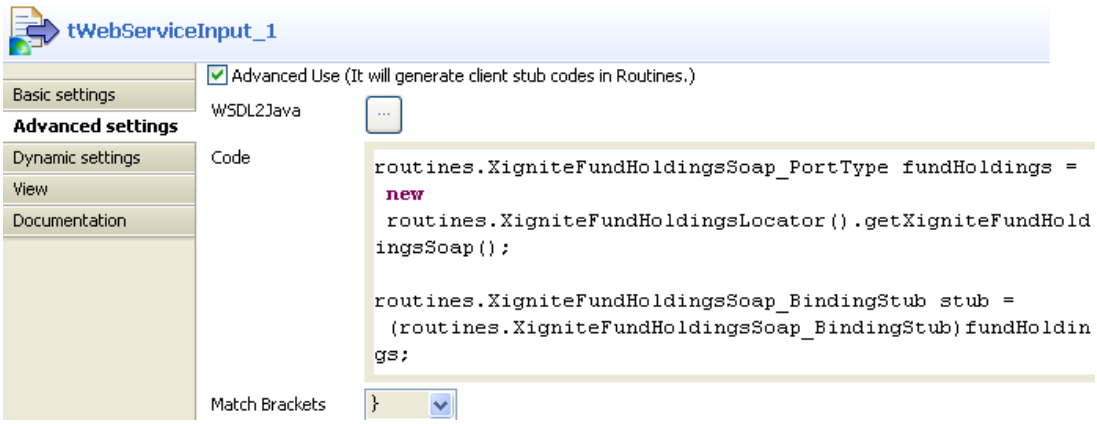
- In the **Property Type** list, select **Built-in** and complete the fields that follow manually.
- In the **Schema Type** list, select **Built-in** and click the **[...]** button to configure the data structure (schema) manually, as shown in the figure below:



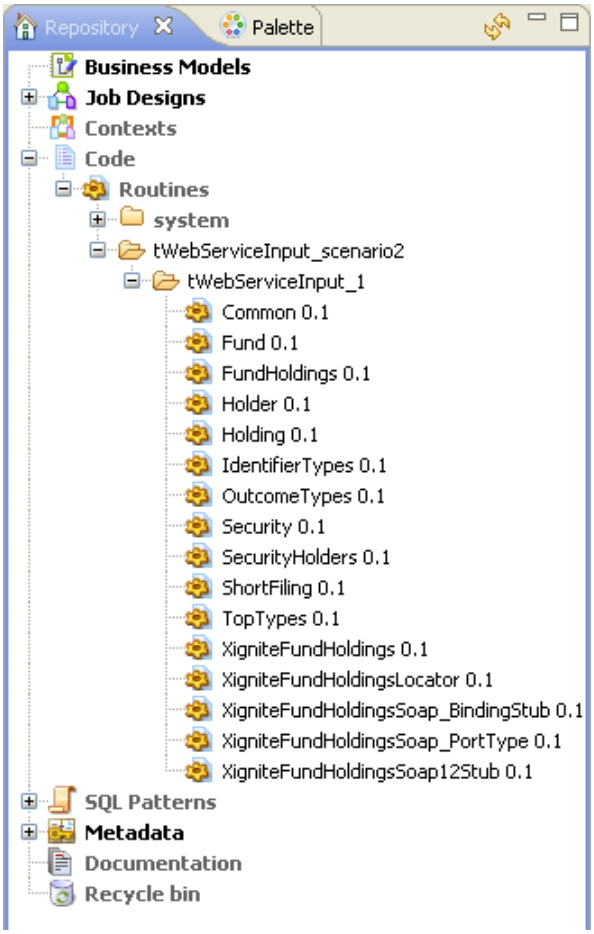
- Click **OK** to validate the schema and close the window.

  A dialog box opens and asks you if you want to propagate the modifications.

- Click **Yes**.
- In the **WSDL** field, enter the URL from which to get the WSDL.
- In the **Time out** field, enter the desired duration of the Web Service connection.
- Click the **Advanced settings** tab to display the corresponding view where you can set the **tWebServiceInput** advanced features:



- Select the check box next to **Advanced Use** to display the advanced configuration fields.
- Click the **[...]** button next to the **WSDL2Java** field in order to generate routines from the WSDL Web service.



The routines generated display automatically under **Code > Routines** in the **Repository** tree view. These routines can thus easily be called in the code to build the function required to fetch complex hierarchical data from the Web Service.

- Enter the relevant function in the **Code** field. By default, two examples of code are provided in the Code field. The first example returns one piece of data, and the second example returns several.
- In this scenario, several data are to be returned. Therefore, remove the first example of code and use the second example of code to build the function.

- Replace the pieces of code provided as examples with the relevant routines that have been automatically generated from the WSDL.
- Change `TalendJob_PortType` to the routine name ending with **_Port_Type**, such as: `XigniteFundHoldingsSoap_PortType`.
- Replace the various instances of `TalendJob` with a more relevant name such as the name of the method in use. In this use case: *fundHolding*
- Replace `TalendJobServiceLocator` with the name of the routine ending with **Locator**, such as: `XigniteFundHoldingLocator`.
- Replace both instances of `TalendJobSoapBindingStub` with the routine name ending with **BindingStub**, such as: `XigniteFundHoldingsSoap_BindingStub`.
- Within the brackets corresponding to the pieces of code: stub.setUsername and stub.setPassword, enter your username and password respectively, between quotes.

  For the sake of confidentiality or maintenance, you can store your username and password in context variables.
- The list of funds provided by the *Xignite* Web service is identified using so-called "*symbols*", which are of string type. In this example, we intend to fetch the list of funds of which the symbol is between "I" and "J". To do so, define the following statements: `string startSymbol="I"` and `string endSymbol="J"`.
- Then enter the piece of code to create the result table showing the list of funds (**listFunds**) of funds holdings using the statements defined earlier on: `routines.Fund[] result = fundHoldings.listFunds(startSymbol, endSymbol);`
- Run a loop on the fund list to fetch the funds ranging from "I" to "J": `for(int i = 0; i < result.length; i++) {`.
- Define the results to return, for example: fetch the **CIK** data from the **Security** schema using the code `getSecurity().getCIK()`, then pass them on to the **CIK** output schema.

The function that operates the Web service should read as follows:

```
routines.XigniteFundHoldingsSoap_PortType
fundHoldings = new routines.XigniteFundHoldingsLocator().getXigniteFundHoldingsSoap(
);

routines.XigniteFundHoldingsSoap_BindingStub
stub = (routines.XigniteFundHoldingsSoap_BindingStub)fundHoldings;

stub.setUsername("identifiant");
Stub.setPassword("mot de passe");

String startSymbol="I";
String endSymbol="J";

routines.Fund[ ] result = fundHoldings.listFunds(startSymbol,
endSymbol); for(int i = 0; i < result.length; i++) {

output_row.CIK = (result[i]).getSecurity().getCIK();
output_row.cusip = (result[i]).getSecurity().getCusip();
output_row.symbol = (result[i]).getSecurity().getSymbol();
output_row.ISIN = (result[i]).getSecurity().getISIN();
output_row.valoren = (result[i]).getSecurity().getValoren();
output_row.name = (result[i]).getSecurity().getName();
output_row.market = (result[i]).getSecurity().getMarket();
output_row.category =
(result[i]).getSecurity().getCategoryOrIndustry();
output_row.asOfDate = (result[i]).getAsOfDate();
```

The outputs defined in the Java function `output_row.output` must match the columns defined in the component schema exactly. The case used must also be matched in order for the data to be retrieved.

- In the **Match Brackets** field, select the number of brackets to use to end the For loop, based on the number of open brackets. For this scenario, select one bracket only as only one bracket has been opened in the function.
- Double-click the **tLogRow** component to display the **Component** view and set its parameters.
- Click the **[...]** button next to the **Edit Schema** field in order to check that the preceding component schema was properly propagated to the output component. If needed, click the **Sync Columns** button to retrieve the schema.
- Save your Job and press **F6** to run it.

```
Starting job tWebServiceInput_scenario2 at 13:59 27/07/2009.

[statistics] connecting to socket on port 3922
[statistics] connected

|893957753|IAAAX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
|893957746|IAABX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
|893957720|IAALX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
0000895430|44980Q724|IACAX|||ING International Capital Appre
|FUNDS|Foreign Large Growth|7/27/2009
N/A|893958454|IACBX|||Ta Idex Salomon All Cap Class B |FUNDS|Large
Blend|7/27/2009
N/A|00141T122|IACFX|||Aim China Institutional Class
|FUNDS|Pacific/Asia ex-Japan Stk|7/27/2009
```

The funds comprised between “I” and “J” are returned and displayed in the *Talend Open Studio for Big Data* console.

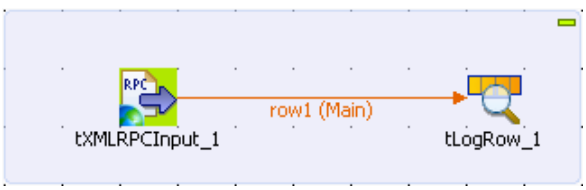
tXMLRPCInput

tXMLRPCInput Properties

Component family	Internet	
Function	Calls the defined method from the invoked RPC service, and returns the class as defined, based on the given parameters.	
Purpose	Invokes a Method through a Web service and for the described purpose	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job. In the RPC context, the schema corresponds to the output parameters. If two parameters are meant to be returned, then the schema should contain two columns.
	Server URL	URL of the RPC service to be accessed
	Need authentication / *Username and Password*	Select this check box and fill in a username and password if required to access the service.
	Method Name	Enter the exact name of the Method to be invoked. The Method name MUST match the corresponding method described in the RPC Service. The Method name is also case-sensitive.
	Return class	Select the type of data to be returned by the method. Make sure it fully matches the one defined in the method.
	Parameters	Enter the parameters expected by the method as input parameters.
Usage	This component is generally used as a Start component. It requires to be linked to an output component.	
Limitation	n/a	

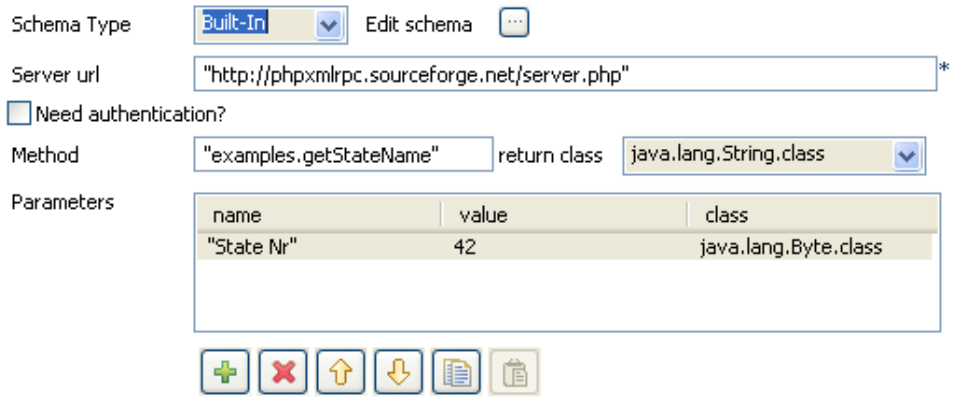
Scenario: Guessing the State name from an XMLRPC

This scenario describes a two-component Job aiming at using a RPC method and displaying the output on the console view.

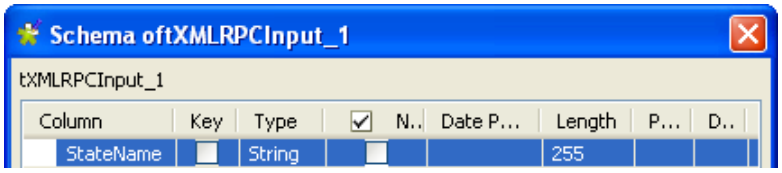


- Drop the **tXMLRPCInput** and a **tLogRow** components from the **Palette** to the design workspac.

- Set the **tXMLRPCInput** basic settings.



- Define the **Schema type** as **Built-in** for this use case.
- Set a single-column schema as the expected output for the called method is only one parameter: *StateName*.



- Then set the **Server url**. For this demo, use: *http://phpxmlrpc.sourceforge.net/server.php*
- No authentication details are required in this use case.
- The **Method** to be called is: *examples.getStateName*
- The **return class** is not compulsory for this method but might be strictly required for another. Leave the default setting for this use case.
- Then set the input **Parameters** required by the method called. The **Name** field is not used in the code but the value should follow the syntax expected by the method. In this example, the Name used is *State Nr* and the value randomly chosen is *42*.
- The class has not much impact using this demo method but could have with another method, so leave the default setting.
- On the **tLogRow** component **Component** view, check the box: **Print schema column name in front of each value**.
- Then save the Job and press **F6** to execute it.

```
Starting job xmlrpc at 16:25 21/09/2007.
StateName: South Dakota
Job xmlrpc ended at 16:25 21/09/2007. [exit code=0]
```

South Dakota is the state name found using the `GetStateName` RPC method and corresponds the 42nd State of the United States as defined as input parameter.

Logs & Errors components

This chapter details the main components that you can find in the **Logs & Errors** family of the the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Logs & Errors family groups together the components which are dedicated to log information catching and Job error handling.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAssert

tAssert Properties

The **tAssert** component works alongside **tAssertCatcher** to evaluate the status of a Job execution. It concludes with the boolean result based on an assertive statement related to the execution and feed the result to **tAssertCatcher** for proper Job status presentation.

Component family	Logs & Errors	
Function	Provides the Job status messages to **tAssertCatcher**.	
Purpose	Generates the boolean evaluation on the concern for the Job execution status. The status includes: - **Ok**: the Job execution succeeds. - **Fail**: the Job execution fails. The tested Job's result does not match the expectation or an execution error occured at runtime.	
Basic settings	*Description*	Type in your descriptive message to help identify the assertion of a **tAssert**.
	Expression	Type in the assertive statement you base the evaluation on.
Usage	This component follows the action the assertive condition is directly related to. It can be the intermediate or end component of the main Job, or the start, intermediate or end component of the secondary Job.	
Limitation	The evaluation of **tAssert** is captured only by **tAssertCatcher**.	

Scenario: Setting up the assertive condition for a Job execution

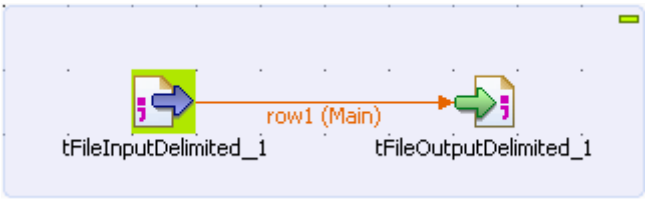
This scenario describes how to set up an assertive condition in **tAssert** in order to evaluate that a Job execution succeeds or not. Moreover, you can also find out how the two different evaluation results display and the way to read them. Apart from **tAssert**, the scenario uses the following components as well:

- **tFileInputDelimited** and **tFileOutputDelimited**. The two components compose the main Job of which the execution status is evaluated. For the detailed information on the two components, see section *tFileInputDelimited* and section *tFileOutputDelimited*.
- **tFileCompare**. It realizes the comparison between the output file of the main Job and a standard reference file. The comparative result is evaluated by **tAssert** against the assertive condition set up in its settings. For more detailed information on **tFileCompare**, see section *tFileCompare*.
- **tAssertCatcher**. It captures the evaluation generated by **tAssert**. For more information on **tAssertCatcher**, see section *tAssertCatcher*.
- **tLogRow**. It allows you to read the captured evaluation. For more information on **tLogRow**, see section *tLogRow*.

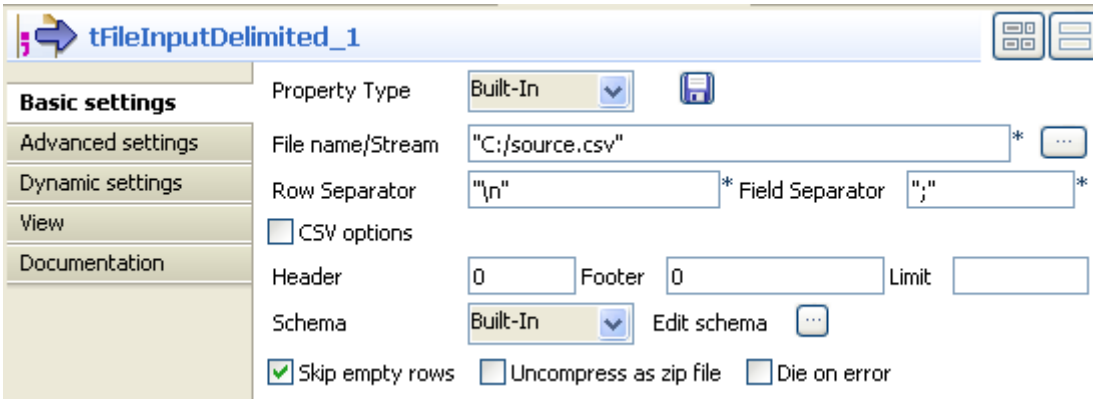
First proceed as follows to design the main Job:

- Prepare a delimited *.csv* file as the source file read by your main Job.

- Edit two rows in the delimited file. The contents you edit are not important, so feel free to simplify them.
- Name it *source.csv*.
- In *Talend Open Studio for Big Data*, create a new job *JobAssertion*.
- Place **tFileInputDelimited** and **tFileOutputDelimited** on the workspace.
- Connect them with a **Row Main** link to create the main Job.



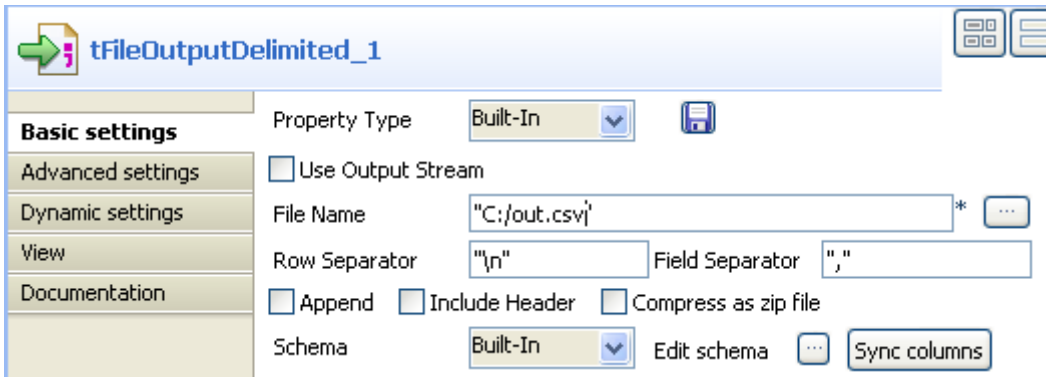
- Double-click **tFileInputDelimited** to open its **Component** view.
- In the **File Name** field of the **Component** view, fill in the path or browse to *source.csv*.



- Still in the **Component** view, set **Property Type** to **Built-In** and click … next to **Edit schema** to define the data to pass on to **tFileOutputDelimited**. In the scenario, define the data presented in *source.csv* you created.

For more information about schema types, see *Talend Open Studio for Big Data User Guide*.

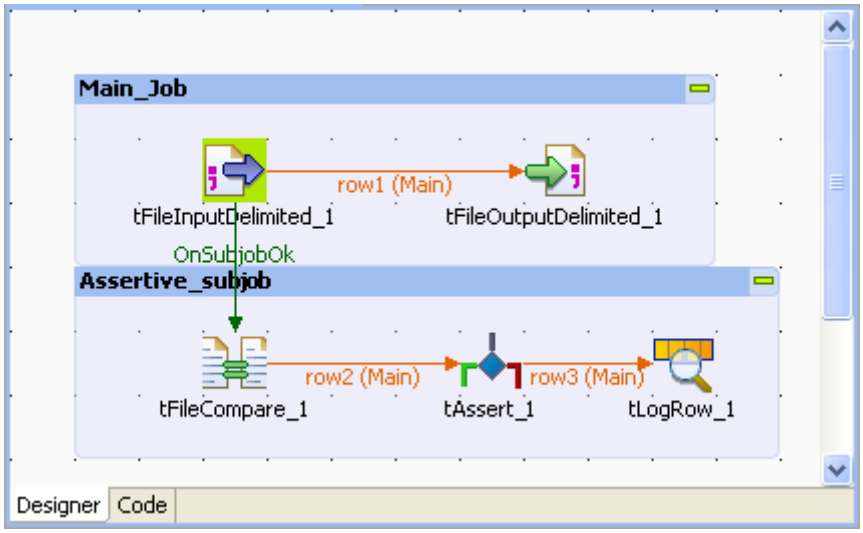
- Define the other parameters in the corresponding fields according to *source.csv* you created.
- Double-click **tFileOutputDelimited** to open its **Component** view.
- In the **File Name** field of the **Component** view, fill in or browse to specify the path to the output file, leaving the other fields as they are by default.



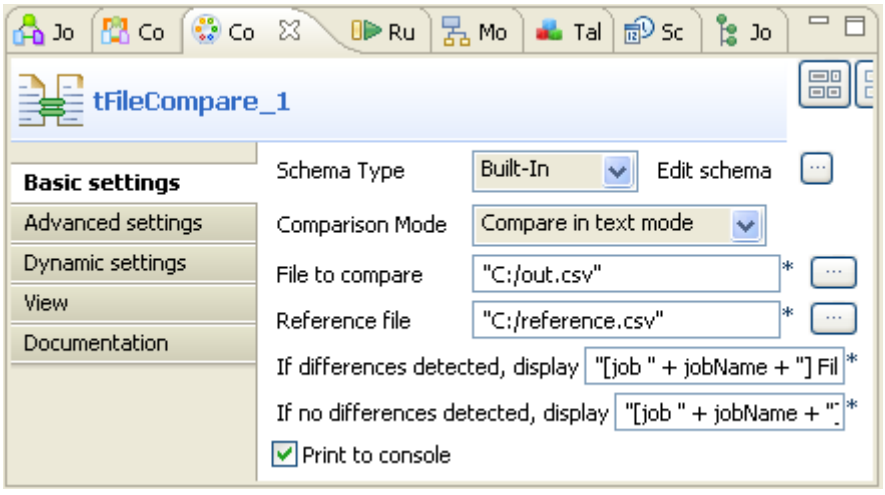
- Press **F6** to execute the main Job. It reads *source.csv*, pass the data to **tFileOutputDelimited** and output an delimited file, *out.csv*.

Then contine to edit the Job to see how **tAssert** evaluates the execution status of the main Job.

- Rename *out.csv* as *reference.csv*.This file is used as the expected result the main Job should output.
- Place **tFileCompare**, **tAssert** and **tLogRow** on the workspace.
- Connect them with **Row Main** link.
- Connect **tFileInputDelimited** to **tFileCompare** with **OnSubjobOk** link.

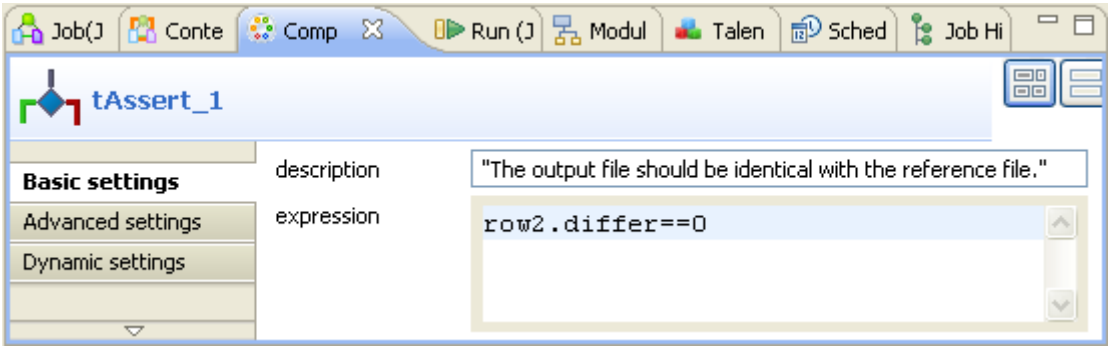


- Double-click **tFileCompare** to open its **Component** view.
- In the **Component** view, fill in the corresponding file paths in the **File to compare** field and the **Reference file** field, leaving the other fields as default.



For more information on the **tFileCompare** component, see section *tFileCompare*.

- Then click **tAssert** and click the **Component** tab on the lower side of the workspace.



- In the **Component** view, edit the assertion `row2.differ==0` in the **expression** field and the descriptive message of the assertion in **description** field.

In the **expression** field, `row2` is the data flow transmissing from **tFileCompare** to **tAssert**, `differ` is one of the columns of the **tFileCompare** schema and presents whether the compared files are identical, and `0` means no difference is detected between the *out.csv* and *reference.csv* by **tFileCompare**. Hence when the compared files are identical, the assertive condition is thus fulfilled, **tAssert** concludes that the main Job succeeds; otherwise, it concludes failure.

The *differ* column is in the read-only **tFileCompare** schema. For more information on its schema, see section *tFileCompare*.

- Press **F6** to execute the Job.

- Check the result presented in the **Run** view

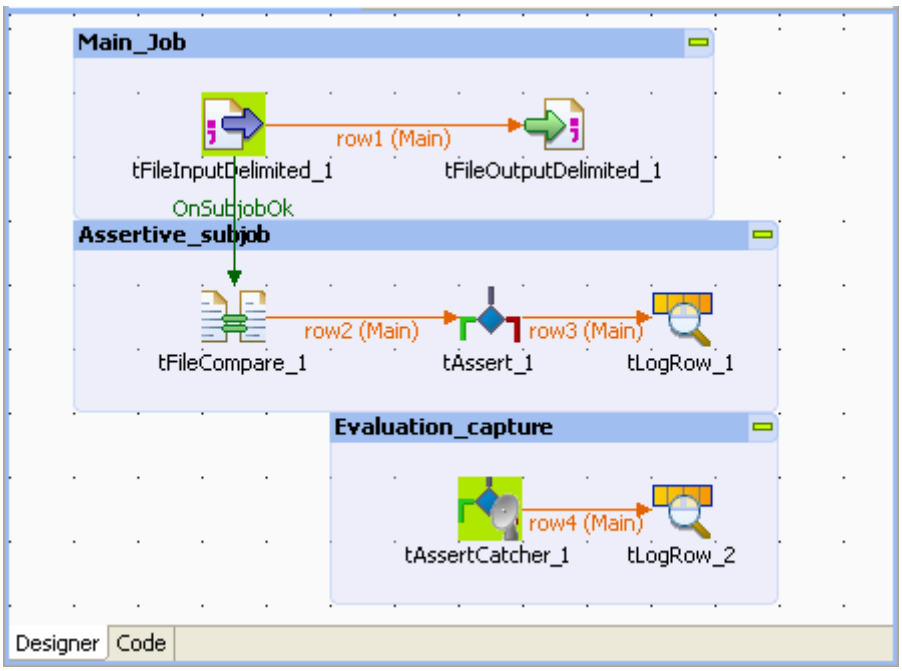
```
Starting job JobAssertion at 14:00 29/01/2010.

[statistics] connecting to socket on port 3788
[statistics] connected
[job JobAssertion] Files are identical
.----------+----------------+-------------------+------------+--------------+------+---------
-----------------------------.
|                                                          tLogRow_1
|
|=---------+----------------+-------------------+------------+--------------+------+---------
-----------------------------=|
|file      |file_ref        |moment             |job         |component     |differ|message
|
|=---------+----------------+-------------------+------------+--------------+------+---------
-----------------------------=|
|C:/out.csv|C:/reference.csv|2010-01-29 14:00:17|JobAssertion|tFileCompare_1|0     |[job
JobAssertion] Files are identical|
'----------+----------------+-------------------+------------+--------------+------+---------
-----------------------------'
[statistics] disconnected
Job JobAssertion ended at 14:00 29/01/2010. [exit code=0]
```

The console shows the comparison result of **tFileCompare**: `Files are identical`. But you find nowhere the evaluation result of **tAssert**.

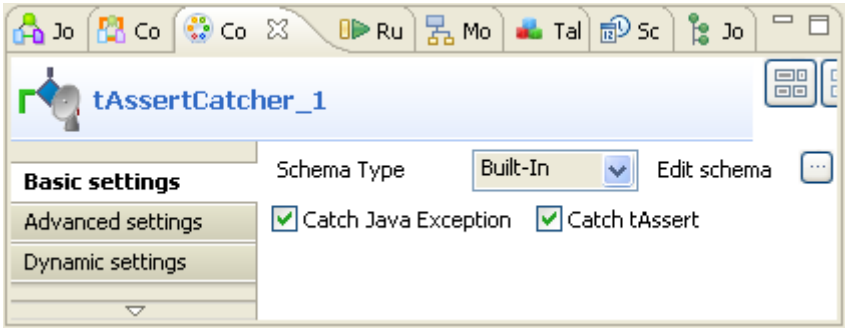
So you need **tAssertCatcher** to capture the evaluation.

- Place **tAssertCatcher** and **tLogRow** on the workspace.

- Connect them with **Row Main** link.



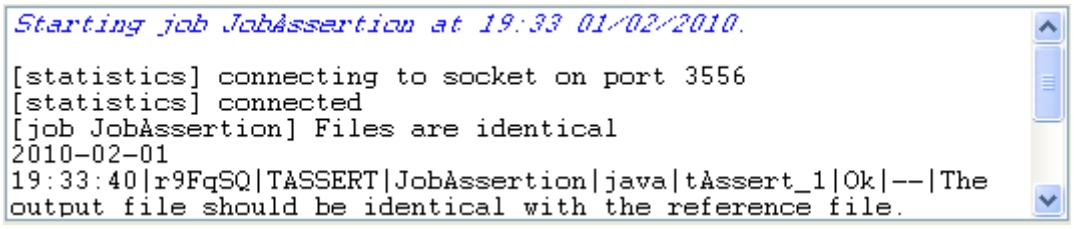
- Use the default configuration in the **Component** view of **tAssertCatcher**.



- Press **F6** to execute the Job.
- Check the result presented in the **Run** view. You will see the Job status information is added in:

```
2010-01-29 15:37:33|fAvAzH|TASSERT|JobAssertion|java|tAssert_1|Ok|--|
The output file should be identical with the reference file
```

.



The descriptive information on *JobAssertion* in the console is organized according to the **tAssertCatcher** schema. This schema includes, in the following order, the execution time, the process ID, the project name, the Job name, the code language, the evaluation origin, the evaluation result, detailed information of the evaluation, descriptive message of the assertion. For more information on the schema of **tAssertCatcher**, see section *tAssertCatcher*.

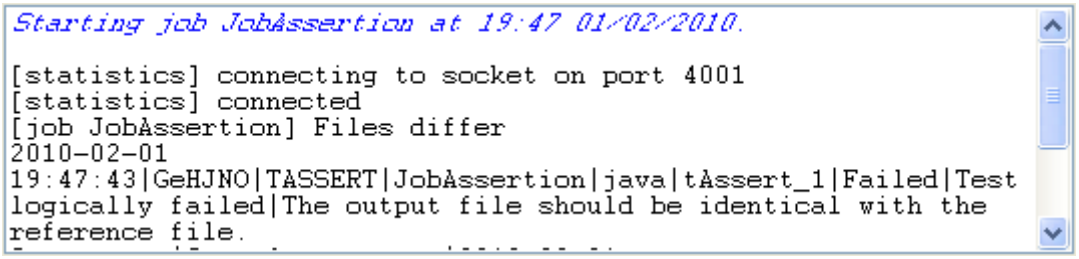
The console indicates that the execution status of Job *JobAssertion* is *Ok*. In addition to the evalution, you can still see other descriptive information about *JobAssertion* including the descriptive message you have edited in the **Basic settings** of **tAssert**.

Then you will perform operations to make the main Job fail to generate the expected file. To do so, proceed as follows in the same Job you have executed:

- Delete a row in *reference.csv*.
- Press **F6** to execute the Job again.
- Check the result presented in **Run** view.

```
2010-02-01 19:47:43|GeHJNO|TASSERT|JobAssertion|tAssert_1|Failed|Test
logically failed|The output file should be identical with the reference
file
```

.



The console shows that the execution status of the main Job is *Failed*. The detailed explanation for this status is closely behind it, reading `Test logically failed`.

You can thus get a basic idea about your present Job status: it fails to generate the expected file because of a logical failure. This logical failure could come from a logical mistake during the Job design.

The status and its explanatory information are presented respectively in the *status* and the *substatus* columns of the **tAssertCatcher** schema. For more information on the columns, see section *tAssertCatcher*.

tAssertCatcher

tAssertCatcher Properties

Component family	Logs & Errors	
Function	Based on its pre-defined schema, fetches the execution status information from repository, Job execution and **tAssert**.	
Purpose	Generates a data flow consolidating the status information of a job execution and transfer the data into defined output files.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the fields to be processed and passed on to the next component. In this particular case, the schema is read-only, as this component gathers standard log information including:
		Moment: Processing time and date.
		Pid: Process ID.
		Project: Project which the job belongs to.
		Job: Job name.
		Language: Language used by the Job (Java)
		Origin: Status evaluation origin. The origin may be different **tAssert** components.
		Status: Evaluation fetched from **tAssert**. They may be - **Ok**: if the assertive statement of **tAssert** is evaluated as true at runtime. - **Failed**: if the assertive statement of **tAssert** is evaluated as false or an execution error occurs at runtime. The tested Job's result does not match the expectation or an execution error occured at runtime.
		Substatus: Detailed explanation for failed execution. The explanation can be: - **Test logically failed**: the investigated Job does not produce the expected result. - **Execution error**: an execution error occured at runtime.
		Description: Descriptive message you typed in in **Basic settings** of **tAssert**.
	Catch Java Exception	This check box allows to capture Java exception errors, once checked.
	Catch tAssert	This check box allows to capture the evaluations of **tAssert**.
Usage	This component is the start component of a secondary Job which fetches the execution status information from several sources. It generates a data flow to transfer the information to the component which proceeds.	
Limitation	This component must be used with **tAssert** together.	

Related scenarios

For using case in relation with **tAssertCatcher**, see **tAssert** scenario:

- section *Scenario: Setting up the assertive condition for a Job execution*

tChronometerStart

tChronometerStart Properties

Component family	Logs & Errors	
Function	Starts measuring the time a subjob takes to be executed.	
Purpose	Operates as a chronometer device that starts calculating the processing time of one or more subjobs in the main Job, or that starts calculating the processing time of part of your subjob.	
Usage	You can use **tChronometerStart** as a start or middle component. It can precede one or more processing tasks in the subjob. It can precede one or more subjobs in the main Job.	
Limitation	n/a	

Related scenario

For related scenario, see section *Scenario: Measuring the processing time of a subjob and part of a subjob*.

tChronometerStop

tChronometerStop Properties

Component family	Logs & Errors	
Function	Measures the time a subjob takes to be executed.	
Purpose	Operates as a chronometer device that stops calculating the processing time of one or more subjobs in the main Job, or that stops calculating the processing time of part of your subjob. **tChronometerStop** displays the total execution time.	
Basic settings	*Since options*	Select either check box to select measurement starting point: **Since the beginning**: stops time measurement launched at the beginning of a subjob. **Since a tChronometerStart**: stops time measurement launched at one of the **tChronometerStart** components used on the data flow of the subjob.
	Display duration in console	When selected, it displays subjob execution information on the console.
	Display component name	When selected, it displays the name of the component on the console.
	Caption	Enter desired text, to identify your subjob for example.
	Display human readable duration	When selected, it displays subjob execution information in readable time unites.
Usage	Cannot be used as a start component.	
Limitation	n/a	

Scenario: Measuring the processing time of a subjob and part of a subjob

This scenario is a subjob that does the following in a sequence:

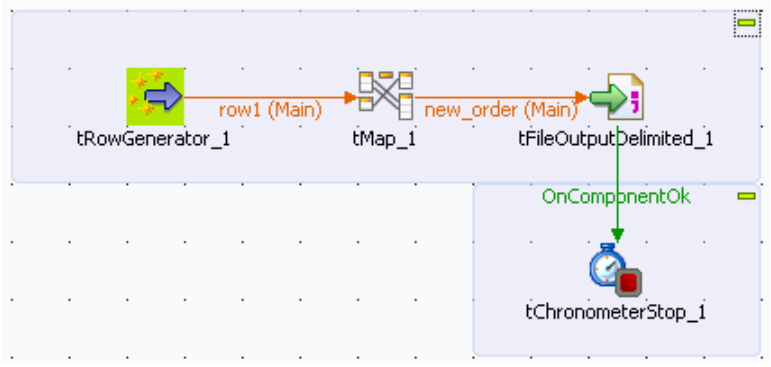
- generates 1000 000 rows of first and last names,
- gathers first names with their corresponding last names,
- stores the output data in a delimited file,
- measures the duration of the subjob as a whole,
- measures the duration of the name replacement operation,
- displays the gathered information about the processing time on the **Run** log console.

To measure the processing time of the subjob:

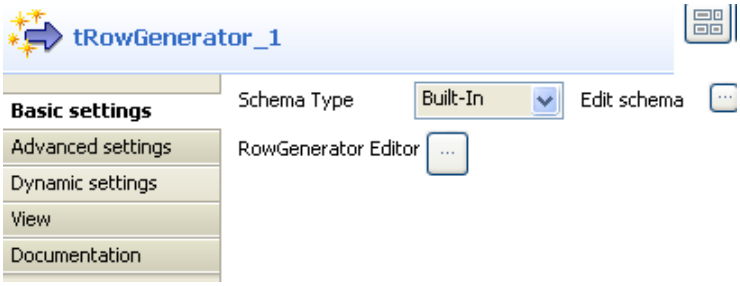
- Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tMap**, **tFileOutputDelimited**, and **tChronometerStop**.

- Connect the first three components using **Main Row** links.

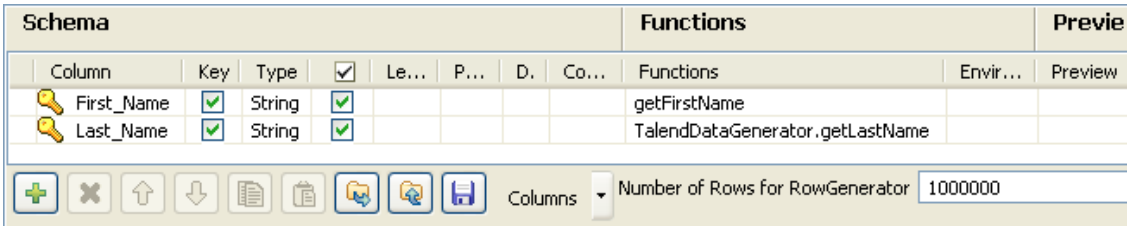
When connecting **tMap** to **tFileOutputDelimited**, you will be prompted to name the output table. The name used in this example is “new_order”.



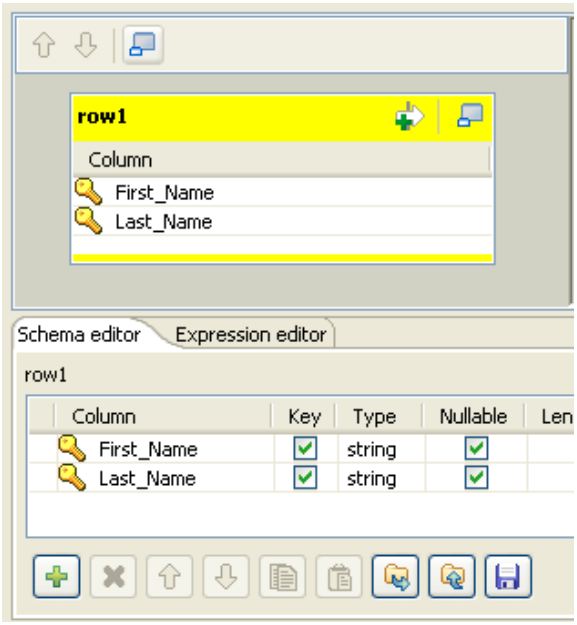
- Connect **tFileOutputDelimited** to **tChronometerStop** using an **OnComponentOk** link.
- Select **tRowGenerator** and click the **Component** tab to display the component view.
- In the **component** view, click **Basic settings**. The **Component** tab opens on the **Basic settings** view by default.



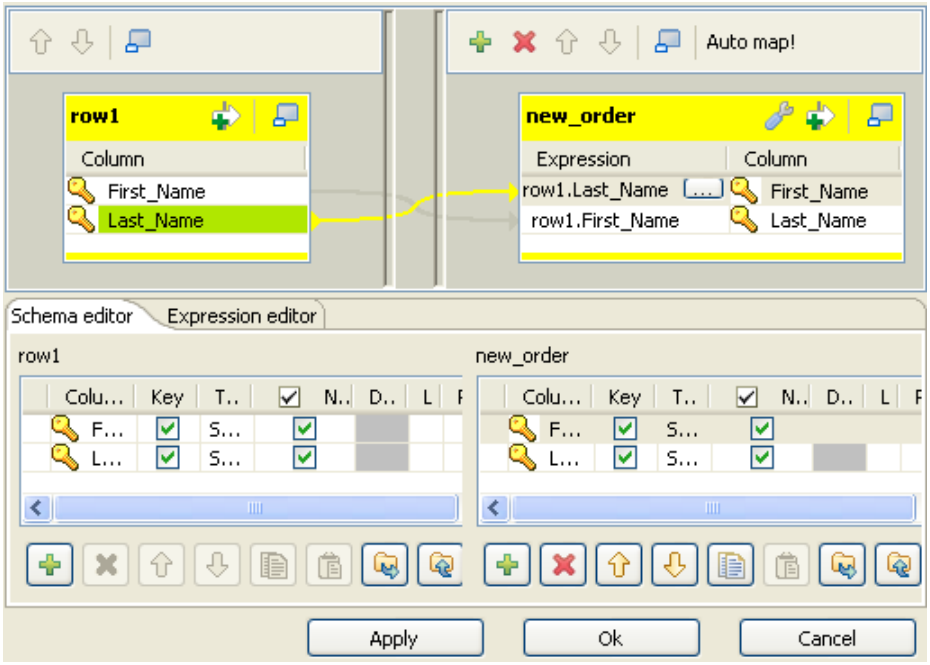
- Click **Edit schema** to define the schema of the **tRowGenerator**. For this Job, the schema is composed of two columns: *First_Name* and *Last_Name*, so click twice the **[+]** button to add two columns and rename them.
- Click the **RowGenerator Editor** three-dot button to open the editor and define the data to be generated.



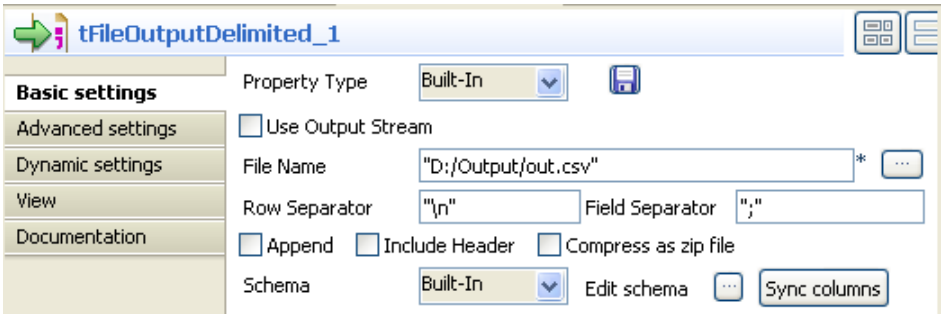
- In the **RowGenerator Editor**, specify the number of rows to be generated in the **Number of Rows for RowGenerator** field and click **OK**. The **RowGenerator Editor** closes.
- You will be prompted to propagate changes. Click **Yes** in the popup message.
- Double-click on the **tMap** component to open the Map editor. The Map editor opens displaying the input metadata of the **tRowGenerator** component.



- In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and define them.
- In the Map editor, drag the *First_Name* row from the input table to the *Last_Name* row in the output table and drag the *Last_Name* row from the input table to the *First_Name* row in the output table.
- Click **Apply** to save changes.
- You will be prompted to propagate changes. Click **Yes** in the popup message.
- Click **OK** to close the editor.

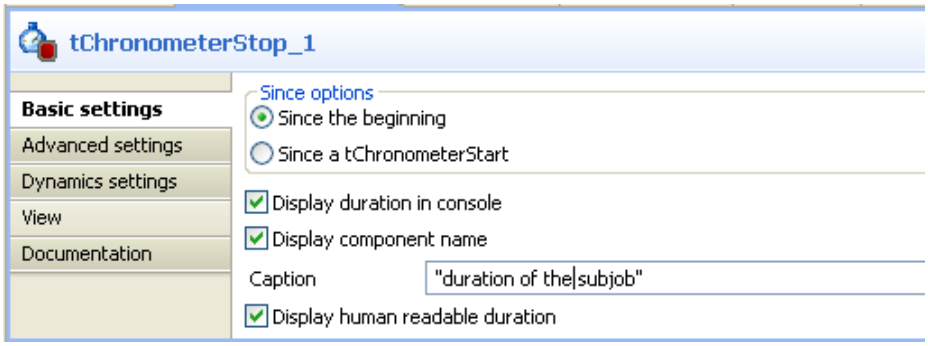


- Select **tFileOutputDelimited** and click the **Component** tab to display the component view.
- In the **Basic settings** view, set **tFileOutputDelimited** properties as needed.



- Select **tChronometerStop** and click the **Component** tab to display the component view.
- In the **Since options** panel of the **Basic settings** view**,** select **Since the beginning** option to measure the duration of the subjob as a whole.

t



- Select/clear the other check boxes as needed. In this scenario, we want to display the subjob duration on the console preceded by the component name.
- If needed, enter a text in the **Caption** field.
- Save your Job and press **F6** to execute it.

```
Starting job tChronometerStop at 11:40 05/03/2010.

[statistics] connecting to socket on port 3399
[statistics] connected
[ tChronometerStop_1 ]     2seconds   duration of the subjob
2968 milliseconds
[statistics] disconnected
Job tChronometerStop ended at 11:40 05/03/2010. [exit code=0]
```

You can measure the duration of the subjob the same way by placing **tChronometerStop** below **tRowGenerator**, and connecting the latter to **tChronometerStop** using an **OnSubjobOk** link.

tDie

tDie properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a tLogCatcher in order for the log data collected to be encapsulated and passed on to the output defined.

Component family	Logs & Errors	
Function	Kills the current Job. Generally used with a **tCatch** for log purpose.	
Purpose	Triggers the **tLogCatcher** component for exhaustive log before killing the Job.	
Basic settings	*Die message*	Enter the message to be displayed before the Job is killed.
	Error code	Enter the error code if need be, as an integer
	Priority	Set the level of priority, as an integer
Usage	Cannot be used as a start component.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tDie**, see **tLogCatcher** scenarios:

- section *Scenario 1: warning & log on entries*
- section *Scenario 2: Log & kill a Job*

tFlowMeter

tFlowMeter Properties

Component family	Logs & Errors	
Function	Counts the number of rows processed in the defined flow.	
Purpose	The number of rows is then meant to be caught by the **tFlowMeterCatcher** for logging purpose.	
Basic settings	*Use input connection name as label*	Select this check box to reuse the name given to the input main row flow as label in the logged data.
	Mode	Select the type of values for the data measured: **Absolute**: the actual number of rows is logged **Relative**: a ratio (%) of the number of rows is logged. When this option is selected, a **Connections List** shows to let you select a reference connection.
	Thresholds	Adds a threshold to watch proportions in volumes measured. you can decide that the normal flow has to be between low and top end of a row number range, and if the flow is under this low end, there is a bottleneck.
Usage	Cannot be used as a start component as it requires an input flow to operate.	
Limitation	n/a	

If you have a need of log, statistics and other measurement of your data flows, see *Talend Open Studio for Big Data* **User Guide**.

Related scenario

For related scenario, see section *Scenario: Catching flow metrics from a Job*

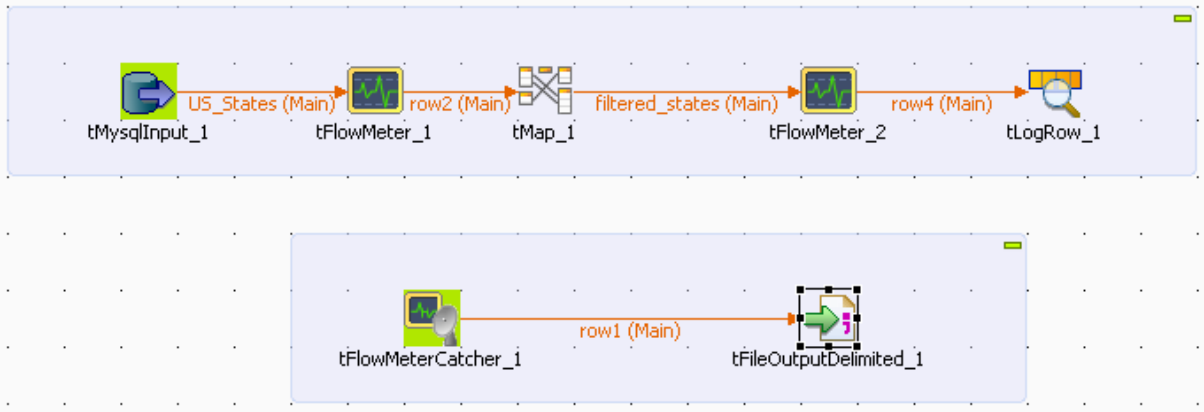
tFlowMeterCatcher

tFlowMeterCatcher Properties

Component family	Logs & Errors	
Function	Based on a defined sch.ema, the **tFlowMeterCatcher** catches the processing volumetric from the **tFlowMeter** component and passes them on to the output component.	
Purpose	Operates as a log function triggered by the use of a **tFlowMeter** component in the Job.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the fields to be processed and passed on to the next component. In this particular case, the schema is read-only, as this component gathers standard log information including:
		Moment: Processing time and date
		Pid: Process ID
		Father_pid: Process ID of the father Job if applicable. If not applicable, Pid is duplicated.
		Root_pid: Process ID of the root Job if applicable. If not applicable, pid of current Job is duplicated.
		System_pid: Process id generated by the system
		Project: Project name, the Job belongs to.
		Job: Name of the current Job
		Job_repository_id: ID generated by the application.
		Job_version: Version number of the current Job
		Context: Name of the current context
		Origin: Name of the component if any
		Label: Label of the row connection preceding the **tFlowMeter** component in the Job, and that will be analyzed for volumetrics.
		Count: Actual number of rows being processed
		Reference: Number of rows passing the reference link.
		Thresholds: Only used when the relative mode is selected in the **tFlowMeter** component.
Usage	This component is the start component of a secondary Job which triggers automatically at the end of the main Job.	
Limitation	The use of this component cannot be separated from the use of the **tFlowMeter**. For more information, see section *tFlowMeter*	

Scenario: Catching flow metrics from a Job

The following basic Job aims at catching the number of rows being passed in the flow processed. The measures are taken twice, once after the input component, that is, before the filtering step and once right after the filtering step, that is, before the output component.

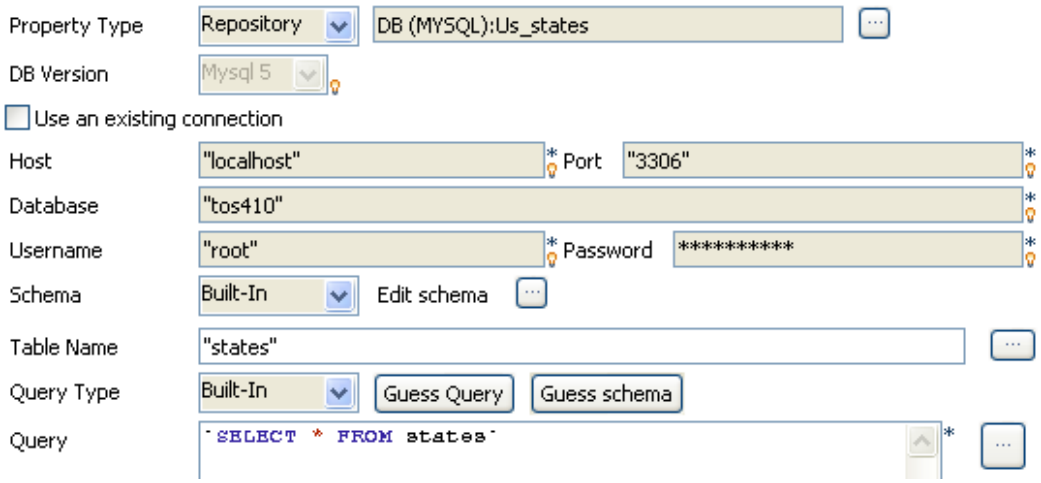


- Drop the following components from the **Palette** to the design workspace: **tMysqlInput**, **tFlowMeter** (x2), **tMap**, **tLogRow**, **tFlowMeterCatcher** and **tFileOutputDelimited**.

- Link components using row main connections and click on the label to give consistent name throughout the Job, such as *US_States* from the input component and *filtered_states* for the output from the **tMap** component, for example.

- Link the **tFlowMeterCatcher** to the **tFileOutputDelimited** component using a row main link also as data is passed.

- On the **tMysqlInput** Component view, configure the connection properties as **Repository**, if the table metadata are stored in the Repository. Or else, set the Type as **Built-in** and configure manually the connection and schema details if they are built-in for this Job.

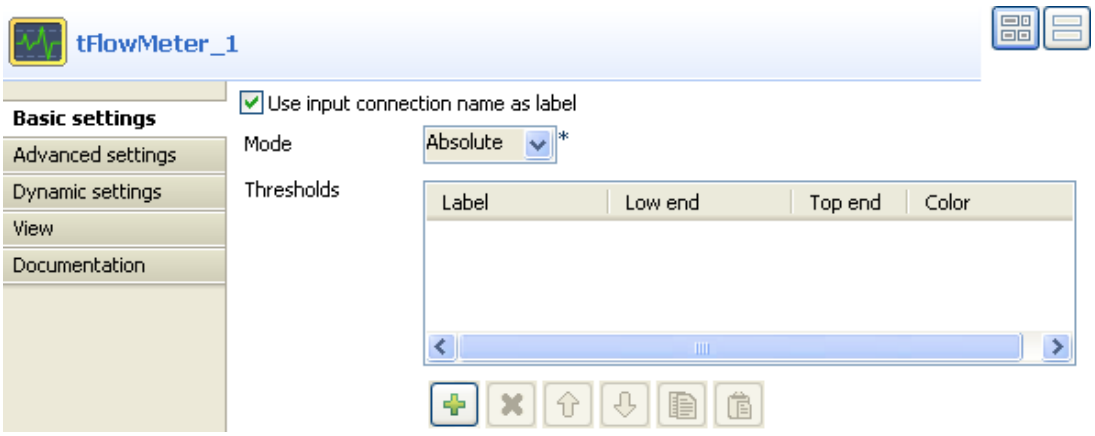


- The 50 States of the USA are recorded in the table *states*. In order for all 50 entries of the table to get selected, the query to run onto the Mysql database is as follows:

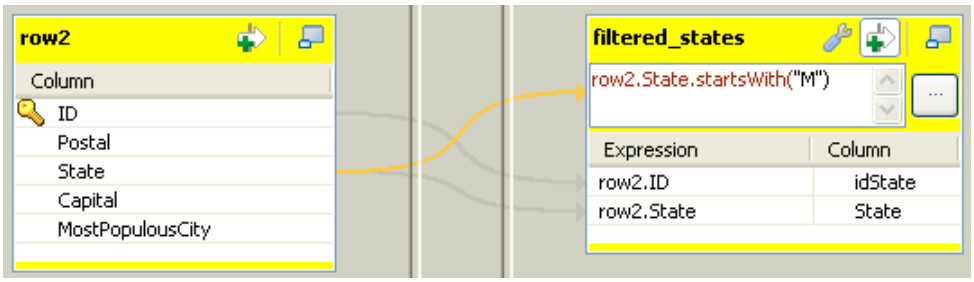
```
select * from states.
```

- Select the relevant **encoding type** on the Advanced settings vertical tab.

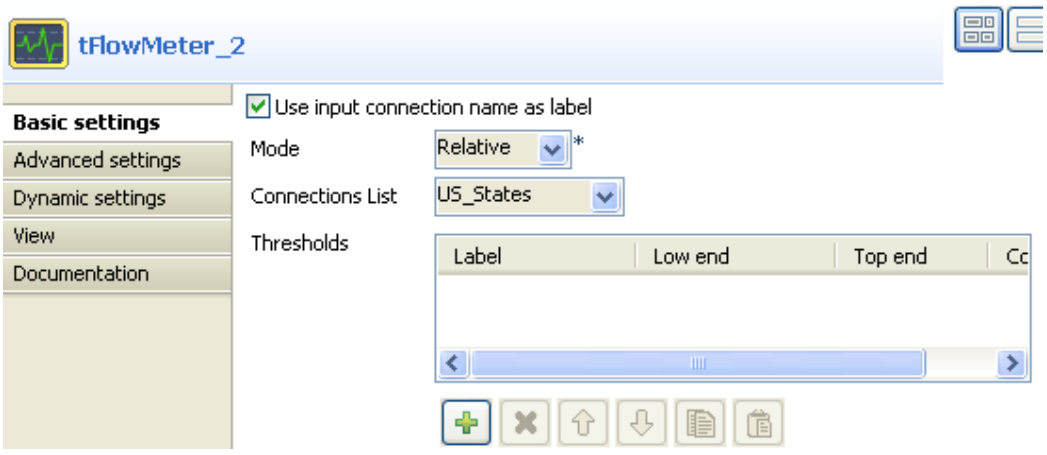
- Then select the following component which is a **tFlowMeter** and set its properties.



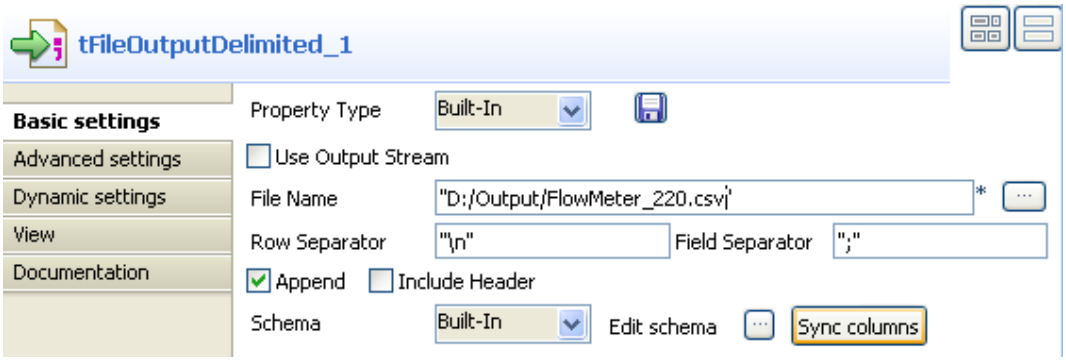
- Select the check box **Use input connection name as label**, in order to reuse the label you chose in the log output file (**tFileOutputDelimited**).
- The mode is **Absolute** as there is no reference flow to meter against, also no **Threshold** is to be set for this example.
- Then launch the **tMap** editor to set the filtering properties.
- For this use case, drag and drop the ID and State columns from the Input area of the **tMap** towards the Output area. No variable is used in this example.



- On the Output flow area (labelled *filtered_states* in this example), click the arrow & plus button to activate the expression filter field.
- Drag the *State* column from the Input area (*row2*) towards the expression filter field and type in the rest of the expression in order to filter the state labels starting with the letter *M*. The final expression looks like: `row2.State.startsWith("M")`
- Click **OK** to validate the setting.
- Then select the second **tFlowMeter** component and set its properties.



- Select the check box **Use input connection name as label**.
- Select **Relative** as **Mode** and in the **Reference connections** list, select *US_States* as reference to be measured against.
- Once again, no threshold is used for this use case.
- No particular setting is required in the **tLogRow**.
- Neither does the **tFlowMeterCatcher** as this component's properties are limited to a preset schema which includes typical log information.
- So eventually set the log output component (**tFileOutputDelimited**).



- Select the **Append** check box in order to log all **tFlowMeter** measures.
- Then save your Job and press **F6** to execute it.

```
Starting job FlowMeterCatcher at 17:56 29/08/2007.
19|Maine
20|Maryland
21|Massachusetts
22|Michigan
23|Minnesota
24|Mississippi
25|Missouri
26|Montana
Job FlowMeterCatcher ended at 17:56 29/08/2007. [exit code=0]
```

The **Run** view shows the filtered state labels as defined in the Job.

	A	K	L	M	N	O
1	moment	origin	label	count	reference	thresholds
2	08/29/07 05:56 PM	tFlowMeter_1	US_States	50		
3	08/29/07 05:56 PM	tFlowMeter_2	filtered_states	8	50	

In the delimited csv file, the number of rows shown in column **count** varies between **tFlowMeter1** and **tFlowMeter2** as the filtering has then been carried out. The *reference* column shows also this difference.

tLogCatcher

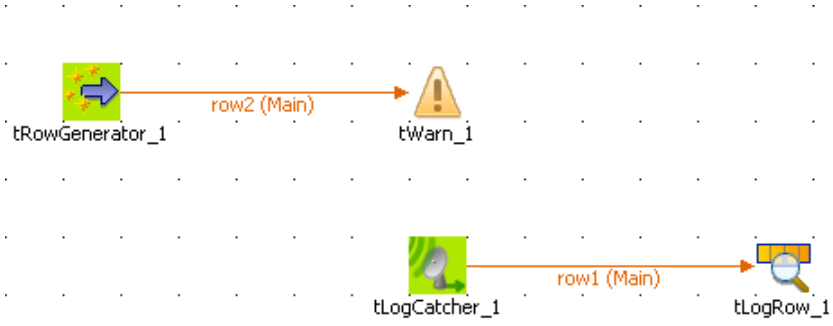
tLogCatcher properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a **tLogCatcher** in order for the log data collected to be encapsulated and passed on to the output defined.

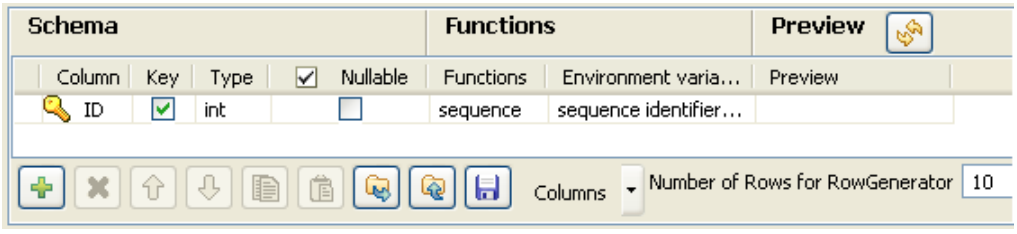
Component family	Logs & Errors	
Function	Fetches set fields and messages from Java Exception, **tDie** and/or **tWarn** and passes them on to the next component.	
Purpose	Operates as a log function triggered by one of the three: Java exception, **tDie** or **tWarn**, to collect and transfer log data.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Catch Java Exception	Select this check box to trigger the **tCatch** function when a Java Exception occurs in the Job
	Catch tDie	Select this check box to trigger the **tCatch** function when a **tDie** is called in a Job
	Catch tWarn	Select this check box to trigger the **tCatch** function when a tWarn is called in a Job
Usage	This component is the start component of a secondary Job which automatically triggers at the end of the main Job	
Limitation	n/a	

Scenario 1: warning & log on entries

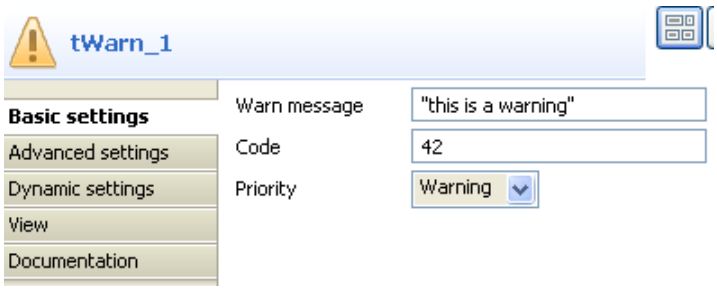
In this basic scenario made of three components, a **tRowGenerator** creates random entries (id to be incremented). The input hits a **tWarn** component which triggers the **tLogCatcher** subjob. This subjob fetches the warning message as well as standard predefined information and passes them on to the **tLogRow** for a quick display of the log data.



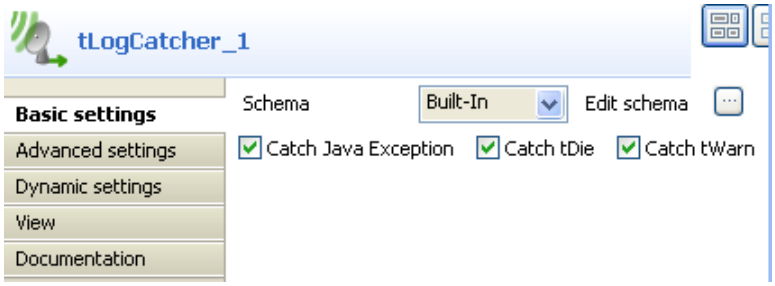
- Drop a **tRowGenerator**, a **tWarn**, a **tLogCatcher** and a **tLogRow** from the **Palette**, on your design workspace
- Connect the **tRowGenerator** to the **tWarn** component.
- Connect separately the **tLogCatcher** to the **tLogRow**.
- On the **tRowGenerator** editor, set the random entries creation using a basic function:



- On the **tWarn Component** view, set your warning message, the code the priority level. In this case, the message is “this is a warning’.
- For this scenario, we will concatenate a function to the message above, in order to collect the first value from the input table.



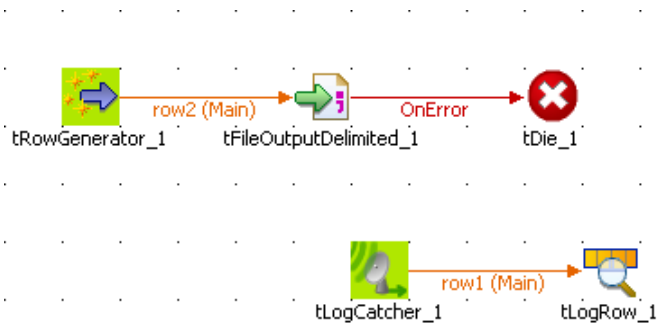
- On the **Basic settings** view of **tLogCatcher**, select the **tWarn** check box in order for the message from the latter to be collected by the subjob.
- Click **Edit Schema** to view the schema used as log output. Notice that the log is comprehensive.



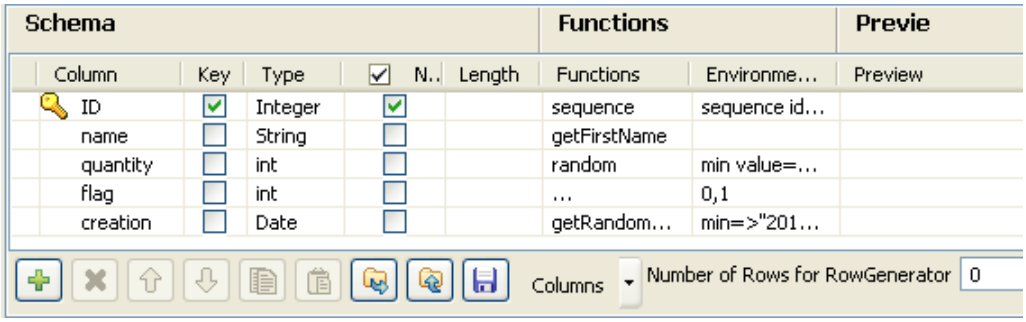
Press **F6** to execute the Job. Notice that the Log produced is exhaustive.

Scenario 2: Log & kill a Job

This scenario uses a **tLogCatcher** and a **tDie** component. A **tRowGenerator** is connected to a **tFileOutputDelimited** using a Row link. On error, the **tDie** triggers the catcher subjob which displays the log data content on the **Run** console.



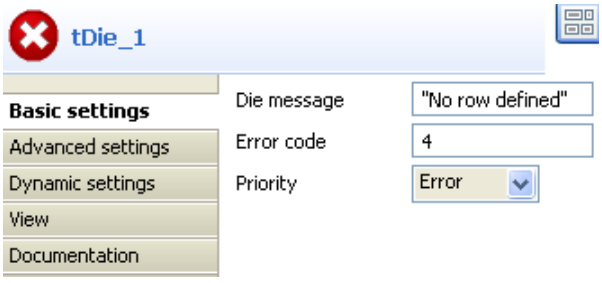
- Drop all required components from various folders of the **Palette** to the design workspace: **tRowGenerator**, **tFileOutputDelimited**, **tDie**, **tLogCatcher**, **tLogRow**.
- On the **tRowGenerator Component** view, define the setting of the input entries to be handled.



- Edit the schema and define the following columns as random input examples: *id*, *name*, *quantity*, *flag and creation*.
- Set the **Number of rows** onto 0. This will constitute the error which the Die operation is based on.
- On the **Values** table, define the functions to feed the input flow.
- Define the **tFileOutputDelimited** to hold the possible output data. The **row** connection from the **tRowGenerator** feeds automatically the output schema. The separator is a simple semi-colon.
- Connect this output component to the **tDie** using a **Trigger > If** connection. Double-click on the newly created connection to define the if:

```
((Integer)globalMap.get("tRowGenerator_1_NB_LINE")) <=0
```

- Then double-click to select and define the **Basic settings** of the **tDie** component.



- Enter your **Die** message to be transmitted to the **tLogCatcher** before the actual kill-job operation happens.
- Next to the Job but not physically connected to it, drop a **tLogCatcher** from the **Palette** to the design workspace and connect it to a **tLogRow** component.

- Define the **tLogCatcher Basic settings**. Make sure the **tDie** box is selected in order to add the Die message to the Log information transmitted to the final component.

```
Starting job tLogCatcher2 at 10:10 20/04/2010.

No row defined
2010-04-20
10:10:57|RPa7uU|RPa7uU|RPa7uU|TDQ_EE_MPX_JAVA|tLogCatcher2|Default|5|tDie|tDie_
1|No row defined|4
Job tLogCatcher2 ended at 10:10 20/04/2010. [exit code=4]
```

- Press **F6** to run the Job and notice that the log contains a black message and a red one.
- The black log data come from the **tDie** and are transmitted by the **tLogCatcher**. In addition the normal Java Exception message in red displays as a Job abnormally died.

tLogRow

tLogRow properties

Component family	Logs & Errors	
Function	Displays data or results in the **Run** console.	
Purpose	**tLogRow** is used to monitor data processed.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You can create the schema and store it locally for this component. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: You have already created and stored the schema in the Repository. You can reuse it in various projects and Job flowcharts. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function is available only when the component is linked with the preceding component using a **Row** connection.
	Basic	Displays the output flow in basic mode.
	Table	Displays the output flow in table cells.
	Vertical	Displays each row of the output flow as a key-value list. With this mode selected, you can choose to show either the unique name or the label of component, or both of them, for each output row.
	Separator (For Basic mode only)	Enter the separator which will delimit data on the Log display.
	Print header (For Basic mode only)	Select this check box to include the header of the input flow in the output display.
	Print component unique name in front of each output row (For Basic mode only)	Select this check box to show the unique name the component in front of each output row to differentiate outputs in case several **tLogRow** components are used.
	Print schema column name in front of each value (For Basic mode only)	Select this check box to retrieve column labels from output schema.
	Use fixed length for values (For Basic mode only)	Select this check box to set a fixed width for the value display.
Usage	This component can be used as intermediate step in a data flow or as a n end object in the Job flowchart.	
Limitation	n/a	

Scenario: Delimited file content display

For related scenarios, see:

- section *Scenario: Reading master data in an MDM hub*.
- section *Scenario: Dynamic context use in MySQL DB insert*.
- section *Scenario 1: warning & log on entries*.
- section *Scenario 2: Log & kill a Job*.

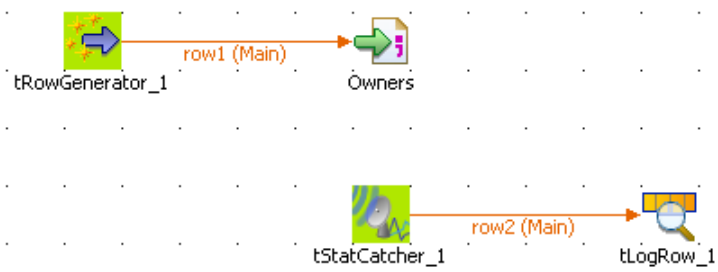
tStatCatcher

tStatCatcher Properties

Component family	Logs & Errors	
Function	Based on a defined schema, gathers the Job processing metadata at a Job level as well as at each component level.	
Purpose	Operates as a log function triggered by the **StatsCatcher Statistics** check box of individual components, and collects and transfers this log data to the output defined.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the fields to be processed and passed on to the next component. In this particular case, the schema is read-only, as this component gathers standard log information including:
		Moment: Processing time and date
		Pid: Process ID
		Father_pid: Process ID of the father Job if applicable. If not applicable, Pid is duplicated.
		Root-pid: Process ID of the root Job if applicable. If not applicable, pid of current Job is duplicated.
		Project: Project name, the Job belongs to.
		Job: Name of the current Job
		Context: Name of the current context
		Origin: Name of the component if any
		Message: Begin or End.
Usage	This component is the start component of a secondary Job which triggers automatically at the end of the main Job. The processing time is also displayed at the end of the log.	
Limitation	n/a	

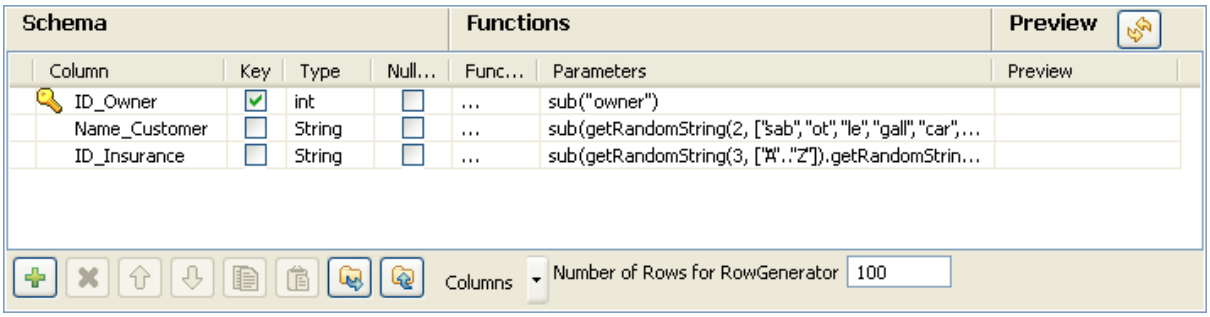
Scenario: Displaying job stats log

This scenario describes a four-component Job, aiming at displaying on the **Run** console the statistics log fetched from the file generation through the tStatCatcher component.



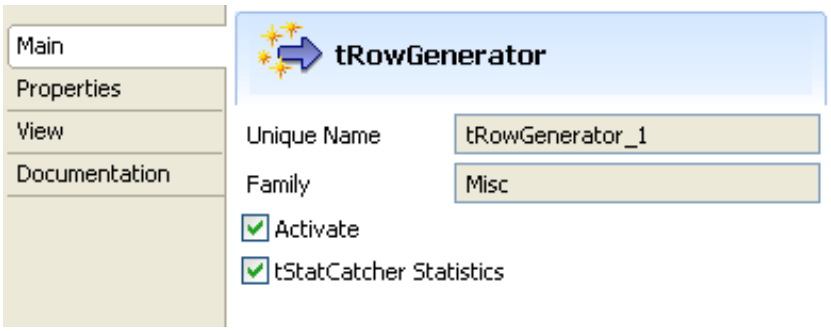
- Drop the required components: **tRowGenerator**, **tFileOutputDelimited**, **tStatCatcher** and **tLogRow** from the **Palette** to the design workspace.

- In the **Basic settings** panel of **tRowGenerator**, define the data to be generated. For this Job, the schema is composed of three columns: *ID_Owners*, *Name_Customer* and *ID_Insurance*.



- The number of rows can be restricted to 100.

- Click on the **Main** tab of the Component view.



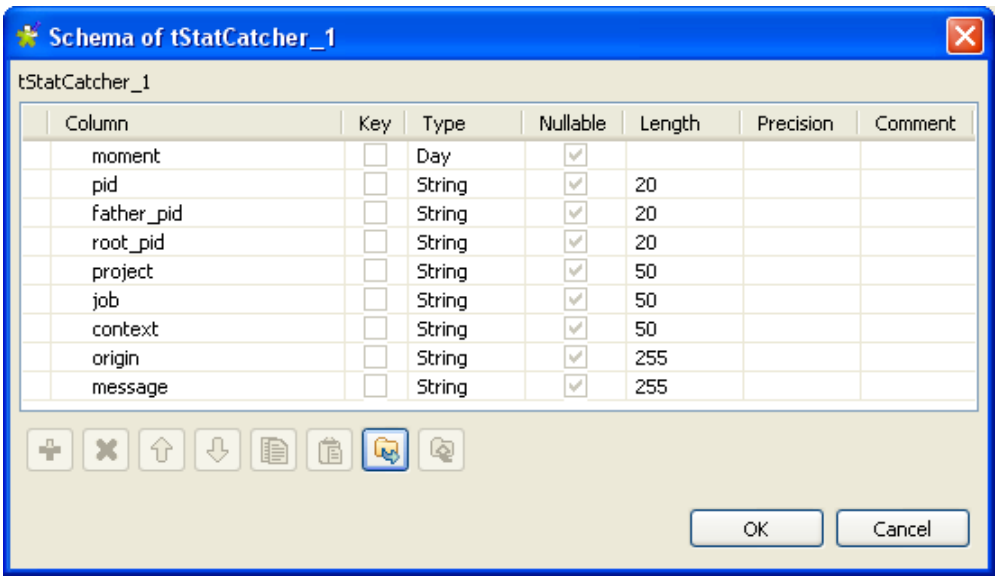
- And select the **tStatCatcher Statistics** check box to enable the statistics fetching operation.

- Then, define the output component's properties. In the **tFileOutputDelimited** Component view, browse to the output file or enter a name for the output file to be created. Define the delimiters, such as semi-colon, and the encoding.

- Click on **Edit schema** and make sure the schema is recollected from the input schema. If need be, click on **Sync Columns**.

- Then click on the **Basic settings** tab of the **Component** view, and select here as well the **tStatCatcher Statistics** check box to enable the processing data gathering.

- In the secondary Job, double-click on the **tStatCatcher** component. Note that the Properties are provided for information only as the schema representing the processing data to be gathered and aggregated in statistics, is defined and read-only.



- Define then the **tLogRow** to set the delimiter to be displayed on the console.
- Eventually, press **F6** to run the Job and display the Job result.

```
Starting job StatsCatch at 15:10 23/02/2007.
2007-02-23 15:10:30|3656|StatsCatch|Default||begin
2007-02-23
15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|begin
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|begin
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|end
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|0.0 seconds
2007-02-23 15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|end
2007-02-23 15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|0.0
seconds
2007-02-23 15:10:30|3656|StatsCatch|Default||end
2007-02-23 15:10:30|3656|StatsCatch|Default||0.0 seconds
Job StatsCatch ended at 15:10 23/02/2007. [exit code=0]
```

The log shows the Begin and End information for the Job itself and for each of the component used in the Job.

tWarn

tWarn Properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a **tLogCatcher** in order for the log data collected to be encapsulated and passed on to the output defined.

Component family	Logs & Errors	
Function	Provides a priority-rated message to the next component.	
Purpose	Triggers a warning often caught by the **tLogCatcher** component for exhaustive log.	
Basic settings	*Warn message*	Type in your warning message.
	Code	Define the code level.
	Priority	Enter the priority level as an integer.
Usage	Cannot be used as a start component. If an output component is connected to it, an input component should be preceding it.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tWarn**, see **tLogCatcher** scenarios:

- section *Scenario 1: warning & log on entries*
- section *Scenario 2: Log & kill a Job*

Misc group components

This chapter details the main components that you can find in **Misc** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The **Misc** family gathers miscellaneous components covering needs such as the creation of sets of dummy data rows, buffering data or loading context variables.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAddLocationFromIP

tAddLocationFromIP Properties

Component family	Misc	
Function	**tAddLocationFromIP** replaces IP addresses with geographical locations.	
Purpose	**tAddLocationFromIP** helps you to geolocate visitors through their IP addresses. It identifies visitors' geographical locations i.e. country, region, city, latitude, longitude, ZIP code...etc.using an IP address lookup database file.	
Basic settings	*Schema type* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. In Talend Open Studio for Big Data, the schema is always built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Database Filepath	The path to the IP address lookup database file.
	Input parameters	**Input column:** Select the input column from which the input values are to be taken.
		input value is a hostname: Check if the input column holds hostnames.
		input value is an IP address: Check if the input column holds IP addresses.
	Location type	**Country code**: Check to replace IP with country code.
		Country name: Check to replace IP with country name.
Usage	This component is an intermediary step in the data flow allowing to replace IP with geolocation information. It can not be a start component as it requires an input flow. It also requires an output component.	
Limitation	Due to license incompatibility, the following JAR required to use this component is not provided. You can easily add the JAR by following the *How to install external modules* section of *Talend Open Studio for Big Data User Guide*. • geoip.jar	

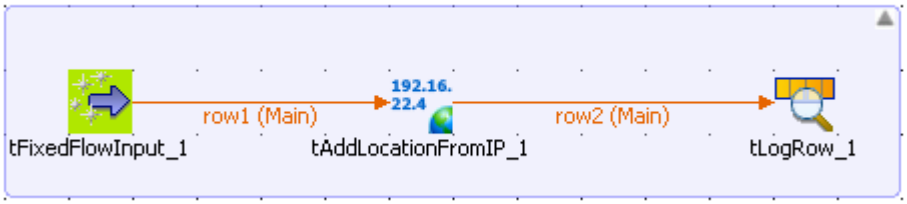
Scenario: Identifying a real-world geographic location of an IP

The following scenario creates a three-component Job that associates an IP with a geographical location. It obtains a site visitor's geographical location based on its IP.

Dropping and linking components

1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput**, **tAddLocationFromIP**, and **tLogRow**.

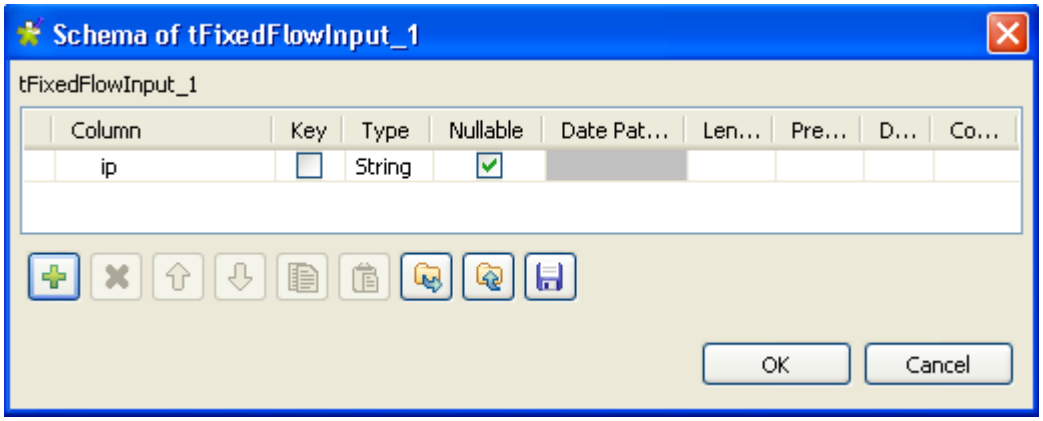
2. Connect the three components using **Row Main** links.



Configuring the components

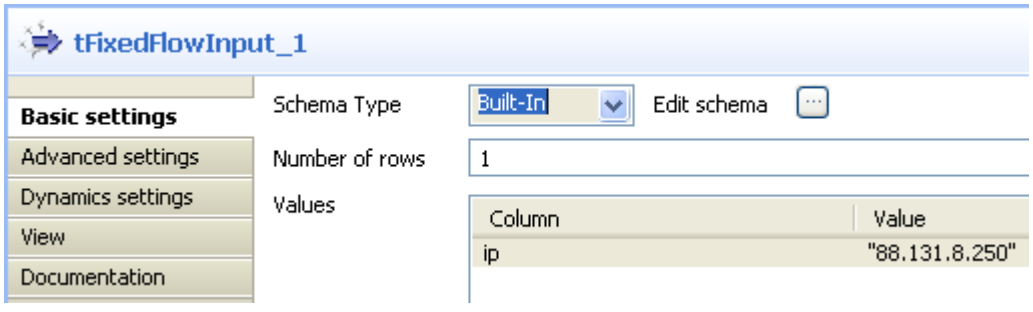
1. In the design workspace, select **tFixedFlowInput**, and click the **Component** tab to define the basic settings for **tFixedFlowInput**.

2. Click the **[...]** button next to **Edit Schema** to define the structure of the data you want to use as input. In this scenario, the schema is made of one column that holds an IP address.

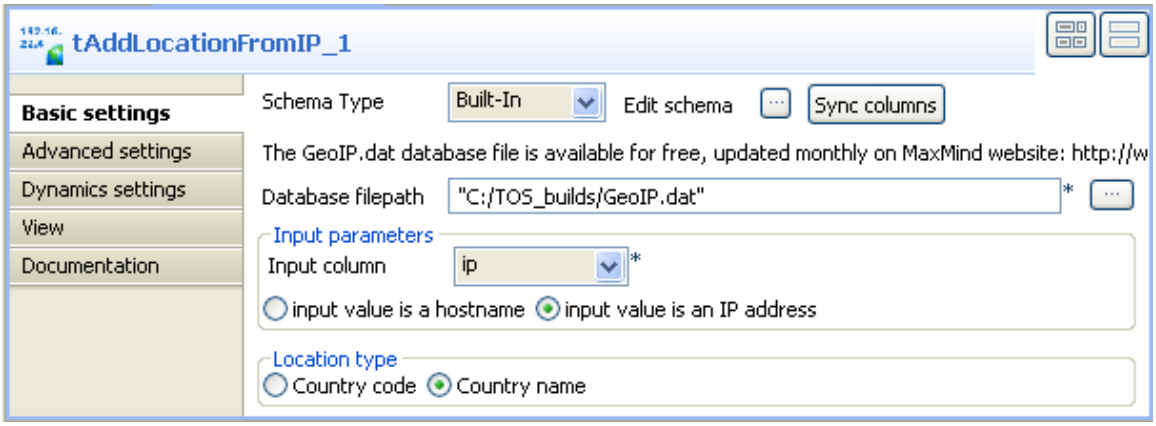


3. Click **OK** to close the dialog box, and accept propagating the changes when prompted by the system. The defined column is displayed in the **Values** panel of the **Basic settings** view.

4. In the **Number of rows** field, enter the number of rows to be generated, and click in the **Value** cell and set the value for the IP address.



5. In the design workspace, select **tAddLocationFromIP** and click the **Component** tab to define the basic settings for **tAddLocationFromIP**.



6. Click the **Sync columns** button to synchronize the schema with the input schema set with **tFixedFlowInput**.
7. Browse to the *GeoIP.dat* file to set its path in the **Database filepath** field.

   Ensure to download the latest version of the IP address lookup database file from the relevant site as indicated in the **Basic settings** view of **tAddLocationFromIp**.

8. In the **Input parameters** panel, set your input parameters as needed. In this scenario, the input column is the *ip* column defined earlier that holds an IP address.
9. In the **Location type** panel, set location type as needed. In this scenario, we want to display the country name.
10. In the design workspace, select **tLogRow** and click the **Component** tab and define the basic settings for **tLogRow** as needed. In this scenario, we want to display values in cells of a table.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6** or click **Run** in the **Run** tab to execute the Job.

```
Starting job add_location at 11:00 06/08/2008.
.------------+--------.
|       tLogRow_1       |
|=-----------+--------=|
|ip          |location|
|=-----------+--------=|
|88.131.8.250|Sweden  |
'------------+--------'

Job add_location ended at 11:00 06/08/2008. [exit code=0]
```

One row is generated to display the country name that is associated with the set IP address.

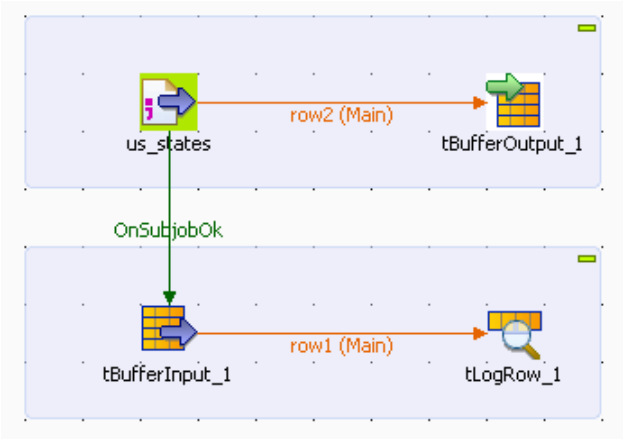
tBufferInput

tBufferInput properties

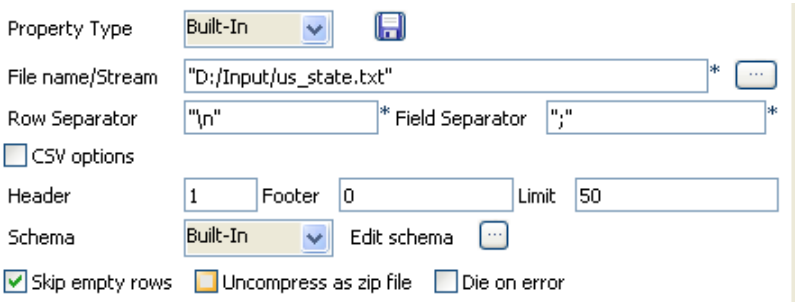
Component family	Misc	
Function	This component retrieves bufferized data in order to process it in a second subjob.	
Purpose	The **tBufferInput** component retrieves data bufferized via a **tBufferOutput** component, for example, to process it in another subjob.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of **tBufferInput**, the column position is more important than the column label as this will be taken into account.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository, hence can be reused in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Usage	This component is the start component of a secondary Job which is triggered automatically at the end of the main Job.	

Scenario: Retrieving bufferized data

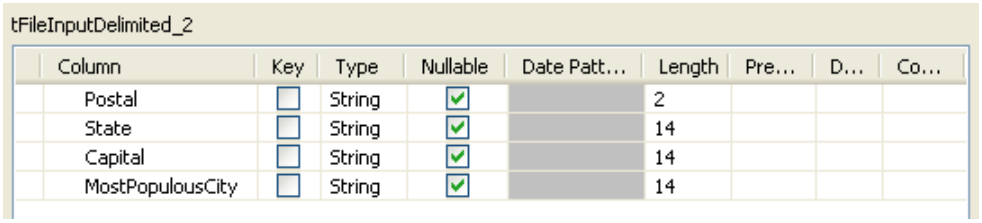
This scenario describes a Job that retrieves bufferized data from a subjob and displays it on the console.



- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited** and **tBufferOutput**.
- Select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.



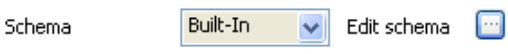
- In the **File Name** field, browse to the delimited file holding the data to be bufferized.
- Define the **Row** and **Field separators**, as well as the **Header**.
- Click **[...]** next to the **Schema type** field to describe the structure of the file.



- Describe the **Schema** of the data to be passed on to the **tBufferOutput** component.
- Select the **tBufferOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.

Generally speaking, the schema is propagated from the input component and automatically fed into the **tBufferOutput** schema. But you can also set part of the schema to be bufferized if you want to.

- Drop the **tBufferInput** and **tLogRow** components from the **Palette** onto the design workspace below the subjob you just created.
- Connect **tFileInputDelimited** and **tBufferInput** via a **Trigger** > **OnSubjobOk** link and connect **tBufferInput** and **tLogRow** via a **Row** > **Main** link.
- Double-click **tBufferInput** to set its **Basic settings** in the **Component** view.
- In the **Basic settings** view, click **[...]** next to the **Edit Schema** field to describe the structure of the file.



- Use the schema defined for the **tFileInputDelimited** component and click **OK**.
- The schema of the **tBufferInput** component is automatically propagated to the **tLogRow**. Otherwise, double-click **tLogRow** to display the **Component** view and click **Sync column**.
- Save your Job and press **F6** to execute it.

```
Starting job BufferFatherJob at 10:41 05/02/2008.
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
IA|Iowa|Des Moines|Des Moines
KS|Kansas|Topeka|Wichita
KY|Kentucky|Frankfort|Louisville
LA|Louisiana|Baton Rouge|New Orleans *
```

The standard console returns the data retrieved from the buffer memory.

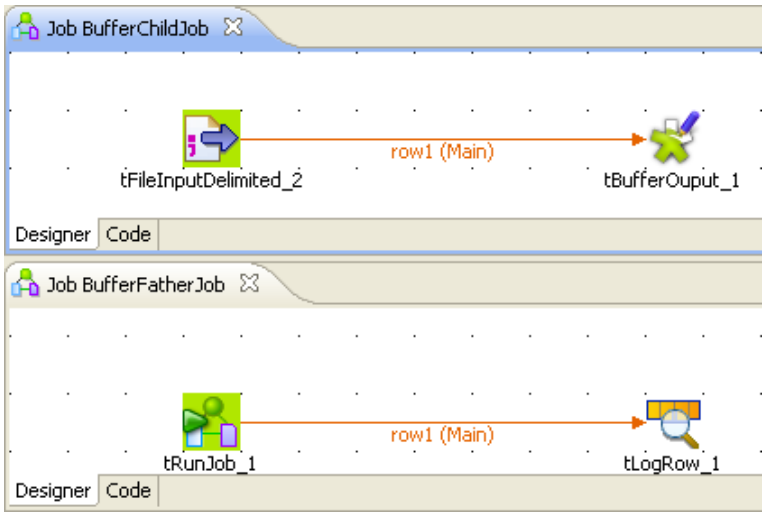
tBufferOutput

tBufferOutput properties

Component family	Misc	
Function	This component collects data in a buffer in order to access it later via webservice for example.	
Purpose	This component allows a Webservice to access data. Indeed it had been designed to be exported as Webservice in order to access data on the web application server directly. For more information, see *Talend Open Studio for Big Data* **User Guide**.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of the **tBufferOutput**, the column position is more important than the column label as this will be taken into account.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Usage	This component is not startable (green background) and it requires an output component.	

Scenario 1: Buffering data (Java)

This scenario describes an intentionally basic Job that bufferizes data in a child job while a parent Job simply displays the bufferized data onto the standard output console. For an example of how to use **tBufferOutput** to access output data directly on the Web application server, see section *Scenario 2: Buffering output data on the webapp server*.

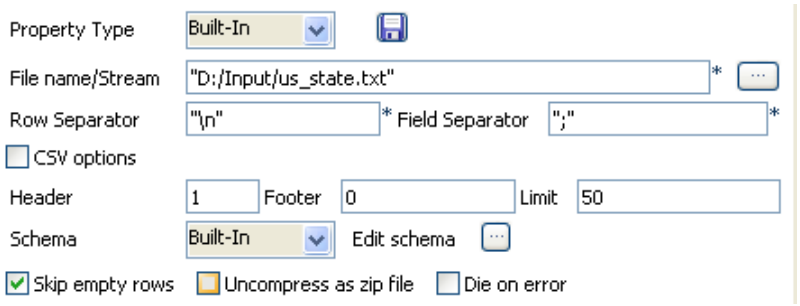


- Create two Jobs: a first Job (*BufferFatherJob*) runs the second Job and displays its content onto the **Run** console. The second Job (*BufferChildJob*) stores the defined data into a buffer memory.

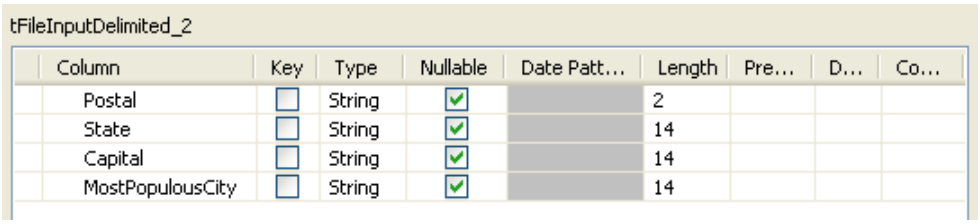
- On the first Job, drop the following components: **tRunJob** and **tLogRow** from the **Palette** to the design workspace.
- On the second Job, drop the following components: **tFileInputDelimited** and **tBufferOutput** the same way.

Let's set the parameters of the second Job first:

- Select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.

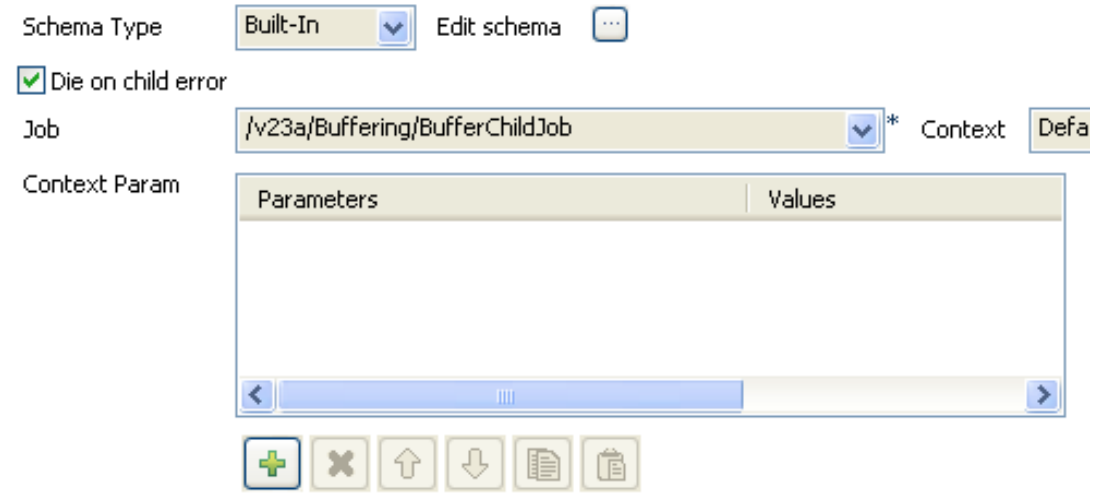


- In **File Name**, browse to the delimited file whose data are to be bufferized.
- Define the **Row** and **Field separators**, as well as the **Header**.



- Describe the **Schema** of the data to be passed on to the **tBufferOutput** component.
- Select the **tBufferOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.

- Generally the schema is propagated from the input component and automatically fed into the **tBufferOutput** schema. But you could also set part of the schema to be bufferized if you want to.
- Now on the other Job (*BufferFatherJob*) Design, define the parameters of the **tRunJob** component.



- Edit the Schema if relevant and select the column to be displayed. The schema can be identical to the bufferized schema or different.

- You could also define context parameters to be used for this particular execution. To keep it simple, the default context with no particular setting is used for this use case.

Press **F6** to execute the parent Job. The **tRunJob** looks after executing the child Job and returns the data onto the standard console:

```
Starting job BufferFatherJob at 10:41 05/02/2008.
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
IA|Iowa|Des Moines|Des Moines
KS|Kansas|Topeka|Wichita
KY|Kentucky|Frankfort|Louisville
```

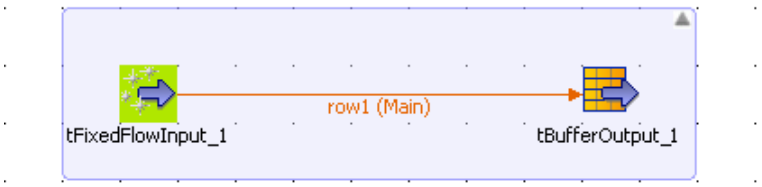
Scenario 2: Buffering output data on the webapp server

This scenario describes a Job that is called as a Webservice and stores the output data in a buffer directly on the server of the Web application. This scenario creates first a Webservice oriented Job with context variables, and next exports the Job as a Webservice.

Creating a Webservice-oriented Job with context variables:

- Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput** and **tBufferOutput**.

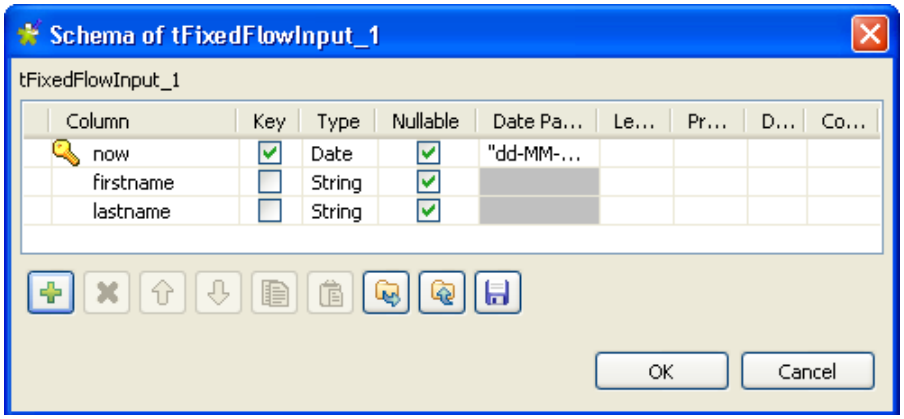
- Connect **tFixedFlowInput** to **tBufferOutput** using a **Row Main** link.



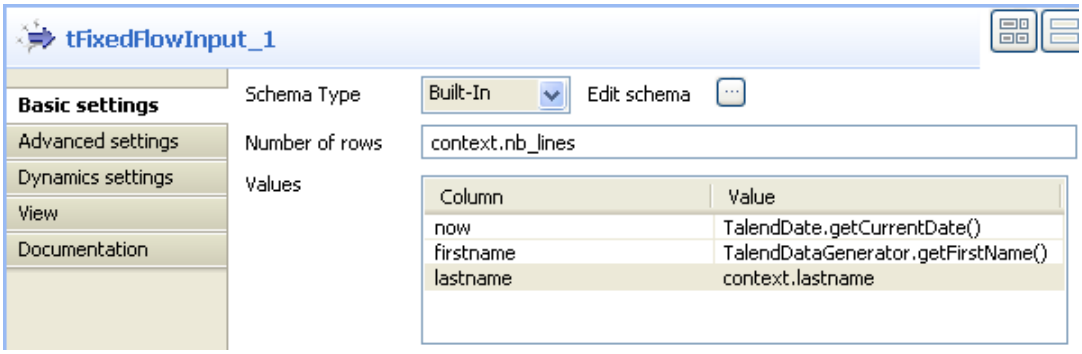
- In the design workspace, select **tFixedFlowInput**.

- Click the **Component** tab to define the basic settings for **tFixedFlowInput**.

- Set the **Schema Type** to **Built-In** and click the three-dot **[...]** button next to **Edit Schema** to describe the data structure you want to create from internal variables. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.



- Click the plus button to add the three parameter lines and define your variables.
- Click **OK** to close the dialog box and accept propagating the changes when prompted by the system. The three defined columns display in the **Values** panel of the **Basic settings** view of **tFixedFlowInput**.

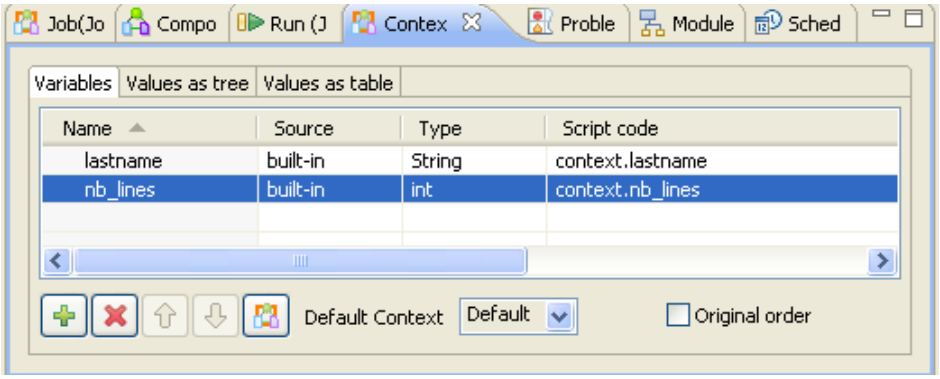


- Click in the **Value** cell of each of the first two defined columns and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *TalendDate.getCurrentDate()* and *talendDatagenerator.getFirstName*, for the *now* and *firstname* columns respectively.

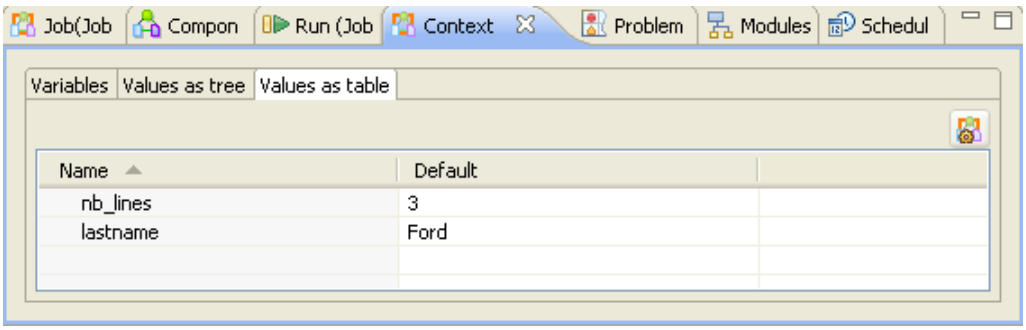
For this scenario, we want to define two context variables: *nb_lines* and *lastname*. In the first we set the number of lines to be generated, and in the second we set the last name to display in the output list. The **tFixedFlowInput** component will generate the number of lines set in the context variable with the three columns: *now*, *firstname* and *lastname*. For more information about how to create and use context variables, see *Talend Open Studio for Big Data* **User Guide**.

To define the two context variables:

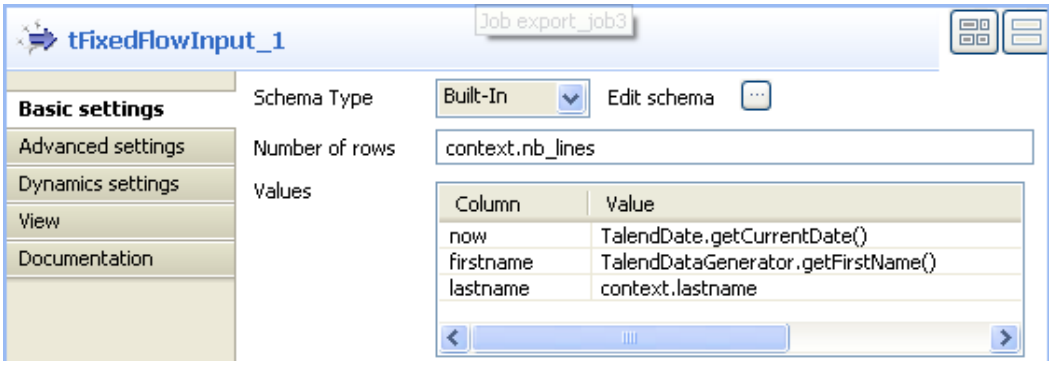
- Select **tFixedFlowInput** and click the **Contexts** tab.
- In the **Variables** view, click the plus button to add two parameter lines and define them.



- Click the **Values as table** tab and define the first parameter to set the number of lines to be generated and the second to set the last name to be displayed.



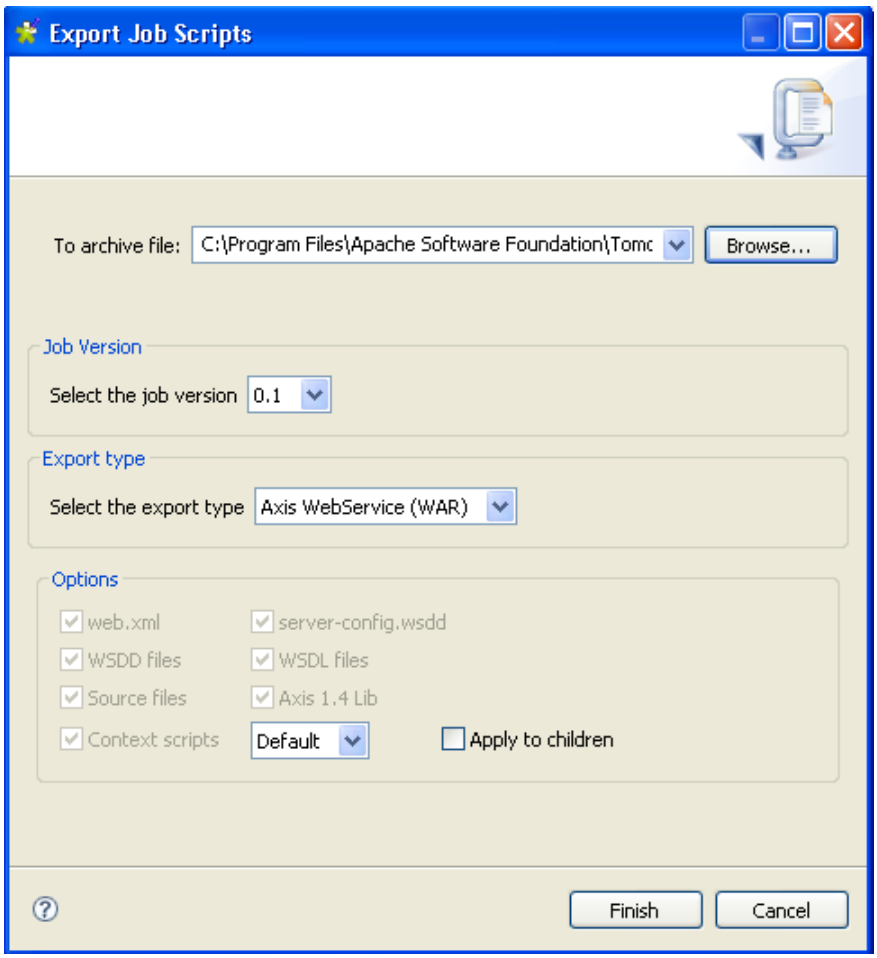
- Click the **Component** tab to go back to the **Basic settings** view of **tFixedFlowInput**.
- Click in the **Value** cell of *lastname* column and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *context.lastname*, the context variable you created for the last name column.



Exporting your Job as a Webservice:

Before exporting your Job as a Web service, see *Talend Open Studio for Big Data* **User Guide** for more information.

- In the **Repository** tree view, right-click on the above created Job and select **Export Job Scripts**. The **[Export Job Scripts]** dialog box displays.

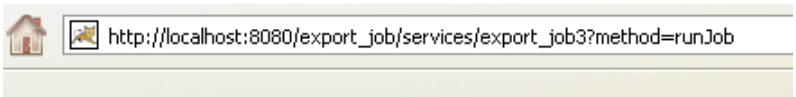


- Click the **Browse...** button to select a directory to archive your Job in.
- In the **Export type** panel, select the export type you want to use in the Tomcat webapp directory (WAR in this example) and click **Finish**. The **[Export Job Scripts]** dialog box disappears.
- Copy the War folder and paste it in a Tomcat webapp directory.

Scenario 3: Calling a Job with context variables from a browser

This scenario describes how to call the Job you created in scenario 2 from your browser with/without modifying the values of the context variables.

Type the following URL into your browser: *http://localhost:8080//export_job/services/export_job3?method=runJob* where “export_job” is the name of the webapp directory deployed in Tomcat and “export_job3” is the name of the Job.



Click **Enter** to execute your Job from your browser.

```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">William</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Millard</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">James</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      </runJobReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

The Job uses the default values of the context variables: *nb_lines* and *lastname*, that is it generates three lines with the current date, first name and Ford as a last name.

You can modify the values of the context variables directly from your browser. To call the Job from your browser and modify the values of the two context variables, type the following URL:

http://localhost:8080//export_job/services/export_job3?method=runJob&arg1=--context_param %20lastname=MASSY&arg2=--context_param%20nb_lines=2.

%20 stands for a blank space in the URL language. In the first argument “arg1”, you set the value of the context variable to display “MASSY” as last name. In the second argument “arg2”, you set the value of the context variable to “2” to generate only two lines.

Click **Enter** to execute your Job from your browser.

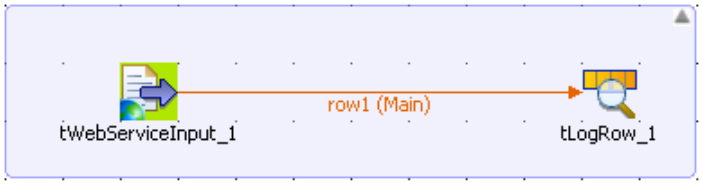
```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Richard</ns1:item>
          <ns1:item xsi:type="xsd:string">MASSY</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Theodore</ns1:item>
          <ns1:item xsi:type="xsd:string">MASSY</ns1:item>
        </ns1:item>
      </runJobReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

The Job generates two lines with MASSY as last name.

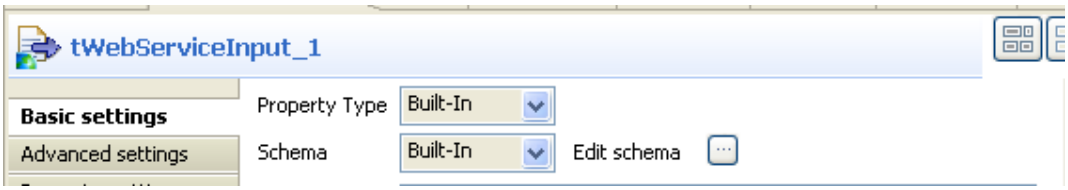
Scenario 4: Calling a Job exported as Webservice in another Job

This scenario describes a Job that calls another Job exported as a Webservice using the **tWebServiceInput**. This scenario will call the Job created in scenario 2.

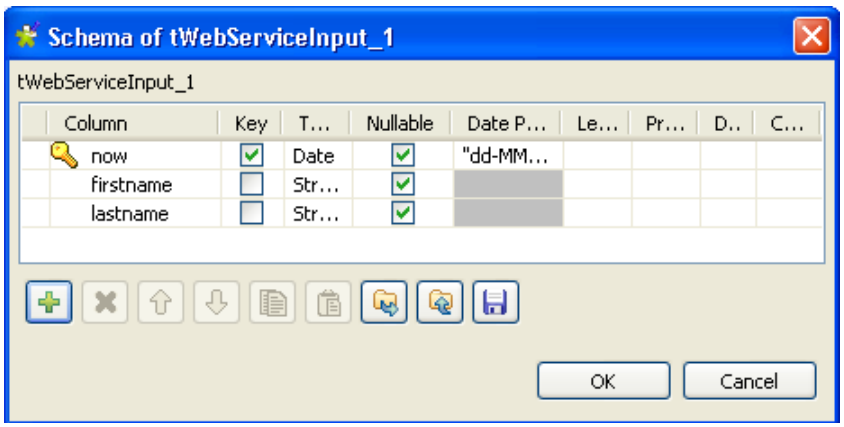
- Drop the following components from the **Palette** onto the design workspace: **tWebServiceInput** and **tLogRow**.
- Connect **tWebserviceInput** to **tLogRow** using a **Row Main** link.



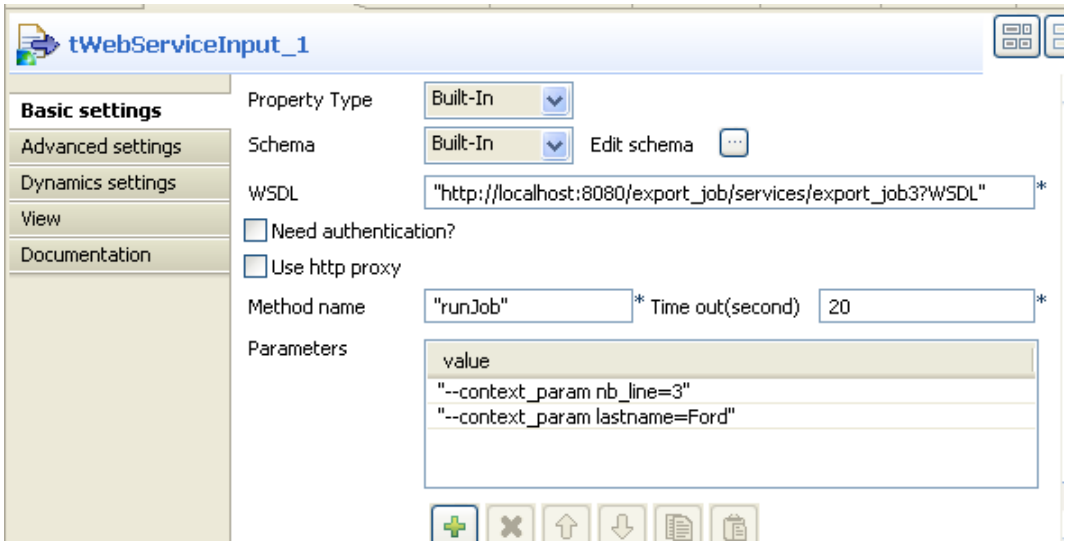
- In the design workspace, select **tWebServiceInput**.
- Click the **Component** tab to define the basic settings for **tWebServiceInput**.



- Set the **Schema Type** to **Built-In** and click the three-dot **[...]** button next to **Edit Schema** to describe the data structure you want to call from the exported Job. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.



- Click the plus button to add the three parameter lines and define your variables.Click **OK** to close the dialog box.
- In the **WSDL** field of the **Basic settings** view of **tWebServiceInput**, enter the URL *http://localhost:8080/export_job/services/export_job3?WSDL* where “export_job” is the name od the webapp directory where the Job to call is stored and “export_job3” is the name of the Job itself.



- In the **Method name** field, enter *runJob*.
- In the **Parameters** panel, Click the plus button to add two parameter lines to define your context variables.
- Click in the first **Value** cell to enter the parameter to set the number of generated lines using the following syntax: *--context_param nb_line=3*.
- Click in the second **Value** cell to enter the parameter to set the last name to display using the following syntax: *--context_param lastname=Ford*.
- Select **tLogRow** and click the **Component** tab to display the component view.
- Set the **Basic settings** for the **tLogRow** component to display the output data in a tabular mode. For more information, see section *tLogRow*.
- Save your Job and press **F6** to execute it.

```
Starting job Call_Webservice_In_Job at 14:12 01/08/2008.
.----------+---------+--------.
|           tLogRow_1           |
|=---------+---------+--------=|
|now       |firstname|lastname|
|=---------+---------+--------=|
|01-08-2008|Ronald   |Ford    |
|01-08-2008|Woodrow  |Ford    |
|01-08-2008|Franklin |Ford    |
'----------+---------+--------'

Job Call_Webservice_In_Job ended at 14:12 01/08/2008. [exit code=0]
```

The system generates three columns with the current date, first name, and last name and displays them onto the log console in a tabular mode.

tContextDump

tContextDump properties

Component family	Misc	
Function	**tContextDump** makes a dump copy the values of the active Job context.	
Purpose	**tContextDump** can be used to transform the current context parameters into a flow that can then be used in a **tContextLoad**. This feature is very convenient in order to define once only the context and be able to reuse it in numerous Jobs via the **tContextLoad**.	
Basic settings	*Schema type* and *Edit schema*	In the **tContextDump** use, the schema is read only and made of two columns, Key and Value, corresponding to the parameter name and the parameter value to be copied. A schema is a row description, i.e., it defines the fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and Job flowcharts. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Print operations	Select this check box to display the context parameters set in the **Run** view.
Usage	This component creates from the current context values, a data flow, therefore it requires to be connected to an output component.	
Limitation	**tContextDump** does not create any non-defined context variable.	

Related Scenario

No scenario is available for this component yet.

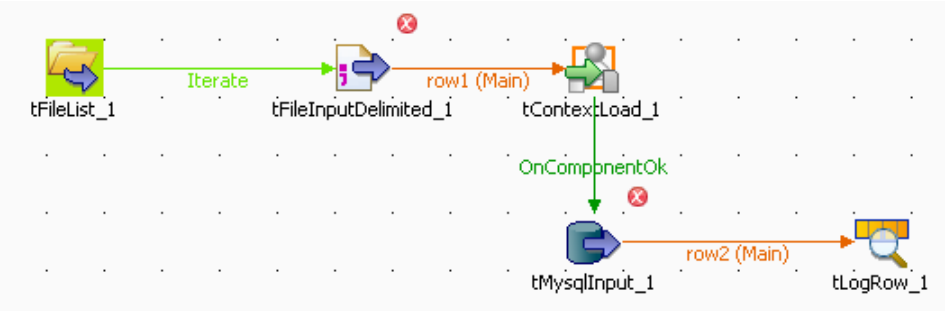
tContextLoad

tContextLoad properties

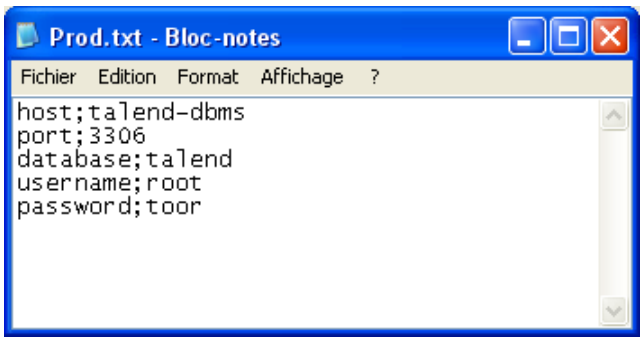
Component family	Misc	
Function	**tContextLoad** modifies dynamically the values of the active context.	
Purpose	**tContextLoad** can be used to load a context from a flow. This component performs also two controls. It warns when the parameters defined in the incoming flow are not defined in the context, and the other way around, it also warns when a context value is not initialized in the incoming flow. But note that this does not block the processing.	
Basic settings	*Schema type* and *Edit schema*	In **tContextLoad**, the schema must be made of two columns, including the parameter name and the parameter value to be loaded. A schema is a row description, i.e., it defines the fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	If a variable loaded, but not in the context	If a variable is loaded but does not appear in the context, select how the notification must be displayed. In the shape of an **Error**, a **warning** or an information (**info**).
	If a variable in the context, but not loaded	If a variable appears in the context but is not loaded, select how the notification must be displayed. In the shape of an **Error**, a **warning** or an information (**info**)
	Print operations	Select this check box to display the context parameters set in the **Run** view.
	Disable errors	Select this check box to prevent the error from displaying.
	Disable warnings	Select this check box to prevent the warning from displaying.
	Disable infos	Select this check box to prevent the information from displaying.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component relies on the data flow to load the context values to be used, therefore it requires a preceding input component and thus cannot be a start component.	
Limitation	**tContextLoad** does not create any non-defined variable in the default context.	

Scenario: Dynamic context use in MySQL DB insert

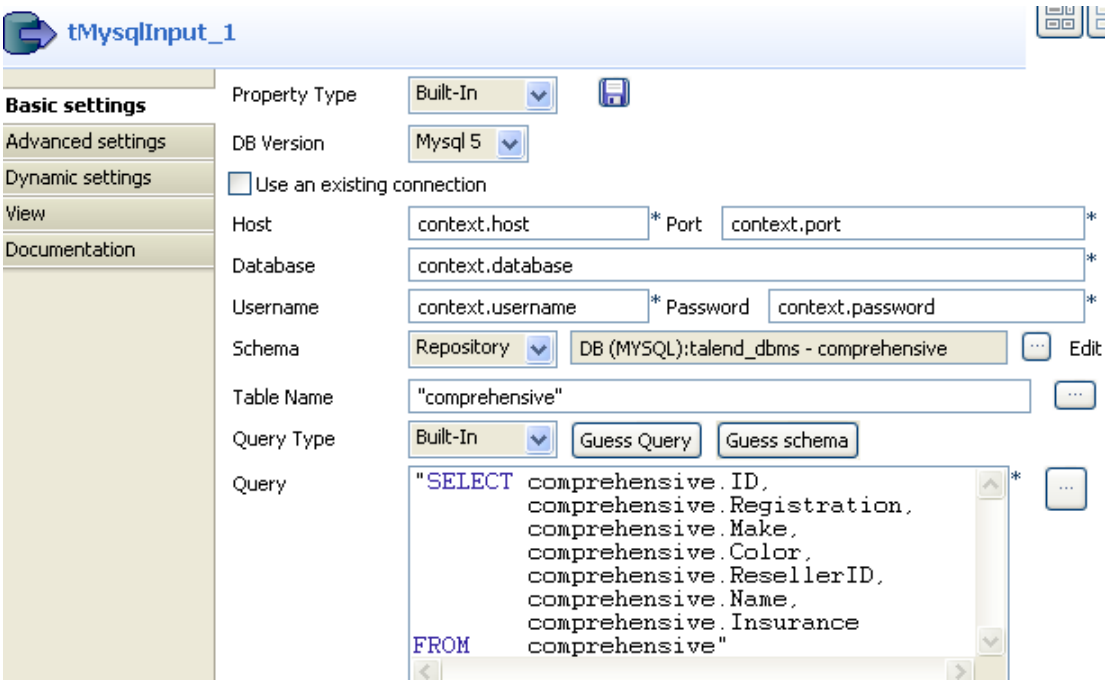
This scenario is made of two subjobs. The first subjob aims at dynamically load the context parameters, and the second subjob uses the loaded context to display the content of a DB table.



- For the first subjob, drop a **tFilelist**, **tFileInputDelimited**, **tContextLoad** from the **Palette** to the design workspace.
- Drop **tMysqlInput** and a **tLogRow** the same way for the second subjob.
- Connect all the components together.
- Create as many delimited files as there are different contexts and store them in a specific directory, named *Contexts*. In this scenario, *test.txt* contains the local database connection details for testing purpose. And *prod.txt* holds the actual production db details.
- Each file is made of two fields, contain the parameter name and the corresponding value, according to the context.



- In the **tFileList** component **Basic settings** panel, select the directory where both context files, *test* and *prod*, are held.
- In the **tFileInputDelimited** component **Basic settings** panel, press **Ctrl+Space bar** to access the global variable list. Select `tFileList_1.CURRENT_FILEPATH` to loop on the context files' directory.
- Define the schema manually (Built-in). It contains two columns defined as: *Key* and *Value*.
- Accept the defined schema to be propagated to the next component (**tContextLoad**).
- For this scenario, select the **Print operations** check box in order for the context parameters in use to be displayed on the **Run** panel.
- Then double-click to open the **tMySQLInput** component **Basic settings**.
- For each of the field values being stored in a context file, press **F5** and define the user-defined context parameter. For example: The **Host** field has for value parameter `context.host`, as the parameter name is *host* in the context file. Its actual value being *talend-dbms*.



- Then fill in the **Schema** information. If you stored the schema in the **Repository Metadata**, then you can retrieve it by selecting **Repository** and the relevant entry in the list.
- In the **Query** field, type in the SQL query to be executed on the DB table specified. In this case, a simple SELECT of the columns of the table, which will be displayed on the **Run** tab, through the **tLogRow** component.
- Eventually, press **F6** to run the Job.

```
Starting job tContextLoad at 16:19 05/03/2010.

tContextLoad_1 set key "host" with value "talend-dbms"
tContextLoad_1 set key "port" with value "3306"
tContextLoad_1 set key "database" with value "talend"
tContextLoad_1 set key "username" with value "root"
tContextLoad_1 set key "password" with value "toor"
12|4322 DP 76|BMW|purple|1|vaesneng|DFA6968
15|0142 CB 08|BMW|yellow|9|mauant|GCY8927
18|8545 GP 25|Mercedes|pink|2|vaesot|BNC6696
24|5382 KC 94|Volkswagen|grey|10|otbouh|GZE3017
40|8386 GH 71|Mercedes|orange|8|carmau|
67|0261 PY 55|Honda|green|6|hirtken|QXK4682
70|1850 CL 46|Mercedes|purple|7|oinegall|WAS9249
77|5653 HW 34|Toyota|yellow|2|carbone|GLN1029
91|6016 DF 08|Renault|pink|5|nanhirt|HWR9250
```

The context parameters as well as the select values from the DB table are all displayed on the **Run** view.

tFixedFlowInput

tFixedFlowInput properties

Component family	Misc	
Function	**tFixedFlowInput** generates as many lines and columns as you want using the context variables.	
Purpose	**tFixedFlowInput** allows you to generate fixed flow from internal variables.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository, hence can be reused in various projects and job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Mode	From the three options, select the mode that you want to use. **Use Single Table :** Enter the data that you want to generate in the relevant value field. **Use Inline Table :** Add the row(s) that you want to generate. **Use Inline Content :** Enter the data that you want to generate, separated by the separators that you have already defined in the **Row** and **Field Separator** fields.
	Number of rows	Enter the number of lines to be generated.
	Values	Between inverted commas, enter the values corresponding to the columns you defined in the schema dialog box via the **Edit schema** button.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as a start or intermediate component and thus requires an output component.	

Related scenarios

For related scenarios, see:

- section *Scenario 2: Buffering output data on the webapp server*.
- section *Scenario: Iterating on a DB table and listing its column names*.
- section *Scenario: Filtering and searching a list of names*.

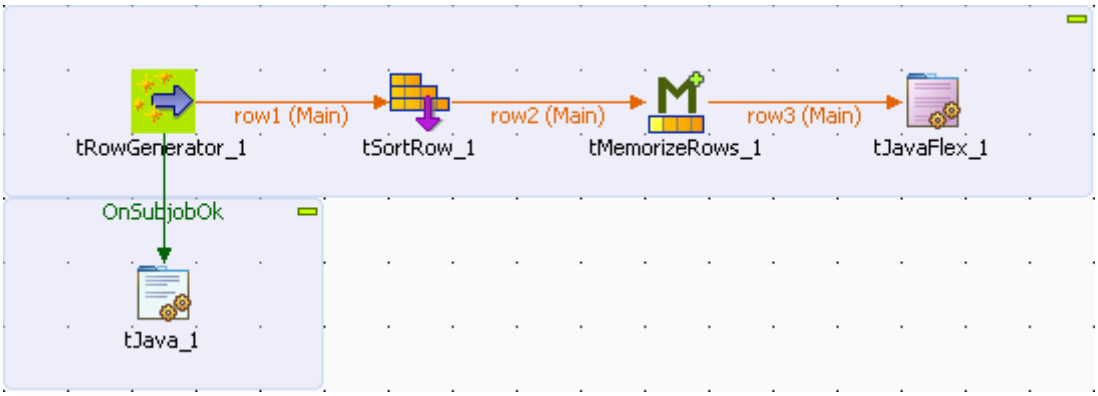
tMemorizeRows

tMemorizeRows properties

Component family	Misc	
Function	**tMemorizeRows** temporarily memorizes an array of incoming data in a row by row sequence and instantiates this array by indexing each of the memorized rows from 0. The maximum number of rows to be memorized at any given time is defined in the **Basic settings** view.	
Purpose	**tMemorizeRows** memorizes a sequence of rows that pass this component and then allows its following component(s) to perform operations of your interest on the memorized rows.	
Basic settings	*Schema type* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. - Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. - Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository, hence can be reused in various projects and job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Row count to memorize	Define the row count to be memorized.
	Columns to memorize	Select the columns to be memorized from the incoming data schema.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as intermediate step in a data flow or the last step before beginning a subjob. Note: You can use the global variable NB_LINE_ROWS to retrieve the value of the **Row count to memorize** field of the **tMemorizeRows** component.	
Connections		Outgoing links (from one component to another): **Row**: Main **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.

Scenario: Counting the occurrences of different ages

This scenario counts how many different ages there are within a group of 12 customers. In this scenario, the customer data is generated at random.



This Job uses 5 components which are:

- **tRowGenerator**: it generates 12 rows of customer data containing IDs, names and ages of the 12 customers.
- **tSortRow**: it sorts the 12 rows according to the age data.
- **tMemorizeRows**: it temporarily memorizes a specific number of incoming data rows at any give time and indexes the memorized data rows.
- **tJavaFlex**: it compares the age values of the data memorized by the preceding component, counts the occurrences of different ages and displays these ages in the **Run** view.
- **tJava**: it displays the number of occurrences of different ages.

To replicate this scenario, proceed as follows:

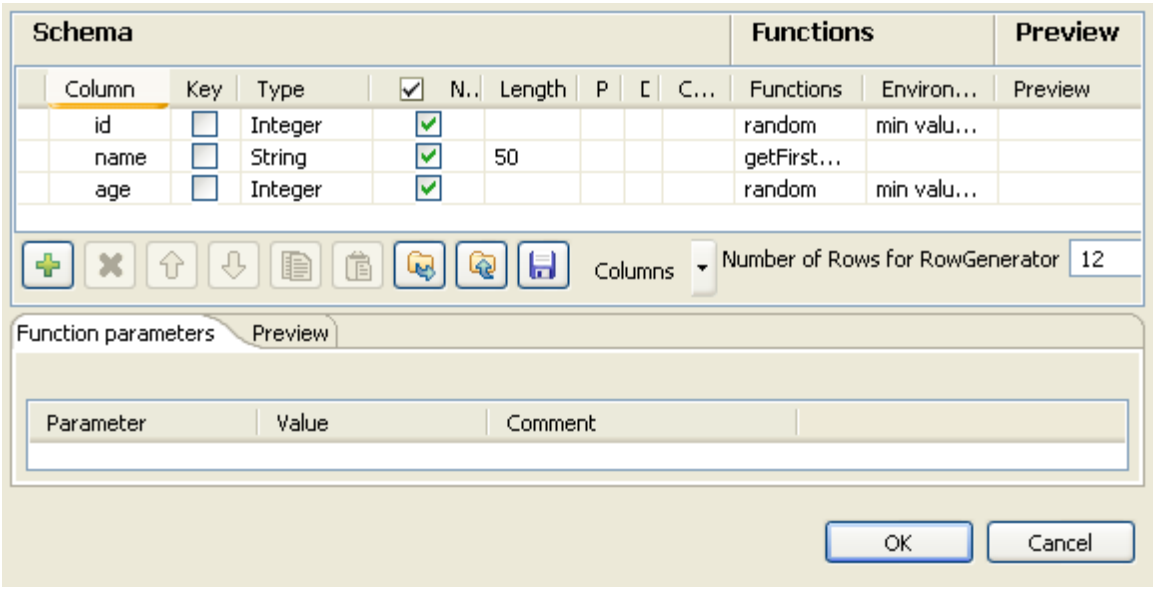
Dropping and linking the components

1. Drop **tRowGenerator**, **tSortRow**, **tMemorizeRows**, **tJavaFlex** and **tJava** on the design workspace.
2. Connect **tRowGenerator** to **tSortRow** using the **Row** > **Main** link.
3. Do the same to link together **tSortRow**, **tMemorizeRows** and **tJavaFlex** using the **Row** > **Main** link.
4. Connect **tRowGenerator** to **tJava** using the **Trigger** > **OnSubjobOk** link.

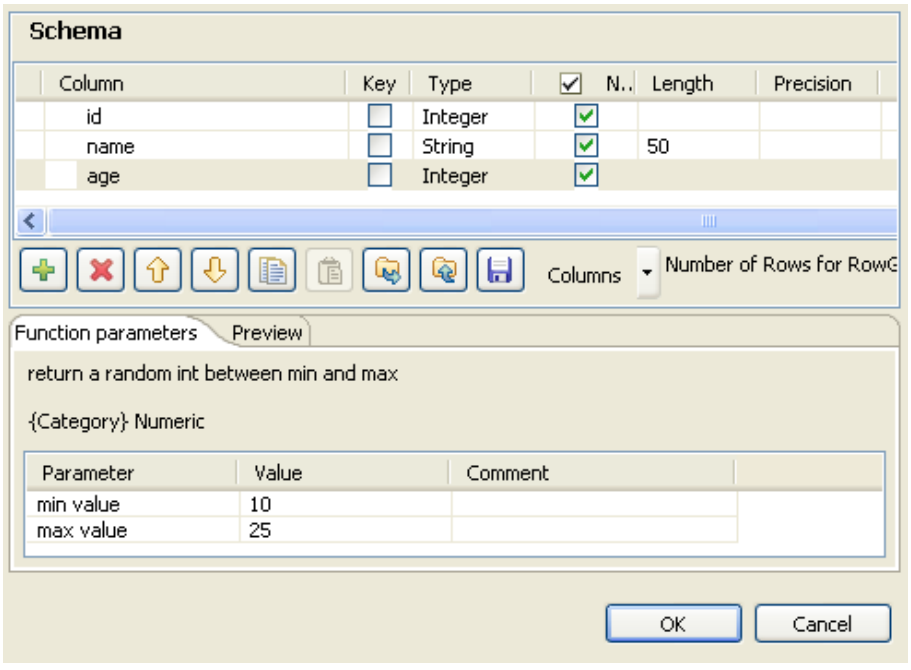
Configuring the components

Configuring the tRowGenerator component

1. Double click the **tRowGenerator** component to open the its editor.



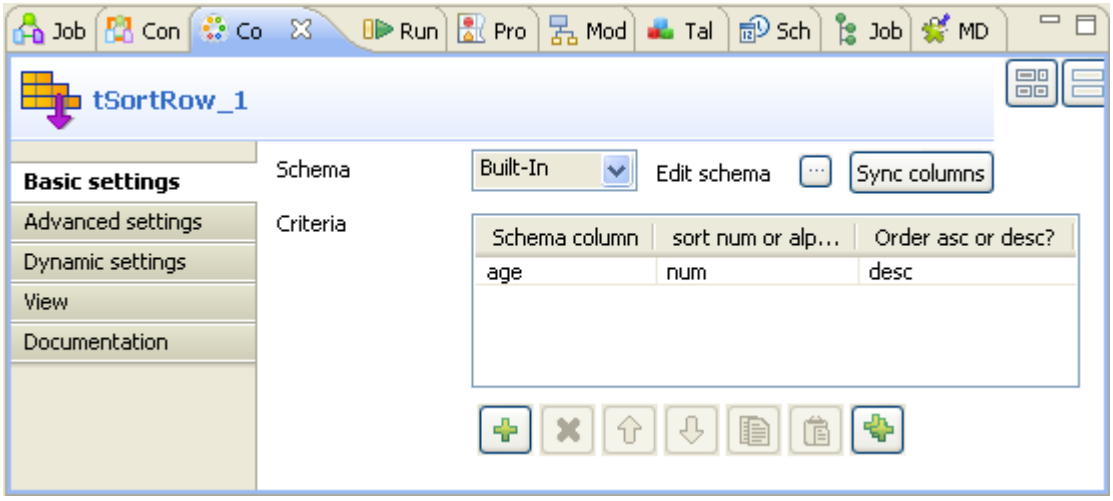
2. In this editor, click the plus button three times to add three columns and name them as: *id*, *name*, *age*.
3. In the **Type** column, select **Integer** for *id* and *age*.
4. In the **Length** column, enter *50* for *name*.
5. In the **Functions** column, select **random** for *id* and *age*, then select **getFirstName** for *name*.
6. In the field of **Number of Rows for RowGenerator**, type in 12.
7. In the **Column** column, click *age* to open its corresponding **Function parameters** view in the lower part of this editor.



In the **Value** column of the **Function parameters** view, type in the minimum age and maximum age that will be generated for the 12 customers. In this example, they are 10 and 25.

Configuring the tSortRow component

1. Double click **tSortRow** to open its **Component** view.



2. In the **Criteria** table, click the plus button to add one row.

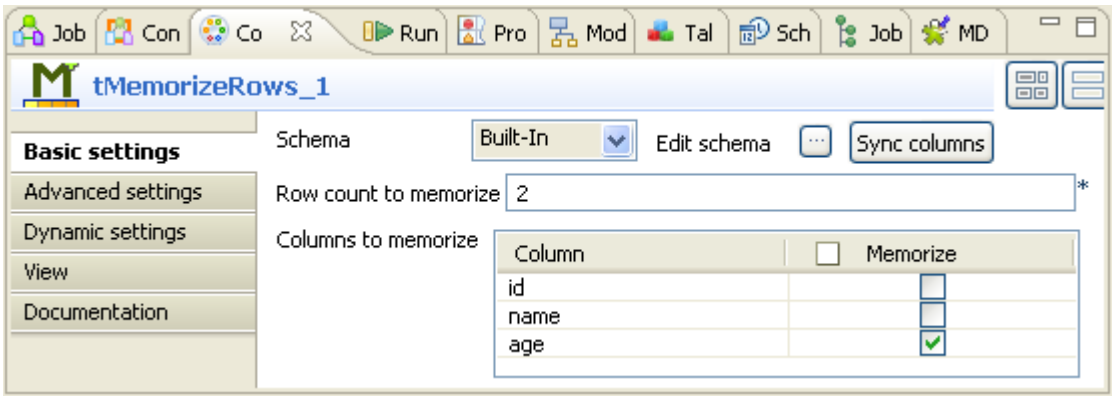
3. In the **Schema column** column, select the data column you want to base the sorting operation on. In this example, select *age* as it is the ages that should be compared and counted.

4. In the **Sort num or alpha** column, select the type of the sorting operation. In this example, select **num**, that is numerical, as *age* is integer.

5. In the **Order asc or desc** column, select **desc** as the sorting order for this scenario.

Configuring the tMemorizeRows component

1. Double click **tMemorizeRows** to open its **Component** view.

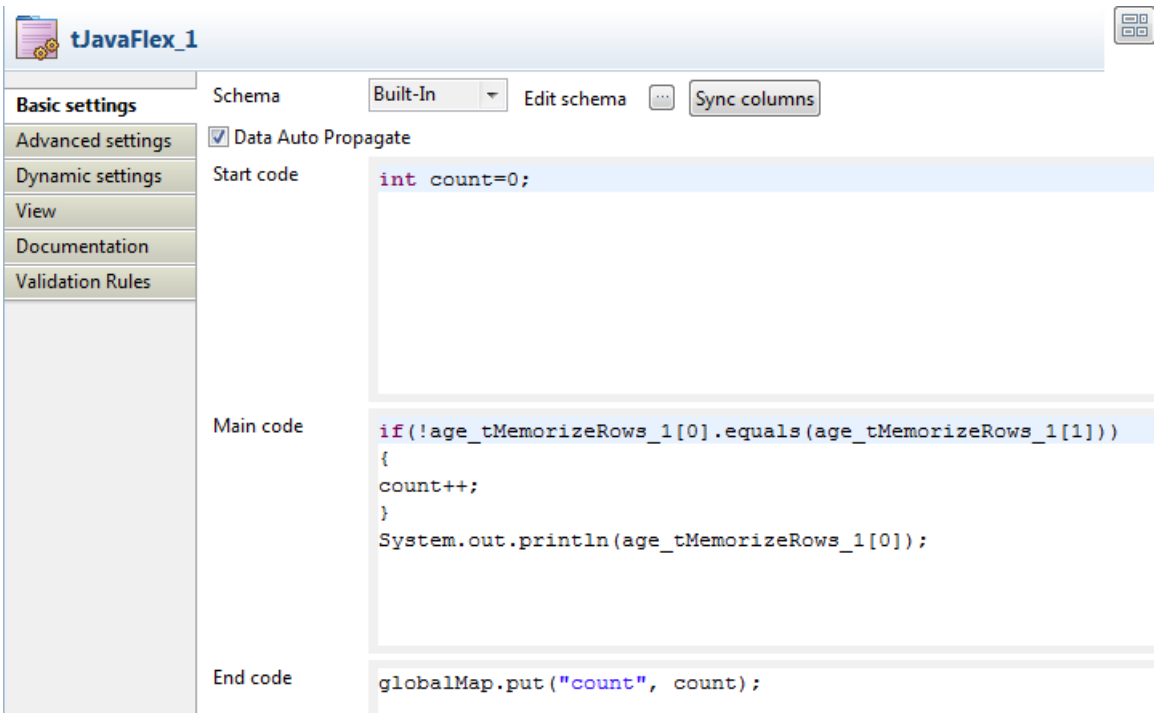


2. In the **Row count to memorize** field, type in the maximum number of rows to be memorized at any given time. As you need to compare ages of two customers for each time, enter *2*. Thus, this component memorizes two rows at maximum at any given moment and always indexes the newly incoming row as 0 and the previously incoming row as 1.

3. In the **Memorize** column of the **Columns to memorize** table, select the check box(es) to determine the column(s) to be memorized. In this example, select the check box corresponding to *age*.

Configuring the tJavaFlex and tJava components

1. Double click **tJavaFlex** to open its **Component** view.



2. In the **Start code** area, enter the Java code that will be called during the initialization phase. In this example, type in `int count=0;` in order to declare a variable *count* and assign the value *0* to it.

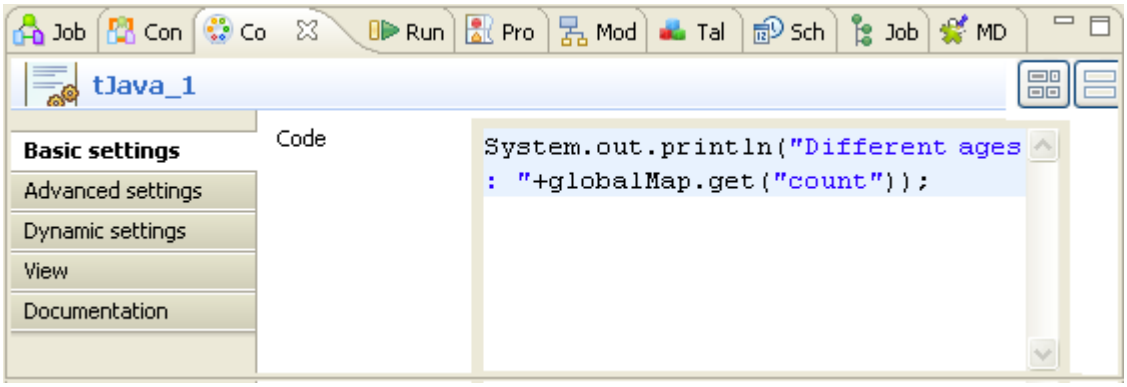
3. In the **Main code** area, enter the Java code to be applied to each row in the data flow. In this scenario, type in

```
if(!age_tMemorizeRows_1[0].equals(age_tMemorizeRows_1[1]))
{
count++;
}
System.out.println(age_tMemorizeRows_1[0]);
```

   This code compares two ages memorized by **tMemorizeRows** each time and count one change every time when the ages are found different. Then this code displays the ages that have been indexed as 0 by **tMemorizeRows**.

4. In the **End code** area, enter the Java code that will be called during the closing phase. In this example, type in `globalMap.put("count", count);` to output the count result.

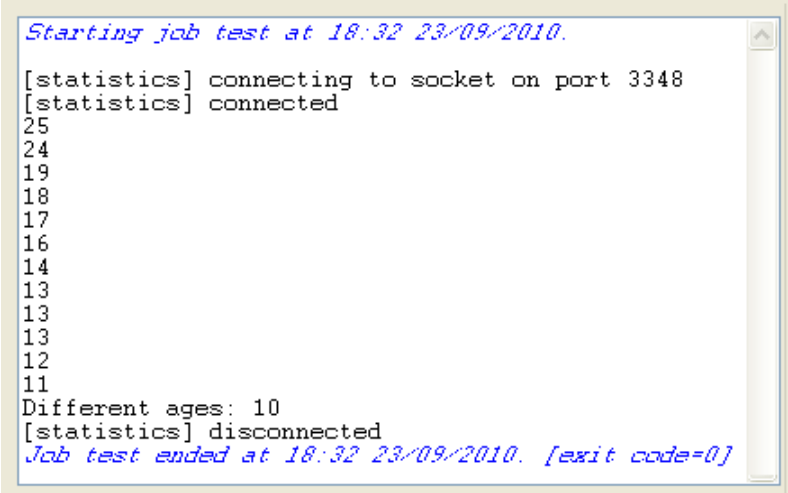
5. Double click **tJava** to open its **Component** view.



6. In the **Code** area, type in the code `System.out.println("Different ages: "+globalMap.get("count"));` to retrieve the count result.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Press **F6**, or click **Run** on the **Run** console to execute the Job.



In the console, you can read that there are 10 different ages within the group of 12 customers.

tMsgBox

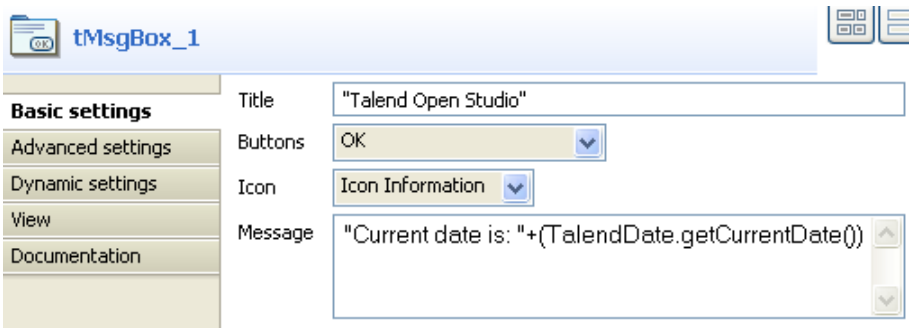
tMsgBox properties

Component family	Misc	
Function	Opens a dialog box with an OK button requiring action from the user.	
Purpose	**tMsgBox** is a graphical break in the job execution progress.	
Basic settings	*Title*	Text entered shows on the title bar of the dialog box created.
	Buttons	Listbox of buttons you want to include in the dialog box. The button combinations are restricted and cannot be changed. The **Question** button displays the**Mask Answer** check box. Select this check box if you want to mask the answer you type in the pop-up window that opens when you run the Job.
	Icon	Icon shows on the title bar of the dialog box.
	Message	Free text to display as message on the dialog box. Text can be dynamic (for example: retrieve and show a file name).
Usage	This component can be used as intermediate step in a data flow or as a start or an end object in the Job flowchart. It can be connected to the next/previous component using either a **Row** or **Iterate** link.	
Limitation	n/a	

Scenario: ‘Hello world!’ type test

The following scenario creates a single-component Job, where **tMsgBox** is used to display the pid (process id) in place of the traditional “Hello World!” message.

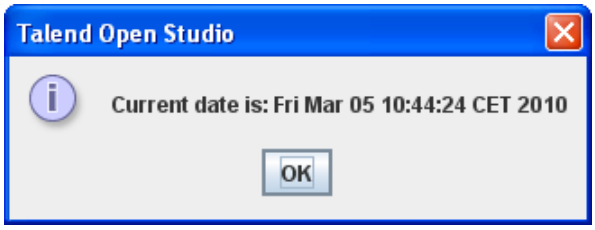
- Drop a **tMsgBox** component from the **Palette** to the design workspace.
- Define the dialog box display properties:



- ‘Title’ is the message box title, it can be any variable.
- In the Message field, enter `"Current date is: "` between double quotation marks. Then click CTRL+Space to display the autocompletion list and select the following system routine, `TalendDate.getCurrentDate`. Put brackets around this routine.

- Switch to the **Run** tab to execute the Job defined.

The Message box displays the message and requires the user to click **OK** to go to the next component or end the Job.



After the user clicked **OK**, the **Run** log is updated accordingly.

Related topic: see *Talend Open Studio for Big Data* **User Guide**.

tRowGenerator

tRowGenerator properties

Component family	Misc	
Function	**tRowGenerator** generates as many rows and fields as are required using random values taken from a list.	
Purpose	Can be used to create an input flow in a Job for testing purposes, in particular for boundary test sets	
Basic settings	*Schema type* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or stored remotely in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: Select the Repository file where the properties are stored. When selected, the fields that follow are filled in automatically using fetched data.
	RowGenerator editor	The editor allows you to define the columns and the nature of data to be generated. You can use predefined routines or type in the function to be used to generate the data specified
Usage	The **tRowGenerator** Editor's ease of use allows users without any Java knowledge to generate random data for test purposes.	
Limitation	n/a	

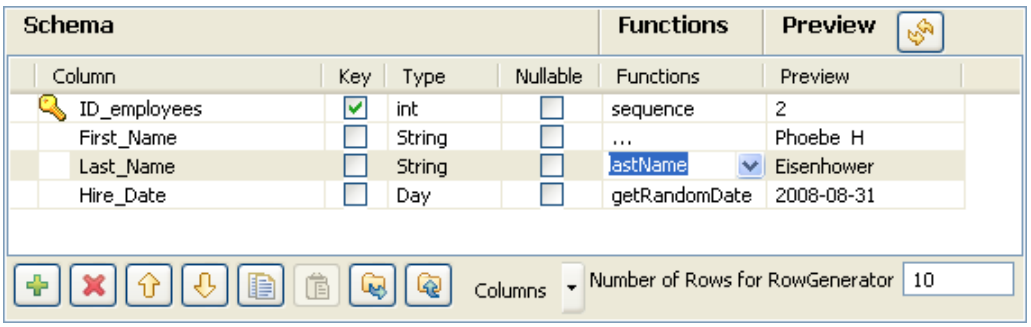
The **tRowGenerator** Editor opens up on a separate window made of two parts:

- a **Schema** definition panel at the top of the window
- and a **Function** definition and preview panel at the bottom.

Defining the schema

First you need to define the structure of data to be generated.

- Add as many columns to your schema as needed, using the **plus (+)** button.
- Type in the names of the columns to be created in the **Columns** area and select the **Key** check box if required
- Make sure you define then the nature of the data contained in the column, by selecting the **Type** in the list. According to the type you select, the list of **Functions** offered will differ. This information is therefore compulsory.

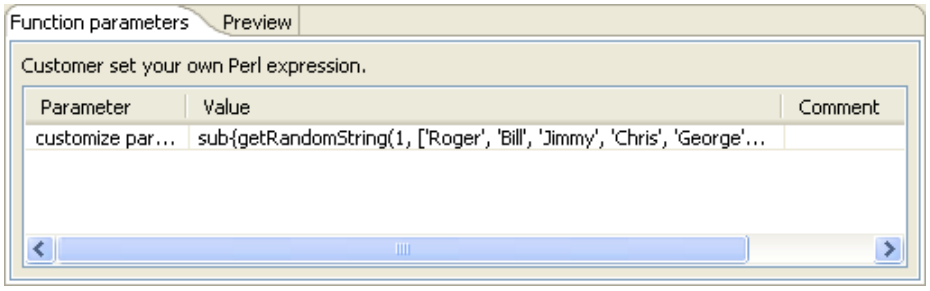


- Some extra information, although not required, might be useful such as **Length**, **Precision** or **Comment**. You can also hide these columns, by clicking on the **Columns** drop-down button next to the toolbar, and unchecking the relevant entries on the list.
- In the **Function** area, you can select the predefined routine/function if one of them corresponds to your needs.You can also add to this list any routine you stored in the **Routine** area of the **Repository**. Or you can type in the function you want to use in the **Function** definition panel. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
- Click **Refresh** to have a preview of the data generated.
- Type in a number of rows to be generated. The more rows to be generated, the longer it'll take to carry out the generation operation.

Defining the function

Select the **[...]** under **Function** in the Schema definition panel in order to customize the function parameters.

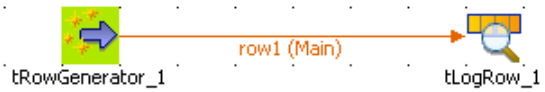
- Select the **Function parameters** tab
- The **Parameter** area displays **Customized parameter** as function name (read-only)



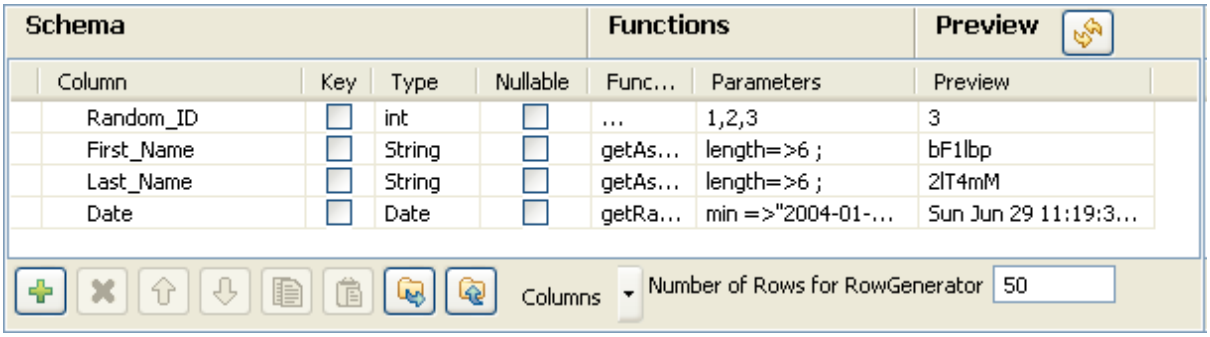
- In the **Value** area, type in the Java function to be used to generate the data specified.
- Click on the **Preview** tab and click **Preview** to check out a sample of the data generated.

Scenario: Generating random java data

The following scenario creates a two-component Job, generating 50 rows structured as follows: a randomly picked-up ID in a 1-to-3 range, a random ascii First Name and Last Name generation and a random date taken in a defined range.

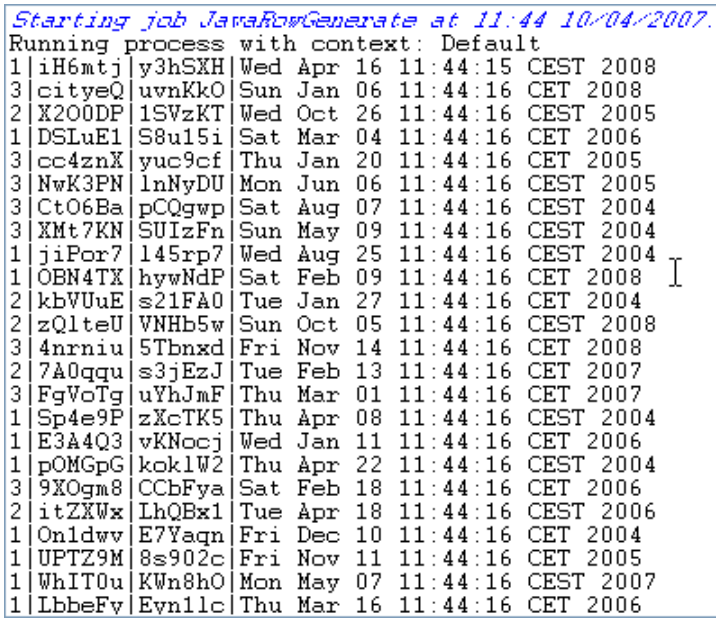


- Drop a **tRowGenerator** and a **tLogRow** component from the **Palette** to the design workspace.
- Right-click **tRowGenerator** and select **Row** > **Main**. Drag this main row link onto the **tLogRow** component and release when the plug symbol displays.
- Double click **tRowGenerator** to open the Editor.
- Define the fields to be generated.



- The random ID column is of integer type, the First and Last names are of string type and the Date is of date type.
- In the **Function** list, select the relevant function or set on the three dots for custom function.
- On the **Function parameters** tab, define the Values to be randomly picked up.

- *First_Name* and *Last_Name* columns are to be generated using the getAsciiRandomString function that is predefined in the system routines. By default the length defined is 6 characters long. You can change this if need be.
- The *Date* column calls the predefined getRandomDate function. You can edit the parameter values in the **Function parameters** tab.
- Set the **Number of Rows** to be generated to 50.
- Click **OK** to validate the setting.
- Double click **tLogRow** to view the Basic settings. The default setting is retained for this Job.
- Press **F6** to run the Job.



The 50 rows are generated following the setting defined in the **tRowGenerator** editor and the output is displayed in the **Run** console.

Orchestration components

This chapter details the main components that you can find in **Orchestration** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Orchestration family groups together components that help you to sequence or orchestrate tasks or processing in your Jobs or subjobs and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tFileList

tFileList belongs to two component families: File and Orchestration. For more information on **tFileList**, see section *tFileList*.

tFlowToIterate

tFlowToIterate Properties

Component family	Orchestration	
Function	**tFlowToIterate** iterates on the input data and generates global variables.	
Purpose	This component is used to read data line by line from the input flow and store the data entries in iterative global variables.	
Basic settings	*Use the default (key, value) in global variables*	When selected, the system uses the default value of the global variable in the current Job.
	Customize	**key**: Type in a name for the new global variable. Press **Ctrl+Space** to access all available variables either global or user-defined.
		value: Click in the cell to access a list of the columns attached to the defined global variable.
Usage	You cannot use this component as a start component. **tFlowToIterate** requires an output component.	
Global Variables		**Number of Lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. **Current iteration**: Indicates the current iteration number. This is available as a **Flow** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

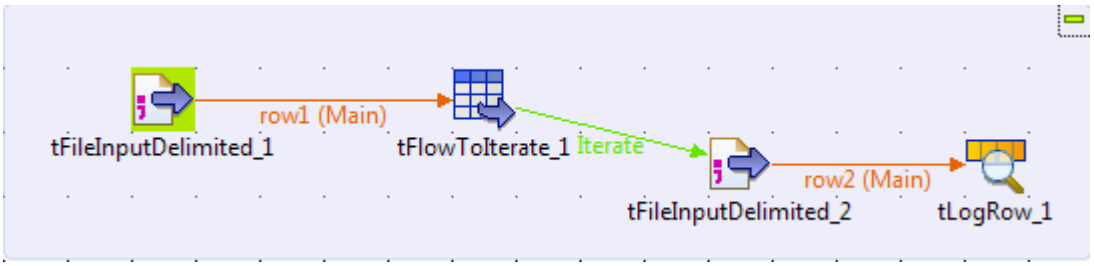
Scenario: Transforming data flow to a list

The following scenario describes a Job that reads a list of files from a defined input file, iterates on each of the files and displays their content row by row on the **Run** console.

Setting up the Job

1. Drop the following components from the **Palette** onto the design workspace: two **tFileInputDelimited** components, a **tFlowToIterate**, and a **tLogRow**.

2. Connect the first **tFileInputDelimited** to **tFlowToIterate** using a **Row** > **Main** link, **tFlowToIterate** to the second **tFileInputDelimited** using an **Iterate** link, and the second **tFileInputDelimited** to **tLogRow** using a **Row** > **Main** link.

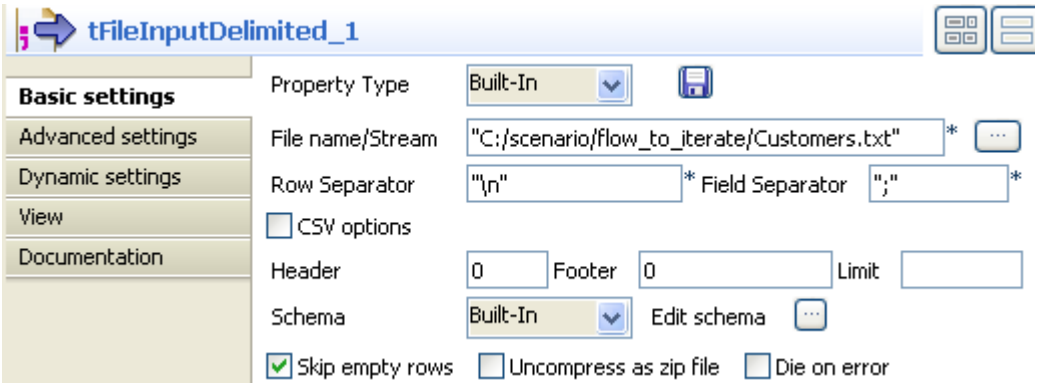


Configuring the Components

1. Double-click the first **tFileInputDelimited** to display its **Basic settings** view.

2. Click the **[...]** button next to the **File Name** field to select the path to the input file.

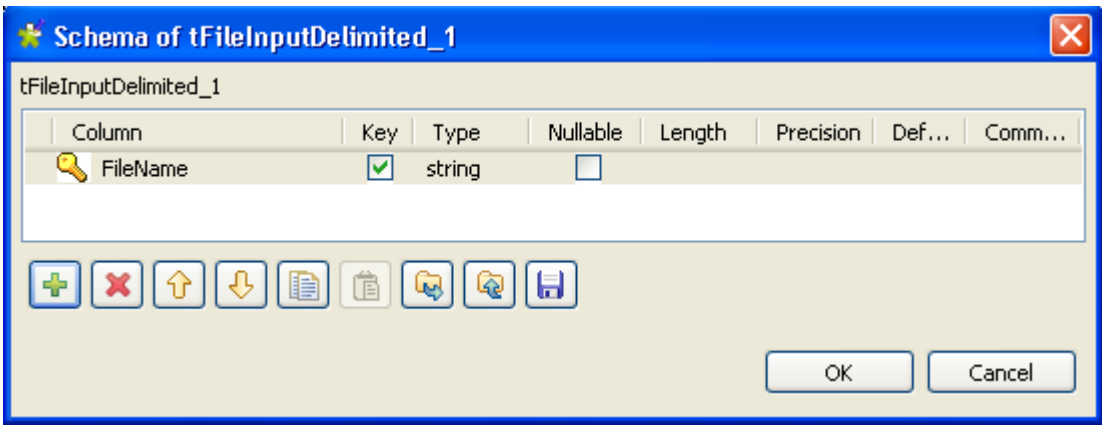
The **File Name** field is mandatory.



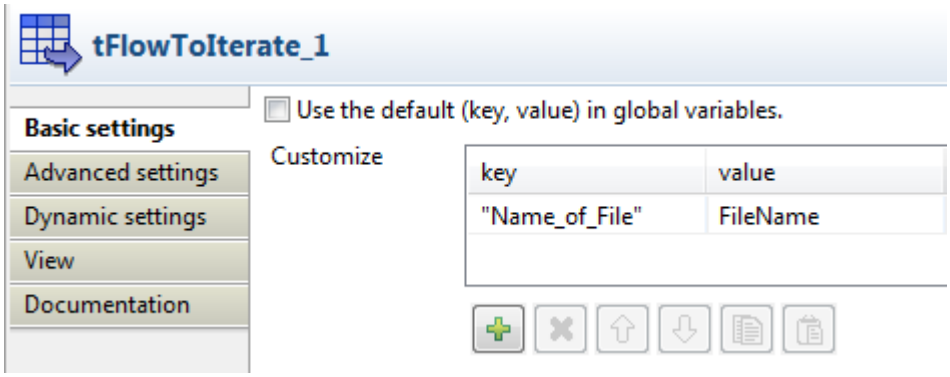
The input file used in this scenario is *Customers.txt*. It is a text file that contains a list of names of three other simple text files: *Name.txt, E-mail.txt* and *Address.txt*. The first text file, *Name.txt*, is made of one column holding customers' names. The second text file, *E-mail.txt*, is made of one column holding customers' e-mail addresses. The third text file, *Address.txt*, is made of one column holding customers' postal addresses.

Fill in all other fields as needed. For more information, see section *tFileInputDelimited properties*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows.

3. Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of one column, *FileName*.



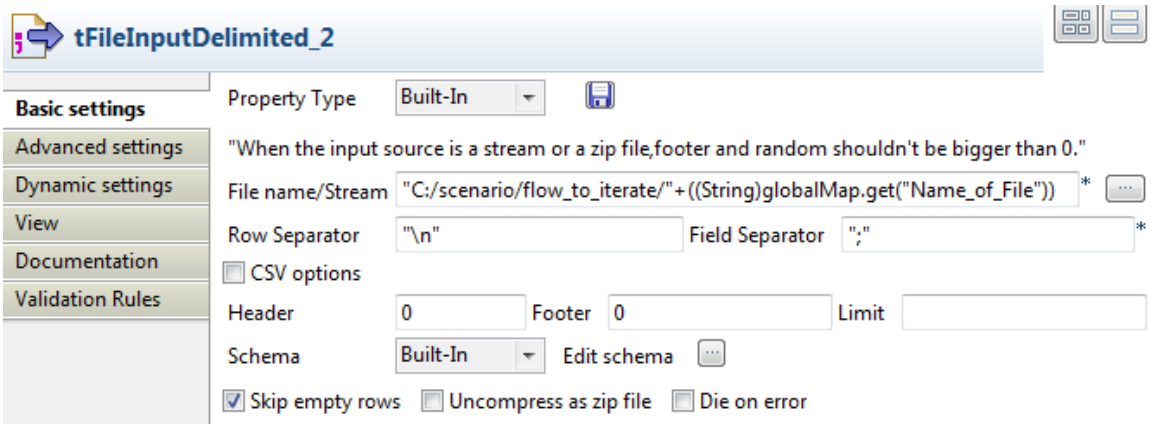
4. Double-click **tFlowToIterate** to display its **Basic settings** view.



Click the plus button to add new parameter lines and define your variables, and click in the **key** cell to enter the variable name as desired. In this scenario, one variable is defined: `"Name_of_File"`.

Alternatively, you can select the **Use the default (key, value) in global variables** check box to use the default in global variables.

5. Double-click the second **tFileInputDelimited** to display its **Basic settings** view.



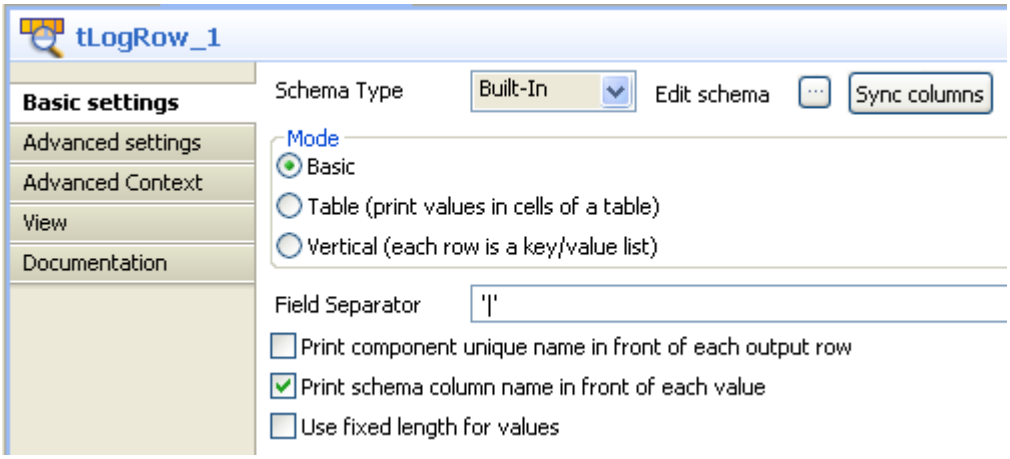
In the **File name** field, enter the directory of the files to be read, and then press **Ctrl+Space** to select the global variable *"Name_of_File"*. In this scenario, the syntax is as follows:

```
"C:/scenario/flow_to_iterate/"+((String)globalMap.get("Name_of_File"))
```

Click **Edit schema** to define the schema column name. In this scenario, it is *RowContent*.

Fill in all other fields as needed. For more information, see section *tFileInputDelimited properties*.

6. In the design workspace, select the last component, **tLogRow**, and click the **Component** tab to define its basic settings.



Define your settings as needed. For more information, see section *tLogRow properties*.

Saving and executing the Job

1. Save your Job by pressing **Ctrl+S**.

2. Execute the Job by pressing **F6** or clicking **Run** on the **Run** tab.

```
Starting job Practice_8_27 at 16:12 28/08/2012.

[statistics] connecting to socket on port 3580
[statistics] connected
RowContent: Madison Moore
RowContent: Andrew Taylor
RowContent: Christopher Anderson
RowContent: madison_moor@hotmail.com
RowContent: andre_taylor@usamail.com
RowContent: chrisopher_anderson@hotmail.com
RowContent: Madison Moore
RowContent:      100 MAIN ST
RowContent:      PO BOX 1022
RowContent:      SEATTLE WA 98104
RowContent:      USA
RowContent: Andrew Taylor
RowContent:      300 BOYLSTON AVE E
RowContent:      SEATTLE WA 98102
RowContent:      USA
RowContent: Christopher Anderson
RowContent:      500 DOCTOR ST
RowContent:      SEATTLE WA 98100
RowContent:      USA
[statistics] disconnected
Job Practice_8_27 ended at 16:12 28/08/2012. [exit code=0]
```

Customers' names, customers' e-mails, and customers' postal addresses appear on the console preceded by the schema column name.

tForeach

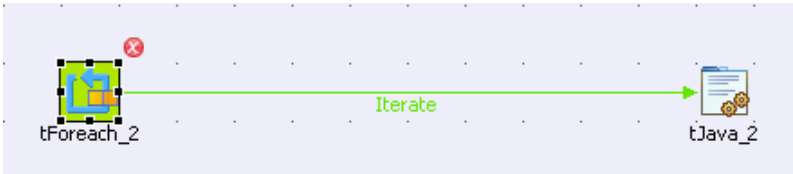
tForeach Properties

Component Family	Orchestration	
Function	**tForeach** creates a loop on a list for an iterate link.	
Purpose	**tForeach** allows you to to create a loop on a list for an iterate link.	
Basic settings	*Values*	Use the **[+]** button to add rows to the **Values** table. Then click on the fields to enter the list values to be iterated upon, between double quotation marks.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	**tForeach** is an input component and requires an Iterate link to connect it to another component.	
Limitation	n/a	

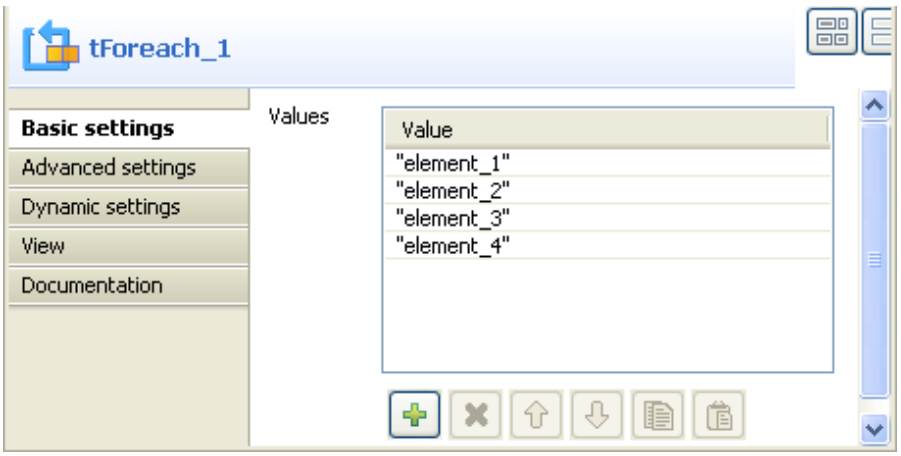
Scenario: Iterating on a list and retrieving the values

This scenario describes a two component Job in which a list is created and iterated upon in a **tForEach** component. The values are then retrieved in a **tJava** component.

- rop a **tForeach** and a **tJava** component onto the design workspace:

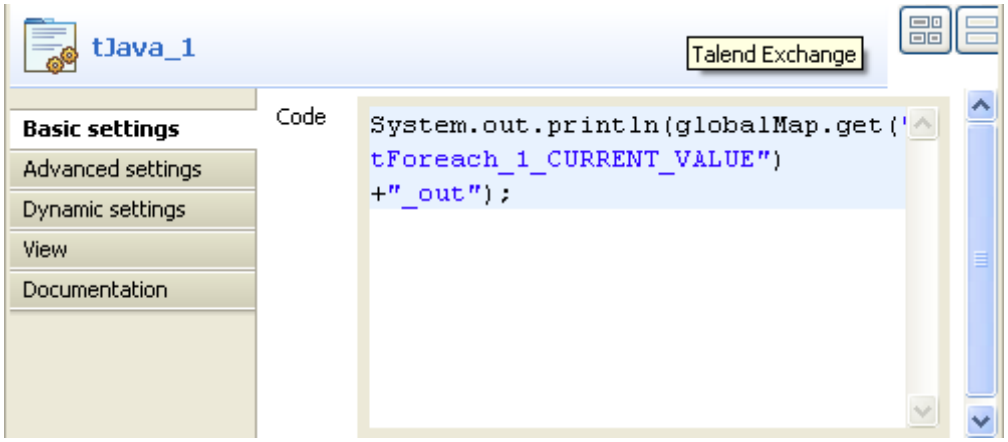


- Link **tForeach** to **tJava** using a **Row > Iterate** connection.
- Double-click **tForEach** to open its **Basic settings** view:



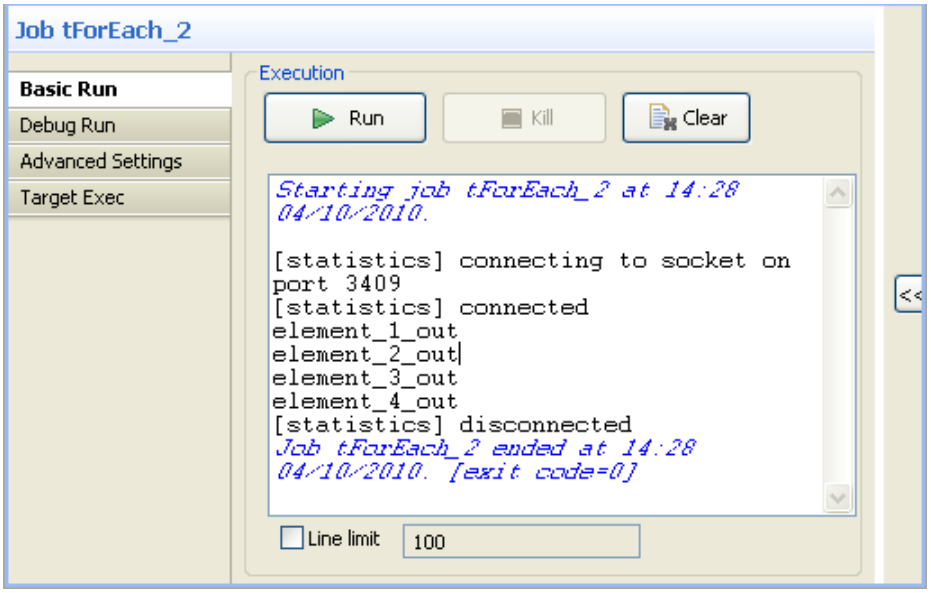
- Click the **[+]** button to add as many rows to the **Values** list as required.

- Click on the **Value** fields to enter the list values, between double quotation marks.
- Double-click **tJava** to open its **Basic settings** view:



- Enter the following Java code in the **Code** area: System.out.println(globalMap.get("tForeach_1_CURRENT_VALUE")+"_out");
- Save the Job and press **F6** to run it

  The **tJava** run view displays the list values retrieved from **tForeach**, each one suffixed with _out:



tInfiniteLoop

tInfiniteLoop Properties

Component Family	Orchestration	
Function	**tInfiniteLoop** runs an infiite loop on a task.	
Purpose	**tInfiniteLoop** allows you to to execute a task or a Job automatically, based on a loop.	
Basic settings	*Wait at each iteration (in milliseconds)*	Enter the time delay between iterations.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	**tInifniteLoop** is an input component and requires an Iterate link to connect it to the following component.	
Global Variables		**Current iteration**: Indicates the current iteration number. This is available as a **Flow** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Related scenario

For an example of the kind of scenario in which **tInifniteLoop** might be used, see section *Scenario: Job execution in a loop*, regarding the **tLoop** component.

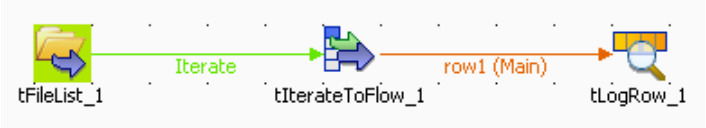
tIterateToFlow

tIterateToFlow Properties

Component family	Orchestration	
Function	**tIterateToFlow** transforms a list into a data flow that can be processed.	
Purpose	Allows you to transform non processable data into a processable flow.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of **tIterateToFlow**, the schema is to be defined
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Mapping	**Column**: Enter a name for the column to be created **Value**: Press **Ctrl+Space** to access all of the available variables, be they global or user-defined.
Advanced Settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This component is not startable (green background) and it requires an output component.	
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate; For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.

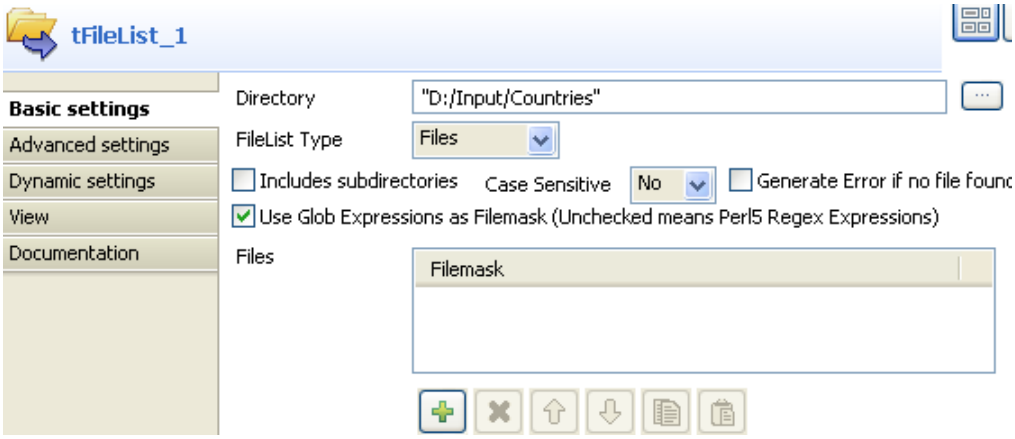
Scenario: Transforming a list of files as data flow

The following scenario describes a Job that iterates on a list of files, picks up the filename and current date and transforms this into a flow, that gets displayed on the console.

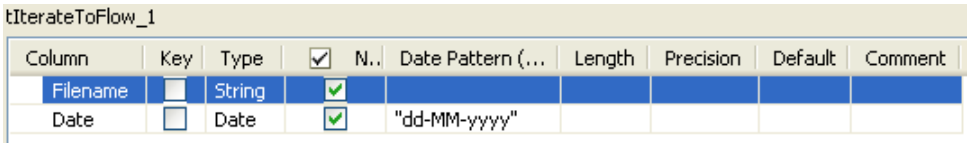


- Drop the following components: **tFileList**, **tIterateToFlow** and **tLogRow** from the **Palette** to the design workspace.

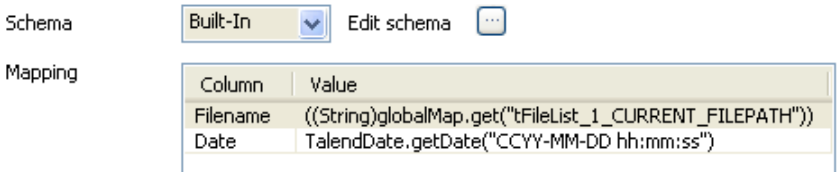
- Connect the **tFileList** to the **tIterateToFlow** using an **iterate** link and connect the Job to the **tLogRow** using a **Row main** connection.
- In the **tFileList Component** view, set the directory where the list of files is stored.



- In this example, the files are three simple .txt files held in one directory: *Countries*.
- No need to care about the case, hence clear the **Case sensitive** check box.
- Leave the **Include Subdirectories** check box unchecked.
- Then select the **tIterateToFlow** component et click **Edit Schema** to set the new schema



- Add two new columns: *Filename* of **String** type and *Date* of **date** type. Make sure you define the correct pattern in Java.
- Click **OK** to validate.
- Notice that the newly created schema shows on the **Mapping** table.



- In each cell of the **Value** field, press **Ctrl+Space bar** to access the list of global and user-specific variables.
- For the *Filename* column, use the global variable: `tFileList_1CURRENT_FILEPATH`. It retrieves the current filepath in order to catch the name of each file, the Job iterates on.
- For the *Date* column, use the Talend routine:`TalendDate.getCurrentDate()` (in Java)
- Then on the **tLogRow** component view, select the **Print values in cells of a table** check box.
- Save your Job and press **F6** to execute it.

```
Starting job tIterateToFlow at 10:28 10/03/2010.

[statistics] connecting to socket on port 3773
[statistics] connected
.----------------------------+--------------------.
|                    tLogRow_1                      |
|============================+====================|
|Filename                    |Date                |
|============================+====================|
|D:\Input\Countries\in-01.txt|2010-03-10 10:28:56|
|D:\Input\Countries\in-02.txt|2010-03-10 10:28:56|
|D:\Input\Countries\in-03.txt|2010-03-10 10:28:56|
'----------------------------+--------------------'

[statistics] disconnected
Job tIterateToFlow ended at 10:28 10/03/2010. [exit code=0]
```

The filepath displays on the *Filename* column and the current date displays on the *Date* column.

tLoop

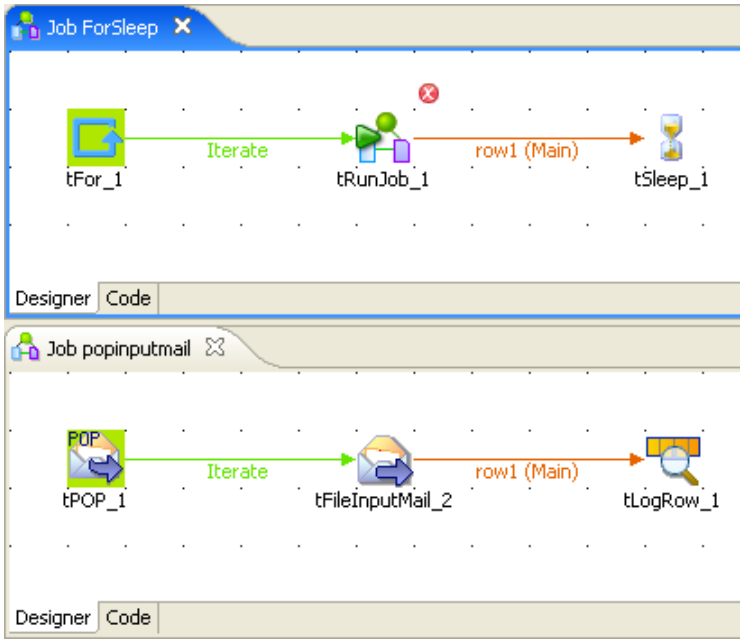
tLoop Properties

Component family	Orchestration	
Function	**tLoop** iterates on a task execution.	
Purpose	**tLoop** allows you to execute a task or a Job automatically, based on a loop	
Basic settings	*Loop Type*	Select a type of loop to be carried out: either **For** or **While**. **For**: The task or Job is carried out for the defined number of iteration **While**: The task or Job is carried until the condition is met.
For	*From*	Type in the first instance number which the loop should start from. A start instance number of 2 with a step of 2 means the loop takes on every even number instance.
	To	Type in the last instance number which the loop should finish with.
	Step	Type in the step the loop should be incremented of. A step of 2 means every second instance.
While	*Declaration*	Type in an expression initiating the loop.
	Condition	Type in the condition that should be met for the loop to end.
	Iteration	Type in the expression showing the operation to be performed at each loop.
	Values are increasing	Select this check box to only allow an increasing sequence. Deselect this check box to only allow a decreasing sequence.
Usage	**tLoop** is to be used as a start component and can only be used with an iterate connection to the next component.	
Global Variables		**Current value**: Indicates the current value. This is available as a **Flow** variable. Returns an integer. **Current iteration**: Indicates the number of the current iteration. This is available as a **Flow** variable Returns an integer. The CURRENT_VALUE variable is available only in case of a **For** type loop. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize.

		For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Scenario: Job execution in a loop

This scenario describes a Job composed of a parent Job and a child Job. The parent Job implements a loop which executes n times a child Job, with a pause between each execution.



- In the parent Job, drop a **tLoop**, a **tRunJob** and a **tSleep** component from the **Palette** to the design workspace.
- Connect the **tLoop** to the **tRunJob** using an **Iterate** connection.
- Then connect the **tRunJob** to a **tSleep** component using a **Row** connection.
- On the child Job, drop the following components: **tPOP**, **tFileInputMail** and **tLogRow** the same way.
- On the **Basic settings** panel of the **tLoop** component, type in the instance number to start from (*1*), the instance number to finish with (*5*) and the step (*1*)
- On the **Basic settings** panel of the **tRunJob** component, select the child Job in the list of stored Jobs offered. In this example: *popinputmail*
- Select the context if relevant. In this use case, the context is *default* with no variables stored.
- In the **tSleep Basic settings** panel, type in the time-off value in second. In this example, type in *3 seconds* in the **Pause** field.
- Then in the child Job, define the connection parameters to the pop server, on the **Basic settings** panel.
- In the **tFileInputMail Basic settings** panel, select a global variable as **File Name**, to collect the current file in the directory defined in the **tPOP** component. Press **Ctrl+Space bar** to access the variable list. In this example, the variable to be used is: `((String)globalMap.get("tPOP_1_CURRENT_FILEPATH"))`
- Define the **Schema**, for it to include the mail element to be processed, such as *author*, *topic*, *delivery date* and *number of lines*.

- In the **Mail Parts** table, type in the corresponding **Mail part** for each column defined in the schema. ex: *author* comes from the *From* part of the email file.
- Then connect the **tFileInputMail** to a **tLogRow** to check out the execution result on the **Run** view.
- Press **F6** to run the Job.

tPostjob

tPostjob Properties

Component family	Orchestration	
Function	**tPostjob** starts the execution of a postjob.	
Purpose	**tPostjob** triggers a task required after the execution of a Job	
Usage	**tPostjob** is a start component and can only be used with an iterate connection to the next component.	
Connections		Outgoing links (from one component to another): **Trigger**: On Component Ok. Incoming links (from one component to another): **Trigger**: Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

For more information about the **tPostjob** component, see *Talend Open Studio for Big Data* **User Guide**.

Related scenario

No scenario is available for this component yet.

tPrejob

tPrejob Properties

Component family	Orchestration	
Function	**tPrejob** starts the execution of a prejob.	
Purpose	**tPrejob** triggers a task required for the execution of a Job	
Usage	**tPrejob** is a start component and can only be used with an iterate connection to the next component.	
Connections		Outgoing links (from one component to another): **Trigger**: On Component Ok.. Incoming links (from one component to another): **Trigger**: Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

For more information about the **tPrejob** component, see *Talend Open Studio for Big Data* **User Guide**.

Related scenario

No scenario is available for this component yet.

tReplicate

tReplicate Properties

Component family	Orchestration	
Function	Duplicate the incoming schema into two identical output flows.	
Purpose	Allows you to perform different operations on the same schema.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit schema** to make changes to the schema. Note that if you make changes to a remote schema, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data User Guide*.
Usage	This component is not startable (green background), it requires an Input component and an output component.	
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**: Main; Reject; For further information regarding connections, see *Talend Open Studio for Big Data User Guide.*

Related scenario

For a use case showing this component in use, see section *tReplaceList*.

tRunJob

tRunJob belongs to two component families: System and Orchestration. For more information on **tRunJob**, see section *tRunJob*.

tSleep

tSleep Properties

Component family	Orchestration	
Function	**tSleep** implements a time off in a Job execution.	
Purpose	Allows you to identify possible bottlenecks using a time break in the Job for testing or tracking purpose. In production, it can be used for any needed pause in the Job to feed input flow for example.	
Basic settings	*Pause (in second)*	Time in second the Job execution is stopped for.
Usage	**tSleep** component is generally used as a middle component to make a break/pause in the Job, before resuming the Job.	
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Related scenarios

For use cases in relation with **tSleep**, see section *Scenario: Job execution in a loop*.

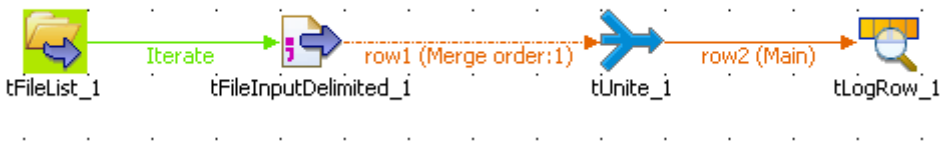
tUnite

tUnite Properties

Component family	Orchestration	
Function	Merges data from various sources, based on a common schema. ***tUnite*** *cannot exist in a data flow loop. For instance, if a data flow goes through several* ***tMap*** *components to generate two flows, they cannot be fed to* ***tUnite****.*	
Purpose	Centralize data from various and heterogeneous sources.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e. it defines the number of fields that will be processed and passed on to the next component. For *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is not startable and requires one or several input components and an output component.	
Global Variables		**Number of lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: Run if; On Component Ok; On Component Error Incoming links (from one component to another): **Row**: Main; Reject. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Iterate on files and merge the content

The following Job iterates on a list of files then merges their content and displays the final 2-column content on the console.

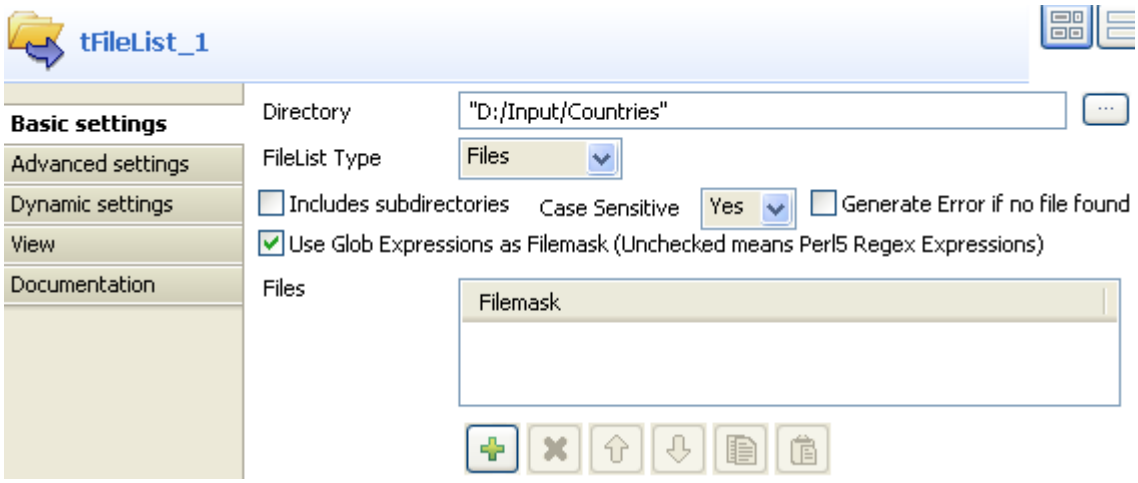


Dropping and linking the components

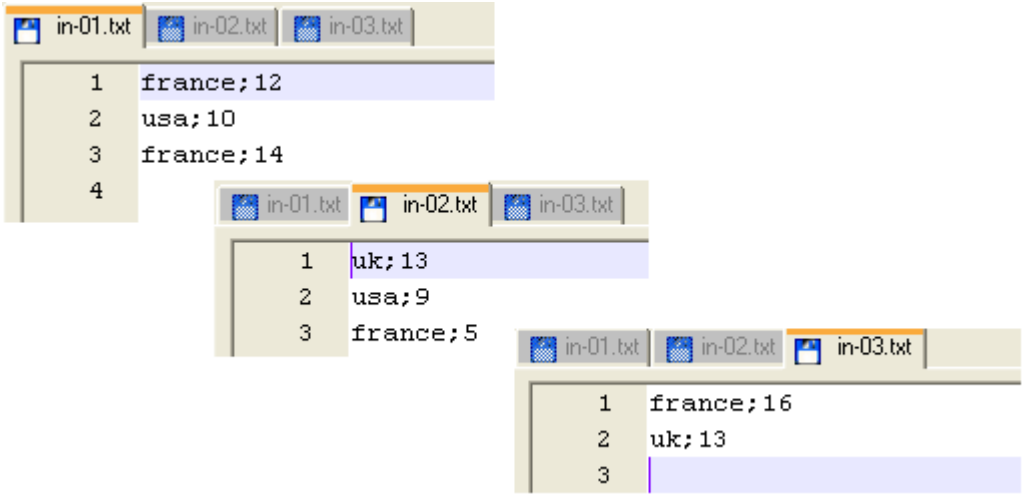
1. Drop the following components onto the design workspace: **tFileList**, **tFileInputDelimited**, **tUnite** and **tLogRow**.
2. Connect the **tFileList** to the **tFileInputDelimited** using an **Iterate** connection and connect the other component using a **row main** link.

Configuring the components

1. In the **tFileList Basic settings** view, browse to the directory, where the files to merge are stored.

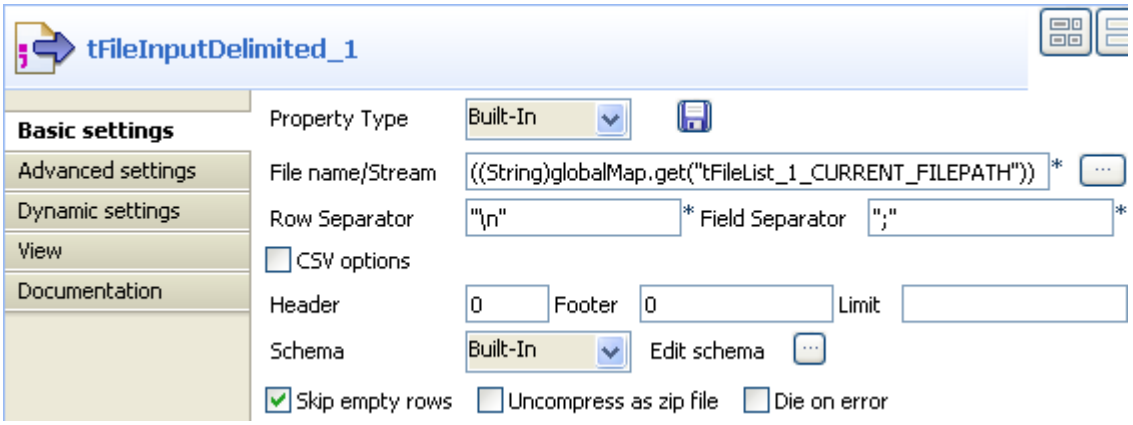


The files are pretty basic and contain a list of countries and their respective score.



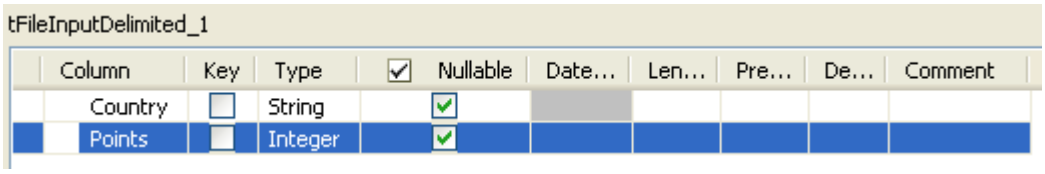
2. In the **Case Sensitive** field, select **Yes** to consider the letter case.

3. Select the **tFileInputDelimited** component, and display this component's **Basic settings** view.



4. Fill in the **File Name/Stream** field by using the **Ctrl+Space bar** combination to access the variable completion list, and selecting `tFileList.CURRENT_FILEPATH` from the global variable list to process all files from the directory defined in the **tFileList**.

5. Click the **Edit Schema** button and set manually the 2-column schema to reflect the input files' content.



For this example, the 2 columns are *Country* and *Points*. They are both **nullable**. The *Country* column is of **String** type and the *Points* column is of **Integer** type.

6. Click **OK** to validate the setting and accept to propagate the schema throughout the Job.

7. Then select the **tUnite** component and display the **Component** view. Notice that the output schema strictly reflects the input schema and is read-only.

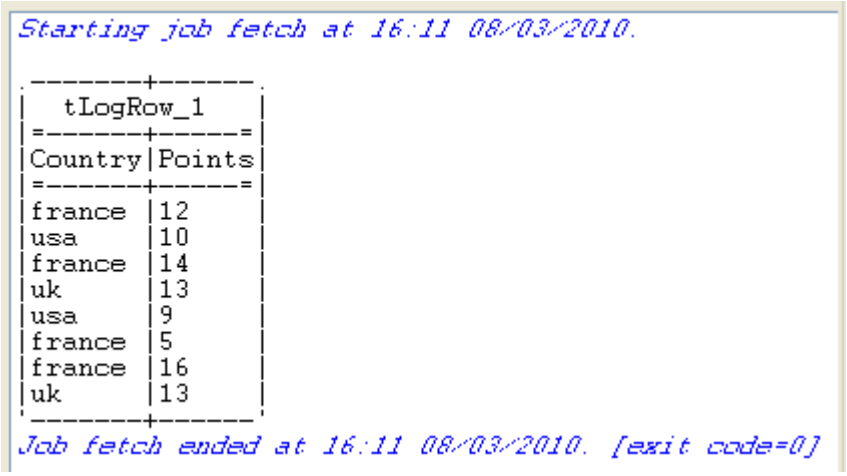
8. In the **Basic settings** view of **tLogRow**, select the **Table** option to display properly the output values.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6**, or click **Run** on the **Run** console to execute the Job.

The console shows the data from the various files, merged into one single table.


Starting job fetch at 16:11 08/03/2010.
tLogRow_1
Country|Points
france |12
usa |10
france |14
uk |13
usa |9
france |5
france |16
uk |13
Job fetch ended at 16:11 08/03/2010. [exit code=0]

tWaitForFile

tWaitForFile properties

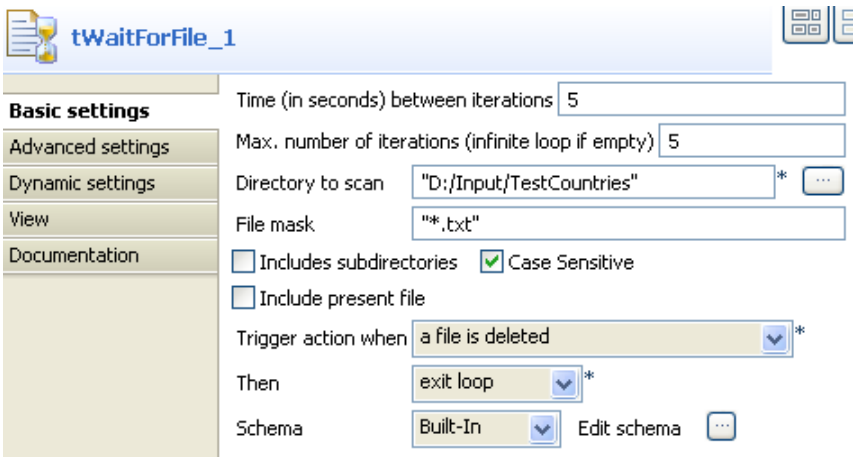
Component family	Orchestration	
Function	**tWaitForFile** component iterates on a given folder for file insertion or deletion then triggers a subjob to be executed when the condition is met.	
Purpose	This component allows a subjob to be triggered given a condition linked to file presence or removal.	
Basic settings	*Time (in seconds) between iterations*	Set the time interval in seconds between each check for the file.
	Max. number of iterations (infinite loop if empty)	Number of checks for file before the jobs times out.
	Directory to scan	Name of the folder to be checked for insert or removal
	File mask	Mask of the file to be searched for insertion or removal.
	Include subdirectories	Select this check box to include the sub-folders.
	Case sensitive	Select this check box to activate case sensitivity.
	Include present file	Select this check box to include the file in use.
	Trigger action when	Select the condition to be met for the action to be carried out: **A file is created A file is deleted A file is updated A file is created or updated or deleted**.
	Then	Select the action to be carried out: either stop the iterations when the condition is met **(exit loop)** or continue the loop until the end of the max iteration number **(continue loop)**.
	Schema and Edit Schema	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. .
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
Advanced Settings	*Wait for file to be released*	Select this check box so that the subjob only triggers after the file insertion/update/removal operation is complete. In case the operation is incomplete, the subjob will not trigger.
Usage	This component plays the role of the start (or trigger) component of the subjob which gets executed under the condition described. Therefore this component requires a subjob to be connected to via an Iterate link.	
Global Variables		**Current iteration**: Indicates the number of the current iteration. This is available as a **Flow** variable. Returns an integer. **Present File**: Indicates the name of the current file in the iteration which activated the trigger. This is available as a **Flow** variable. Returns a string.

		Deleted File: Indicates the path and name of the deleted file, which activated the trigger. This is available as a **Flow** variable Returns a string. **Created File Name**: Indicates the name and path to a newly created file which activated the trigger. This is available as a **Flow** variable. Returns a string. **Updated File**: Indicates the name and path to a file which has been updated, thereby activating the trigger. This is available as a **Flow** variable. Returns a string. **File Name**: Indicates the name of a file which has been created, deleted or updated, thereby activating the trigger. This is available as a **Flow** variable. Returns a string. **Not Updated File Name**: Indicates the names of files which have not been updated, thereby activating the trigger. This is available as a **Flow** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Main; Iterate. **Trigger**: On Subjob Ok; Run if; On Component Ok; On Component Error Incoming links (from one component to another): **Row**:Iterate. **Trigger**: On Subjob Ok; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Scenario: Waiting for a file to be removed

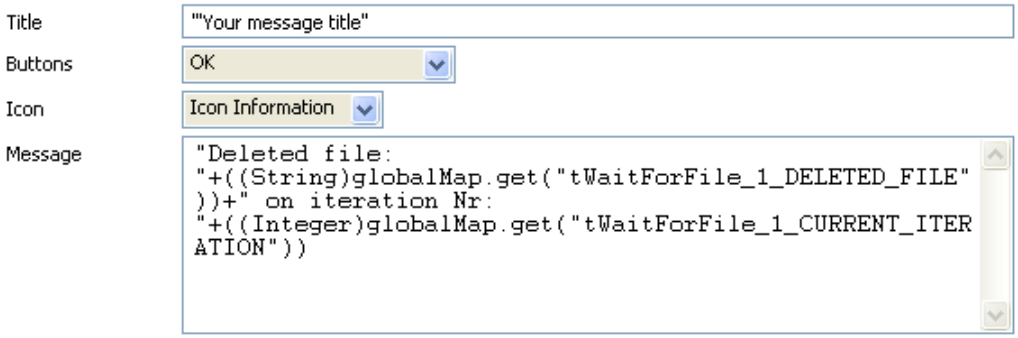
This scenario describes a Job scanning a directory and waiting for a file to be removed from this directory, in order for a subjob to be executed. When the condition of file removal is met, then the subjob simply displays a message box showing the file being removed.

- This use case only requires two components from the **Palette**: **tWaitForFile** and **tMsgbox**
- Click and place these components on the design workspace and connect them using an **Iterate** link to implement the loop.
- Then select the **tWaitForFile** component, and on the **Basic Settings** view of the **Component** tab, set the condition and loop properties:

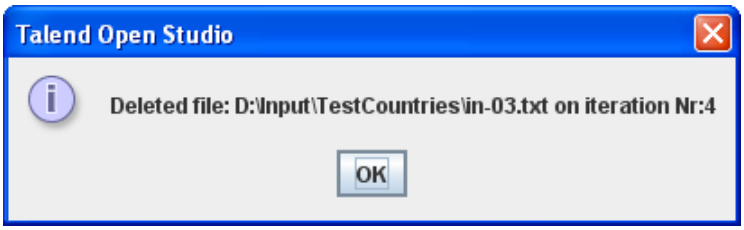


- In the **Time (in seconds) between iteration** field, set the time in seconds you want to wait before the next iteration starts. In this example, the directory will be scanned every 5 seconds.
- In the **Max. number of iterations (infinite loop if empty)** field, fill out the number of iterations max you want to have before the whole Job is forced to end. In this example, the directory will be scanned a maximum of 5 times.
- In the **Directory to scan** field, type in the path to the folder to scan.
- In the **Trigger action when** field, select the condition to be met, for the subjob to be triggered. In this use case, the condition is a file is deleted (or moved) from the directory.
- In the **Then** field, select the action to be carried out when the condition is met before the number of iteration defined is reached. In this use case, as soon as the condition is met, the loop should be ended.
- Then set the subjob to be executed when the condition set is met. In this use case, the subjob simply displays a message box.
- Select the **tMsgBox** component, and on the **Basic Setting** view of the **Component** tab, set the message to be displayed.
- Fill out the **Title** and **Message** fields.
- Select the type of **Buttons** and the **Icon**
- In the **Message** field, you can write any type of message you want to display and use global variables available in the auto-completion list via **Ctrl+Space** combination.
- The message is:

```
"Deleted file: "+((String)globalMap.get("tWaitForFile_1_DELETED_FILE"))+"
on iteration Nr:"+((Integer)globalMap.get("tWaitForFile_1_CURRENT_ITERATION"))
```



Then execute the Job via the **F6** key. While the loop is executing, remove a file from the location defined. The message pops up and shows the defined message.



tWaitForSocket

tWaitForSocket properties

Component Family	Orchestration	
Function	**tWaitForSocket** component makes a loop on a defined port, to look for data, and triggers a subjob when the condition is met.	
Purpose	This component triggers a Job based on a defined condition.	
Basic settings	*Port*	DB server listening port.
	End of line separator	Enter the end of line separator to be used..
	Then	Select the action to be carried out: **keep on listening** or **close socket**
	Print client/server data	Select this check box to display the client or server data.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect the log data at a component level.
Usage	This is an input, trigger component for the subjob executed depending on the condition set. Hence, it needs to be connected to a subjob via an Iterate link.	
Global Variables		**Client input data**: Returns the data transmitted by the client. This is available as a **Flow** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error. Incoming links (from one component to another): **Row**:Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Related scenario

No scenario is available for this component yet.

tWaitForSqlData

tWaitForSqlData properties

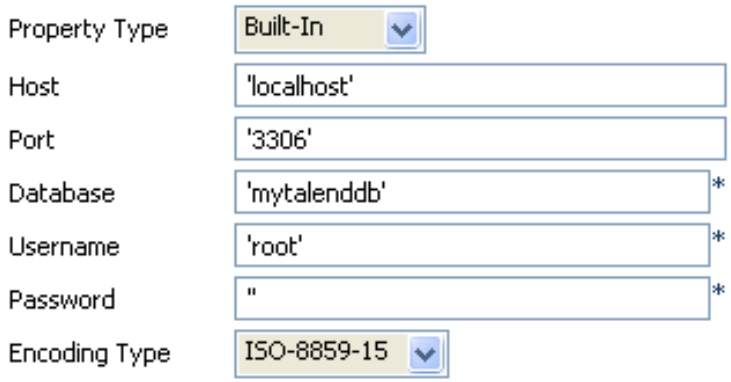
Component family	Orchestration	
Function	**tWaitForSqlData** component iterates on a given connection for insertion or deletion of rows and triggers a subjob to be executed when the condition is met.	
Purpose	This component allows a subjob to be triggered given a condition linked to sql data presence.	
Basic settings	*Wait at each iteration (in seconds)*	Set the time interval in seconds between each check for the sql data.
	Max. iterations (infinite if empty)	Number of checks for sql data before the Jobs times out.
	Use an existing connection/ Component List	A connection needs to be open to allow the loop to check for sql data on the defined DB. When a Job contains the parent Job and the child Job, **Component list** presents only the connection components in the same Job level, so if you need to use an existing connection from the other level, you can From the available database connection component in the level where the current component is, select the **Use or register a shared DB connection** check box. For more information about this check box, see *Databases - traditional components*, *Databases - appliance/datawarehouse components*, or *Databases - other components* for the connection components according to the database you are using. Otherwise, still in the level of the current component, deactivate the connection components and use **Dynamic settings** of the component to specify the intended connection manually. In this case, make sure the connection name is unique and distinctive all over through the two Job levels. For more information about **Dynamic settings**, see your studio user guide.
	Table to scan	Name of the table to be checked for insert or deletion
	Trigger action when rowcount is	Select the condition to be met for the action to be carried out: **Equal to Not Equal to Greater than Lower than Greater or equal to Lower or equal to**
	Value	Define the value to take into account.
	Then	Select the action to be carried out: either stop the iterations when the condition is met **(exit loop)** or continue the loop until the end of the max iteration number **(continue loop)**.
Usage	Although this component requires a Connection component to open the DB access, it plays also the role of the start (or trigger) component of the subjob which gets executed under the condition described. Therefore this component requires a subjob to be connected to via an Iterate link.	
Global Variables		**Current iteration**: Returns the number of the current iteration. This is available as a **Flow** variable. Returns an integer.

		Row count: Indicates the number of records detected in the table. This is available as a **Flow** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Scenario: Waiting for insertion of rows in a table

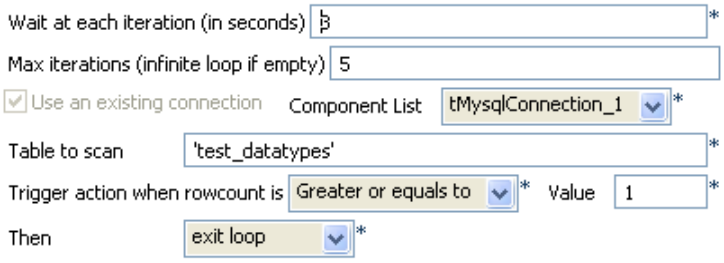
This scenario describes a Job reading a DB table and waiting for data to be put in this table in order for a subjob to be executed. When the condition of the data insertion in the table is met, then the subjob performs a Select* on the table and simply displays the content of the inserted data onto the standard console.

- Drop the following components from the **Palette** onto the design workspace: **tMySqlConnection**, **tWaitForSqlData**, **tMysqlInput**, **tLogRow**.
- Connect the **tMysqlConnection** component to the **tWaitforSqlData** using an **OnSubjobOK** link, available on the right-click menu.
- Then connect the **tWaitForSqlData** component to the subjob using an **Iterate** link as no actual data is transferred in this part. Indeed, simply a loop is implemented by the **tWaitForSqlData** until the condition is met.
- On the subjob to be executed if the condition is met, a **tMysqlInput** is connected to the standard console component, **tLogRow**. As the connection passes on data, use a Row main link.
- Now, set the connection to the table to check at regular intervals. On the **Basic Settings** view of the **tMySqlConnection Component** tab, set the DB connection properties.

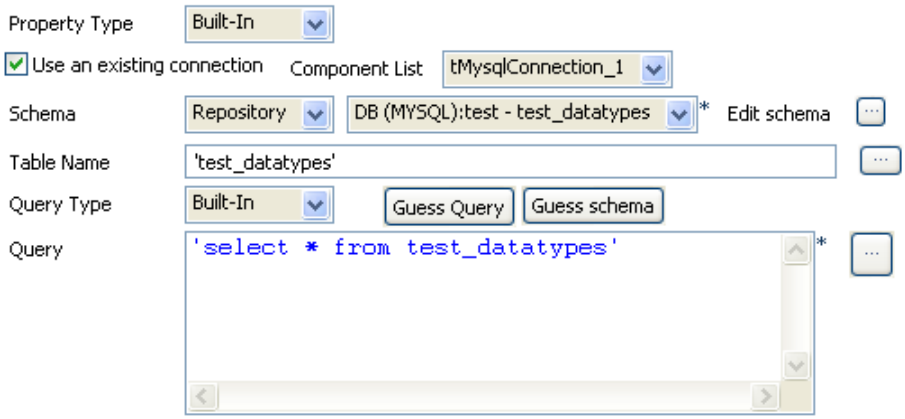


- Fill out the **Host**, **Port**, **Database**, **Username**, **Password** fields to open the connection to the Database table.
- Select the relevant **Encoding** if needed.
- Then select the **tWaitForSqlData** component, and on the **Basic Setting** view of the **Component** tab, set its properties.

- In the **Wait at each iteration** field, set the time in seconds you want to wait before the next iteration starts.

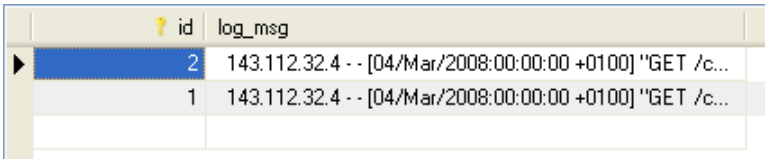


- In the **Max iterations** field, fill out the number of iterations max you want to have before the whole Job is forced to end.
- The **tWaitForSqlData** component requires a connection to be open in order to loop on the defined number of iteration. Select the relevant connection (if several) in the **Component List** combo box.
- In the **Table to scan** field, type in the name of the table in the DB to scan.In this example: *test_datatypes*.
- In the **Trigger action when rowcount is** and **Value** fields, select the condition to be met, for the subjob to be triggered. In this use case, the number of rows in the scanned table should *be greater or equal to 1*.
- In the **Then** field, select the action to be carried out when the condition is met before the number of iteration defined is reached. In this use case, as soon as the condition is met, the loop should be ended.
- Then set the subjob to be executed when the condition set is met. In this use case, the subjob simply selects the data from the scanned table and displays it on the console.
- Select the **tMySqlInput** component, and on the **Basic Setting** view of the **Component** tab, set the connection to the table.



- If the connection is set in the Repository, select the relevant entry on the list. Or alternatively, select the **Use an existing connection** check box and select the relevant connection component on the list.
- In this use case, the schema corresponding to the table structure is stored in the **Repository**.
- Fill out the **Table Name** field with the table the data is extracted from, *Test_datatypes*.
- Then in the **Query** field, type in the Select statement to extract the content from the table.
- No particular setting is required in the **tLogRow** component for this use case.

Then before executing the Job, make sure the table to scan (*test_datatypes*) is empty, in order for the condition (greater or equal to 1) to be met. Then execute the Job by pressing the **F6** key on your keyboard. Before the end of the iterating loop, feed the test_datatypes table with one or more rows in order to meet the condition.



id	log_msg
2	143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET /c...
1	143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET /c...

The Job ends when this table insert is detected during the loop, and the table content is thus displayed on the console.

```
Starting job tWaitForSql at 16:55 06/03/2008.
2| 143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET
/components/none HTTP/1.1" 404 354 "-" "Mozilla/4.0 (compatible
1| 143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET
/components/none HTTP/1.1" 404 354 "-" "Mozilla/4.0 (compatible
Job tWaitForSql ended at 16:55 06/03/2008. [exit code=0]
```

Processing components

This chapter details the main components that you can find in **Processing** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Processing family gathers together components that help you to perform all types of processing tasks on data flows, including aggregation, mapping, transformation, denormalizing, filtering and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAggregateRow

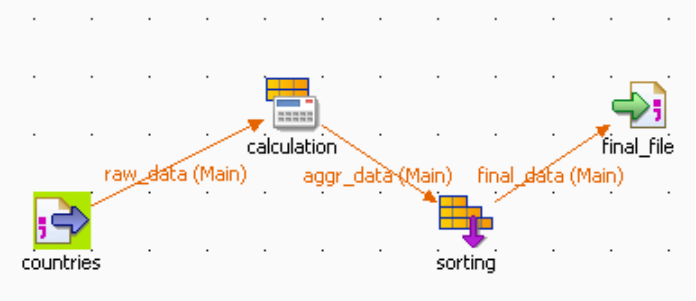
tAggregateRow properties

Component family	Processing	
Function	**tAggregateRow** receives a flow and aggregates it based on one or more columns. For each output line, are provided the aggregation key and the relevant result of set operations (min, max, sum...).	
Purpose	Helps to provide a set of metrics based on values or calculations.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered based on the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations. Ex: Select Country to calculate an average of values for each country of a list or select Country and Region if you want to compare one country's regions with another country' regions.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select the operator among: count, min, max, avg, sum, first, last, list, list(objects), count(distinct), standard deviation.
		Input column: Select the input column from which the values are taken to be aggregated.
		Ignore null values: Select the check boxes corresponding to the names of the columns for which you want the NULL value to be ignored.
Advanced settings	*Delimiter(only for list operation)*	Enter the delimiter you want to use to separate the different operations.
	Use financial precision, this is the max precision for "sum" and "avg" operations, checked option heaps more memory and slower than unchecked.	Select this check box to use a financial precision. This is a max precision but consumes more memory and slows the processing. *We advise you to use the BigDecimal type for the output in order to obtain precise results.*
	Check type overflow (slower)	Checks the type of data to ensure that the Job doesn't crash.
	Check ULP (Unit in the Last Place), ensure that a value will be incremented	Select this check box to ensure the most precise results possible for the Float and Double types.

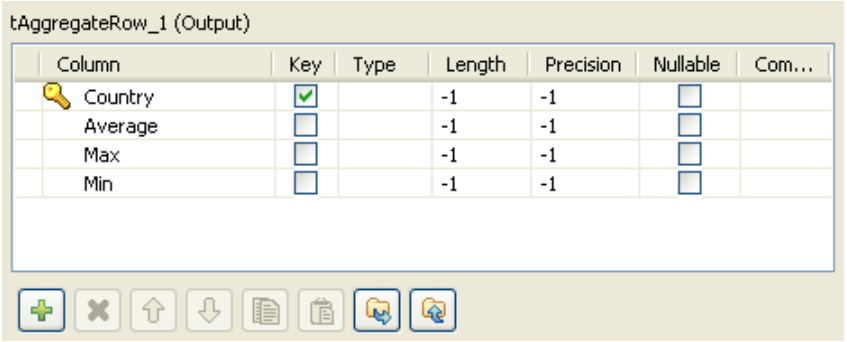
	or decremented correctly, only float and double types. (slower)	
	tStatCatcher Statistics	Check this box to collect the log data at component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step. Usually the use of **tAggregateRow** is combined with the **tSortRow** component	
Limitation	n/a	

Scenario 1: Aggregating values and sorting data

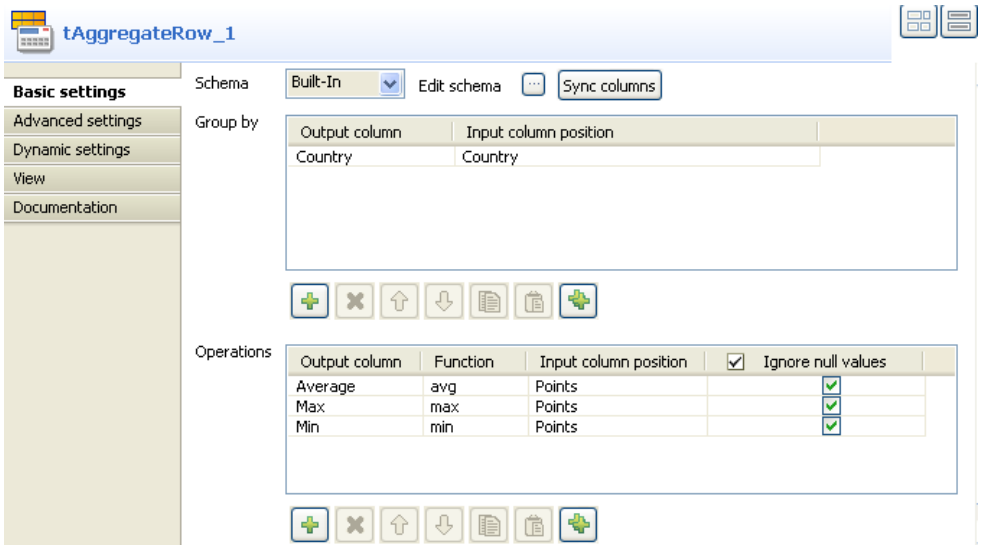
The following scenario describes a four-component Job. As input component, a CSV file contains countries and notation values to be sorted by best average value. This component is connected to a **tAggregateRow** operator, in charge of the average calculation then to a **tSortRow** component for the ascending sort. The output flow goes to the new csv file.



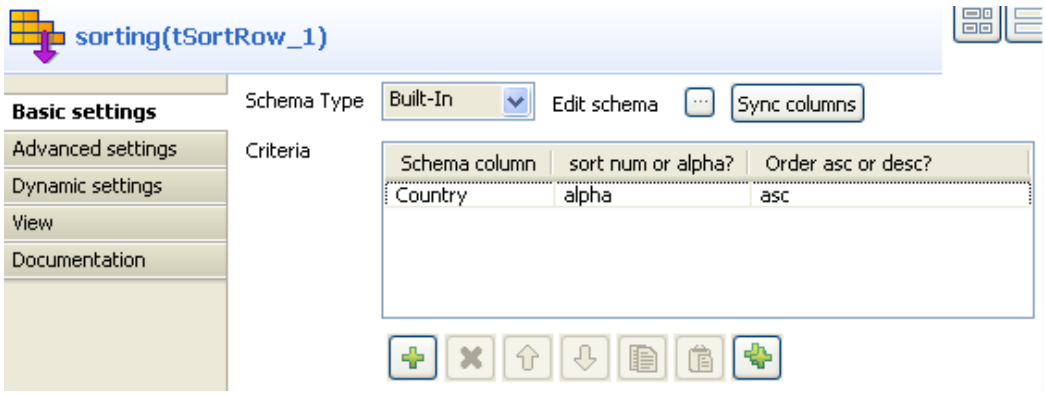
- From the **File** folder in the **Palette**, drop a **tFileInputDelimited** component to the design workspace.
- Click the label and rename it as *Countries*. Or rename it from the **View** tab panel
- In the **Basic settings** tab panel of this component, define the filepath and the delimitation criteria. Or select the metadata file in the repository if it exists.
- Click **Edit schema...** and set the columns: *Countries* and *Points* to match the file structure. If your file description is stored in the Metadata area of the Repository, the schema is automatically uploaded when you click **Repository** in **Schema type** field.
- Then from the **Processing** folder in the **Palette**, drop a **tAggregateRow** component to the design workspace. Rename it as *Calculation*.
- Connect *Countries* to *Calculation* via a right-click and select **Row > Main**.
- Double-click *Calculation* (**tAggregateRow** component) to set the properties. Click **Edit schema** and define the output schema. You can add as many columns as you need to hold the set operations results in the output flow.



- In this example, we'll calculate the average notation value per country and we will display the max and the min notation for each country, given that each country holds several notations. Click **OK** when the schema is complete.
- To carry out the various set operations, back in the **Basic settings** panel, define the sets holding the operations in the **Group By** area. In this example, select **Country** as group by column. Note that the output column needs to be defined a key field in the schema. The first column mentioned as output column in the **Group By** table is the main set of calculation. All other output sets will be secondary by order of display.
- Select the input column which the values will be taken from.
- Then fill in the various operations to be carried out. The functions are *average*, *min*, *max* for this use case. Select the input columns, where the values are taken from and select the check boxes in the **Ignore null values** list as needed.

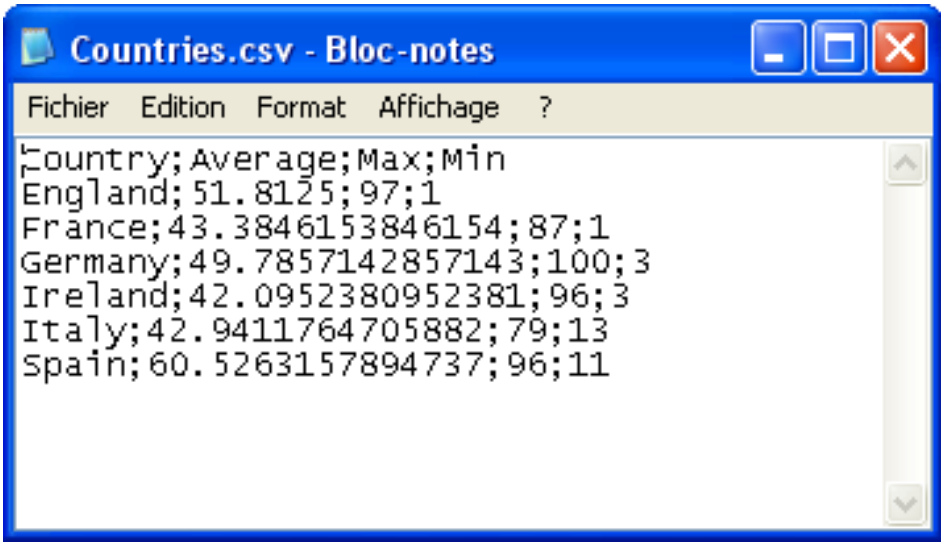


- Drop a **tSortRow** component from the **Palette** onto the design workspace. For more information regarding this component, see section *tSortRow properties*.
- Connect the **tAggregateRow** to this new component using a row main link.
- On the **Component** view of the **tSortRow** component, define the column the sorting is based on, the sorting type and order.



- In this case, the column to be sorted by is *Country*, the sort type is alphabetical and the order is ascending.
- Drop a **tFileOutputDelimited** from the **Palette** to the design workspace and define it to set the output flow.
- Connect the **tSortRow** component to this output component.

- In the **Component** view, enter the output filepath. Edit the schema if need be. In this case the delimited file is of csv type. And select the **Include Header** check box to reuse the schema column labels in your output flow.
- Press **F6** to execute the Job. The csv file thus created contains the aggregating result.



tAggregateSortedRow

tAggregateSortedRow properties

Component family	Processing	
Function	**tAggregateSortedRow** receives a sorted flow and aggregates it based on one or more columns. For each output line, are provided the aggregation key and the relevant result of set operations (min, max, sum...).	
Purpose	Helps to provide a set of metrics based on values or calculations. As the input flow is meant to be sorted already, the performance are hence greatly optimized.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Input rows count	Specify the number of rows that are sent to the **tAggregateSortedRow** component. If you specified a **Limit** for the number of rows to be processed in the input component, you will have to use that same limit in the **Input rows count** field.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered based on the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations. Ex: Select Country to calculate an average of values for each country of a list or select Country and Region if you want to compare one country's regions with another country' regions.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select the operator among: count, min, max, avg, first, last.
		Input column: Select the input column from which the values are taken to be aggregated.
		Ignore null values: Select the check boxes corresponding to the names of the columns for which you want the NULL value to be ignored.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Related scenario

For related use case, see section *Scenario 1: Aggregating values and sorting data*.

tConvertType

tConvertType properties

Component family	Processing	
Function	**tConvertType** allows specific conversions at run time from one **Talend** java type to another.	
Purpose	Helps to automatically convert one **Talend** java type to another and thus.avoid compiling errors.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema.
		Built-in: You create and store the schema locally for only the current component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Auto Cast	This check box is selected by default. It performs an automatic java type conversion.
	Manual Cast	This mode is not visible if the **Auto Cast** check box is selected. It allows you to precise manually the columns where a java type conversion is needed.
	Set empty values to Null before converting	This check box is selected to set the empty values of String or Object type to null for the input data.
	Die on error	This check box is selected to kill the Job when an error occurs.
Usage	This component cannot be used as a start component as it requires an input flow to operate.	
Limitation	n/a	

Scenario: Converting java types

This Java scenario describes a four-component Job where the **tConvertType** component is used to convert Java types in three columns, and a **tMap** is used to adapt the schema and have as an output the first of the three columns and the sum of the two others after conversion.

In this scenario, the input schemas for the input delimited file are stored in the repository, you can simply drag and drop the relevant file node from **Repository - Metadata - File delimited** onto the design workspace to automatically retrieve the **tFileInputDelimited** component's setting. For more information, see *Talend Open Studio for Big Data User Guide*.

Dropping the components

1. Drop the following components from the **Palette** onto the design workspace: **tConvertType**, **tMap**, and **tLogRow**.

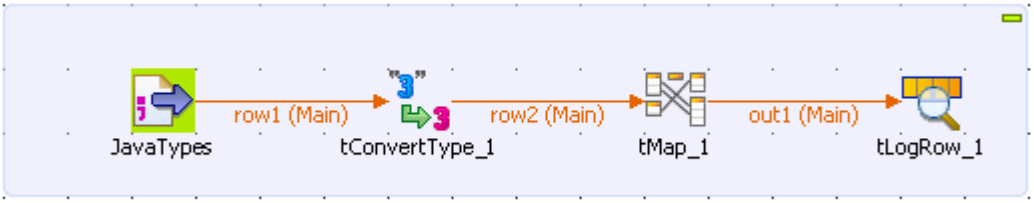
2. In the Repository tree view, expand **Metadata** and from **File delimited** drag the relevant node, *JavaTypes* in this scenario, to the design workspace.

   The **[Components]** dialog box displays.

3. From the component list, select **tFileInputDelimited** and click **Ok**.

   A **tFileInputComponent** called *Java types* displays in the design workspace.

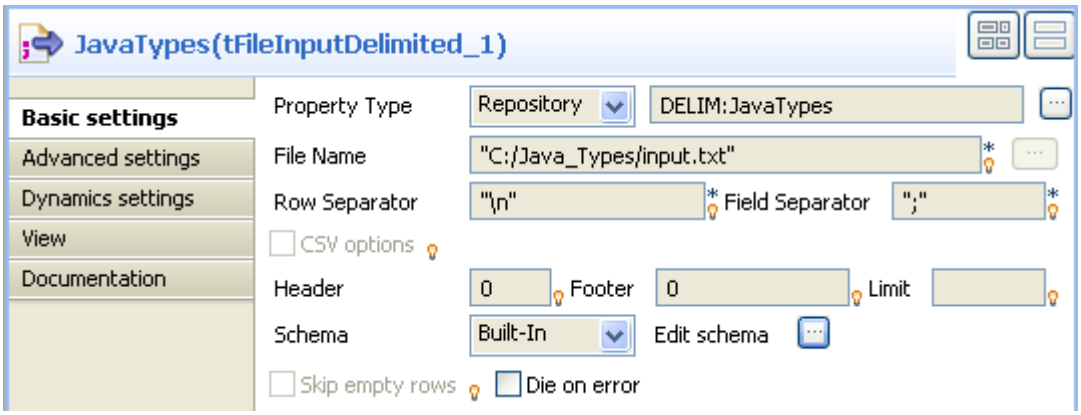
4. Connect the components using **Row > Main** links.



Configuring the components

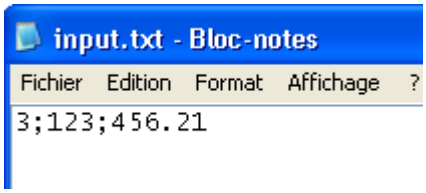
1. Double-click **tFileInputDelimited** to enter its **Basic settings** view.

2. Set **Property Type** to **Repository** since the file details are stored in the repository. The fields to follow are pre-defined using the fetched data.



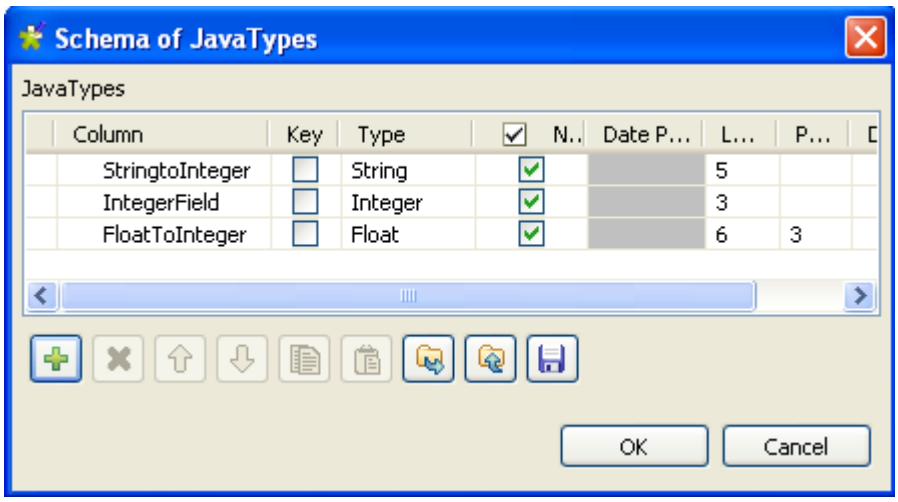
*For Talend Open Studio for Big Data, the **Property type** and **Schema** of components are always **Built-in**. For how to edit a built-in schema, see Talend Open Studio for Big Data User Guide.*

   The input file used in this scenario is called *input*. It is a text file that holds string, integer, and float java types.



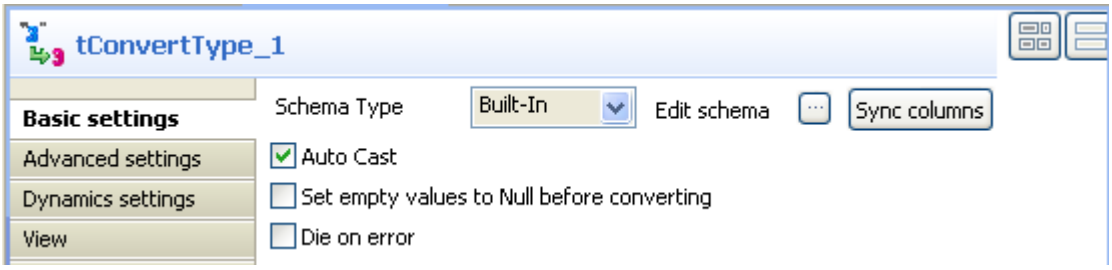
   Fill in all other fields as needed. For more information, see section *tMDMInput properties*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows.

3. Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of three columns, *StringtoInteger*, *IntegerField*, and *FloatToInteger*.



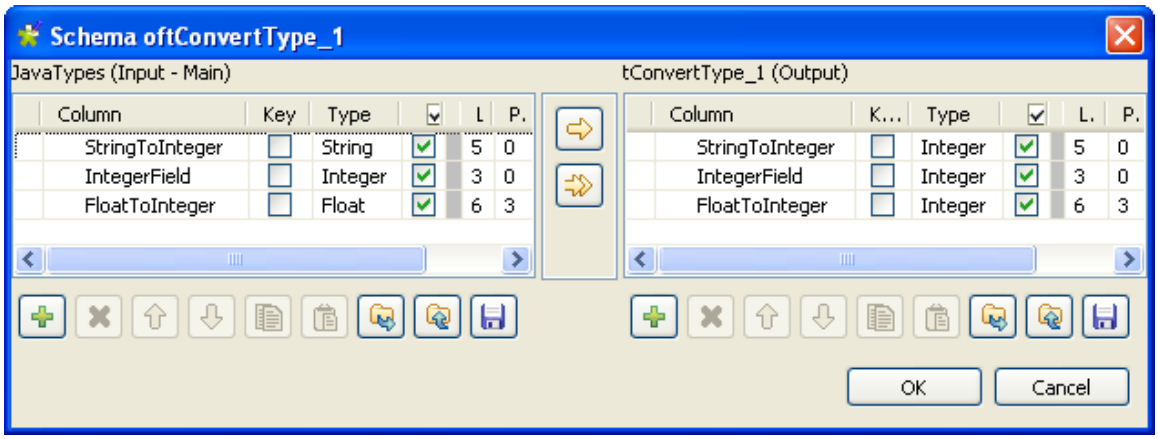
4. Click **Ok** to close the dialog box.

5. Double-click **tConvertType** to enter its **Basic settings** view.



6. Set **Schema Type** to **Built in**, and click **Sync columns** to automatically retrieve the columns from the **tFileInputDelimited** component.

7. Click **Edit schema** to describe manually the data structure of this processing component.

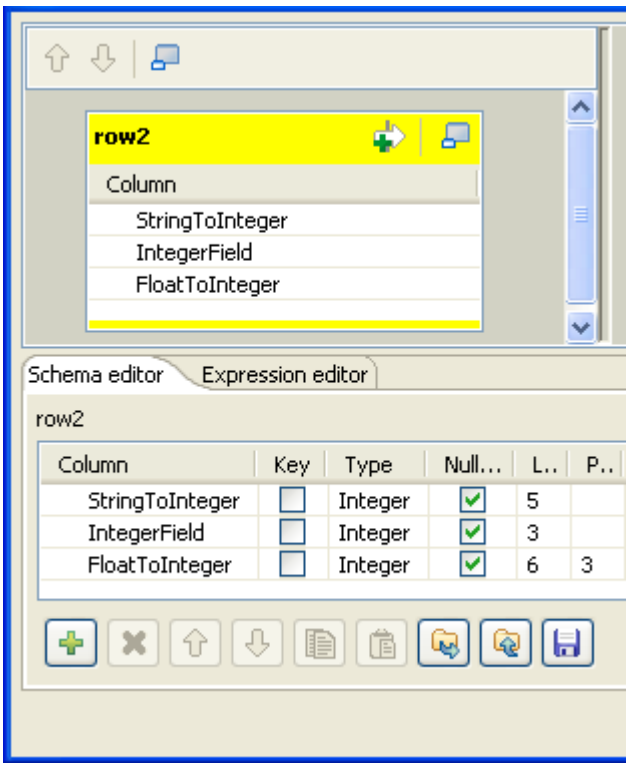


In this scenario, we want to convert a string type data into an integer type and a float type data into an integer type.

Click **OK** to close the **[Schema of tConvertType]** dialog box.

8. Double-click **tMap** to open the Map editor.

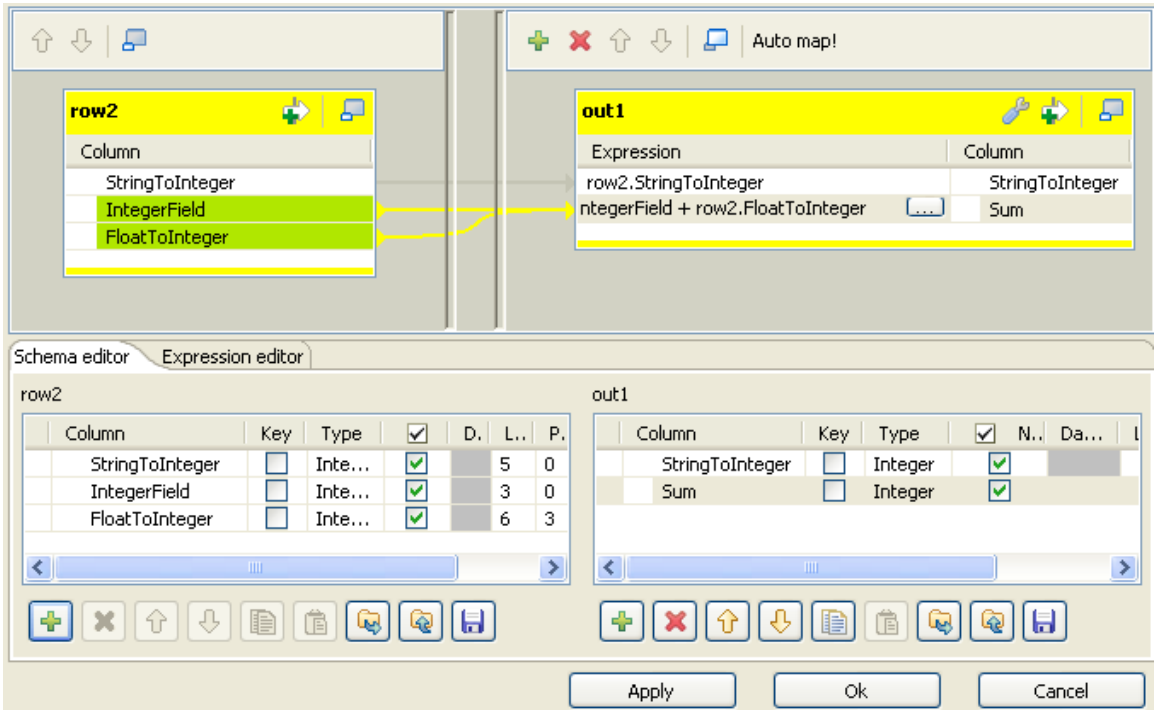
The Map editor displays the input metadata of the **tFileInputDelimited** component



9. In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and name them as *StringToInteger* and *Sum*.

10. In the Map editor, drag the *StringToInteger* row from the input table to the *StringToInteger* row in the output table.

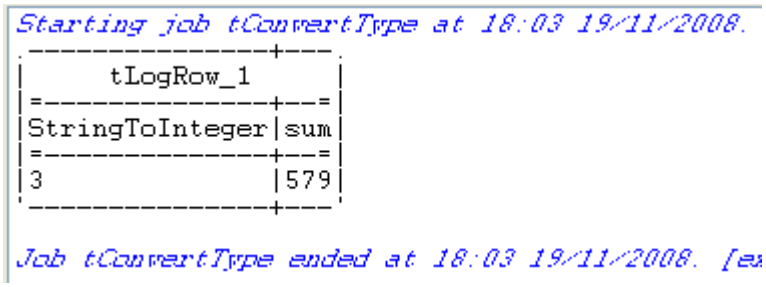
11. In the Map editor, drag each of the *IntegerField* and the *FloatToInteger* rows from the input table to the *Sum* row in the output table and click **OK** to close the Map editor.



12. In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*.

Executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6** to execute it.



The string type data is converted into an integer type and displayed in the *StringToInteger* column on the console. The float type data is converted into an integer and added to the *IntegerField* value to give the addition result in the *Sum* column on the console.

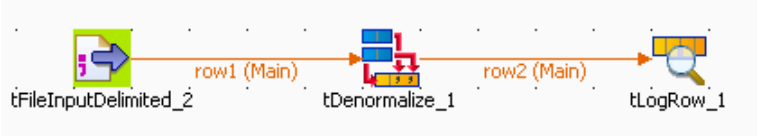
tDenormalize

tDenormalize Properties

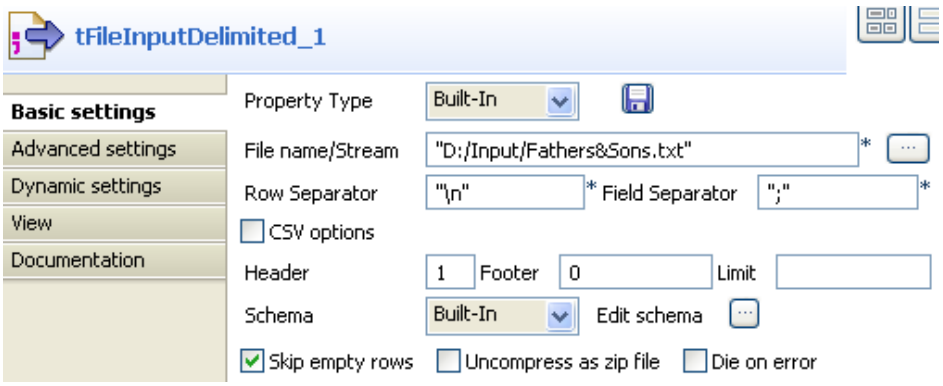
Component family	Processing/Fields	
Function	Denormalizes the input flow based on one column.	
Purpose	**tDenormalize** helps synthesize the input flow.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. In this component, the schema is read-only.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	To denormalize	In this table, define the parameters used to denormalize your columns. **Column**: Select the column to denormalize. **Delimiter**: Type in the separator you want to use to denormalize your data between double quotes. **Merge same value**: Select this check box to merge identical values.
Advanced settings		
	tStatCatcher Statistics	Select this ckeck box to collect the log data at component level.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	n/a	

Scenario 1: Denormalizing on one column

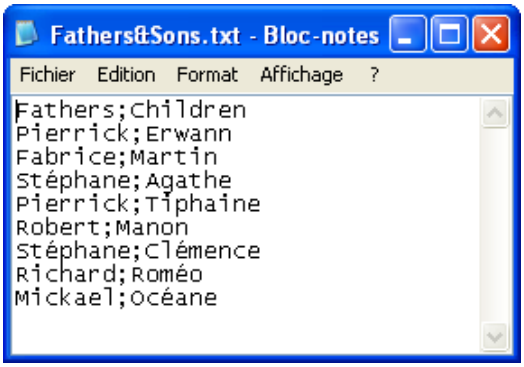
This scenario illustrates a Job denormalizing one column in a delimited file.



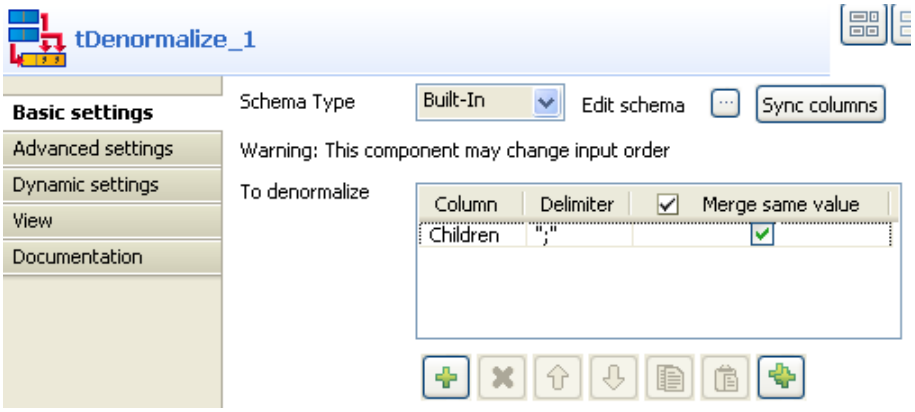
- Drop the following components: **tFileInputDelimited**, **tDenormalize**, **tLogRow** from the **Palette** to the design workspace.
- Connect the components using **Row main** connections.
- On the **tFileInputDelimited Component** view, set the filepath to the file to be denormalized.



- Define the **Header**, **Row Separator** and **Field Separator** parameters.
- The input file schema is made of two columns, *Fathers* and *Children*.



- In the **Basic settings** of **tDenormalize**, define the column that contains multiple values to be grouped.
- In this use case, the column to denormalize is *Children*.



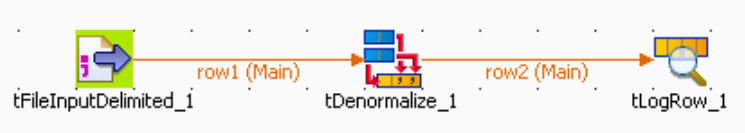
- Set the **Delimiter** to separate the grouped values. Beware as only one column can be denormalized.
- Select the **Merge same value** check box, if you know that some values to be grouped are strictly identical.
- Save your Job and press **F6** to execute it.

```
Starting job tDenormalize at 10:50 09/03/2016.

Mickael|Océane
Stéphane|Agathe;Clémence
Pierrick|Erwann;Tiphaine
Robert|Manon
Richard|Roméo
Fabrice|Martin
Job tDenormalize ended at 10:50 09/03/2016. [exit code=0]
```

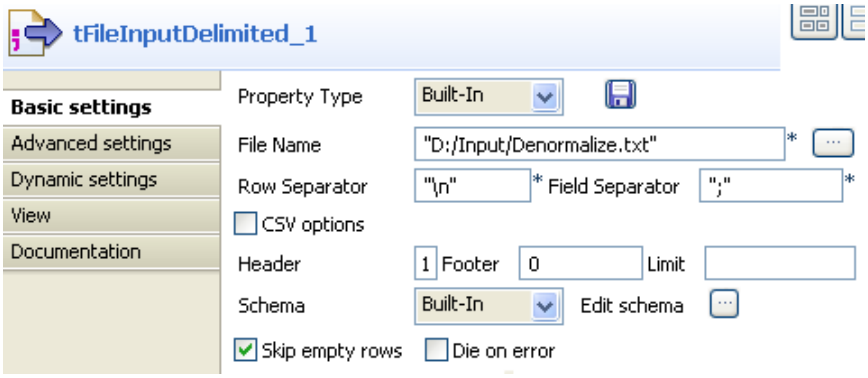
All values from the column *Children* (set as column to denormalize) are grouped by their *Fathers* column. Values are separated by a comma.

Scenario 2: Denormalizing on multiple columns

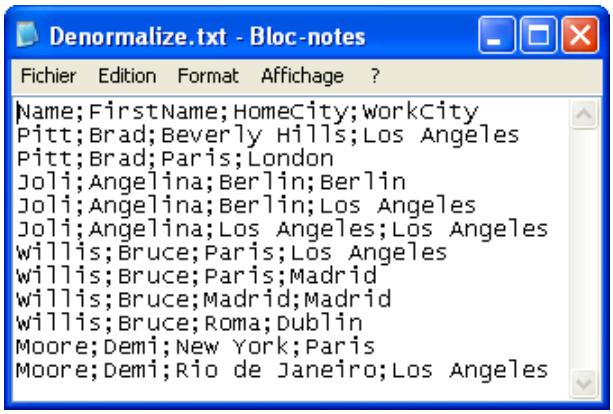
This scenario illustrates a Job denormalizing two columns from a delimited file.



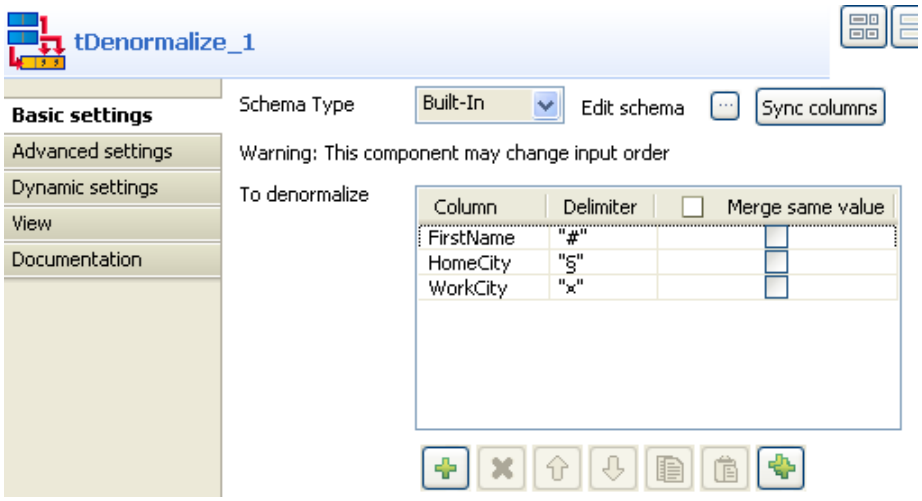
- Drop the following components: **tFileInputDelimited**, **tDenormalize**, **tLogRow** from the **Palette** to the design workspace.
- Connect all components using a **Row main** connection.
- On the **tFileInputDelimited Basic settings** panel, set the filepath to the file to be denormalized.



- Define the **Row** and **Field separators**, the **Header** and other information if required.
- The file schema is made of four columns including: **Name**, **FirstName**, **HomeTown**, **WorkTown**.



- In the **tDenormalize** component **Basic settings**, select the columns that contain the repetition. These are the column which are meant to occur multiple times in the document. In this use case, *FirstName*, *HomeCity* and *WorkCity* are the columns against which the denormalization is performed.
- Add as many line to the table as you need using the plus button. Then select the relevant columns in the drop-down list.



- In the **Delimiter** column, define the separator between double quotes, to split concanated values. For *FirstName* column, type in "#", for *HomeCity*, type in "§", ans for *WorkCity*, type in "¤".
- Save your Job and press **F6** to execute it.

```
Starting job tDenormalize2 at 12:03 09/03/2010.

Moore|Demi#Demi|New York§Rio de Janeiro|Paris¤Los Angeles
Joli|Angelina#Angelina#Angelina|Berlin§Berlin§Los
Angeles|Berlin¤Los Angeles¤Los Angeles
Pitt|Brad#Brad|Beverly Hills§Paris|Los Angeles¤London
Willis|Bruce#Bruce#Bruce#Bruce|Paris§Paris§Madrid§Roma|Los
Angeles¤Madrid¤Madrid¤Dublin
Job tDenormalize2 ended at 12:03 09/03/2010. [exit code=0]
```

- The result shows the denormalized values concatenated using a comma.
- Back to the **tDenormalize** components **Basic settings**, in the To denormalize table, select the **Merge same value** check box to remove the duplicate occurrences.
- Save your Job again and press **F6** to execute it.

```
Starting job tDenormalize2 at 13:44 09/03/2010.

Moore|Demi|New York§Rio de Janeiro|Paris¤Los Angeles
Joli|Angelina|Berlin§Los Angeles|Berlin¤Los Angeles
Pitt|Brad|Beverly Hills§Paris|Los Angeles¤London
Willis|Bruce|Paris§Madrid§Roma|Los Angeles¤Madrid¤Dublin
Job tDenormalize2 ended at 13:44 09/03/2010. [exit code=0]
```

This time, the console shows the results with no duplicate instances.

tDenormalizeSortedRow

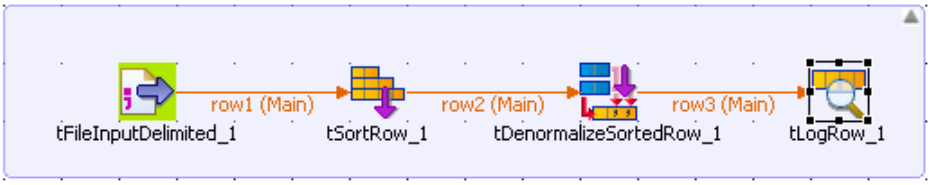
tDenormalizeSortedRow properties

Component family	Processing/Fields	
Function	**tDenormalizeSortedRow** combines in a group all input sorted rows. Distinct values of the denormalized sorted row are joined with item separators.	
Purpose	**tDenormalizeSortedRow** helps synthesizing sorted input flow to save memory.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Input rows count	Enter the number of input rows.
	To denormalize	Enter the name of the column to denormalize.
Usage	This component handles flows of data therefore it requires input and output components.	
Limitation	n/a	

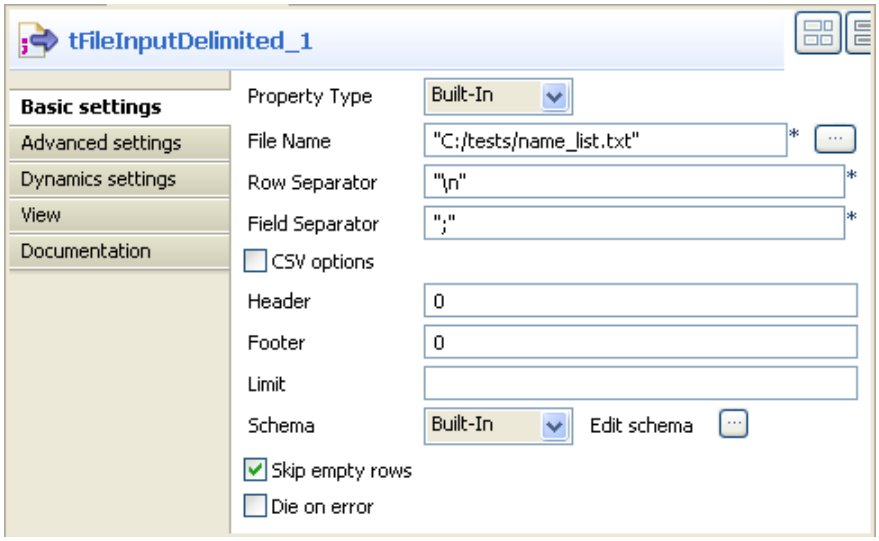
Scenario: Regrouping sorted rows

This Java scenario describes a four-component Job. It aims at reading a given delimited file row by row, sorting input data by sort type and order, denormalizing all input sorted rows and displaying the output on the **Run** log console.

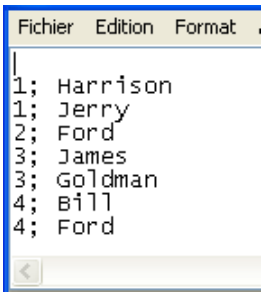
- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tSortRow**, **tDenormalizeSortedRow**, and **tLogRow**.
- Connect the four components using **Row Main** links.



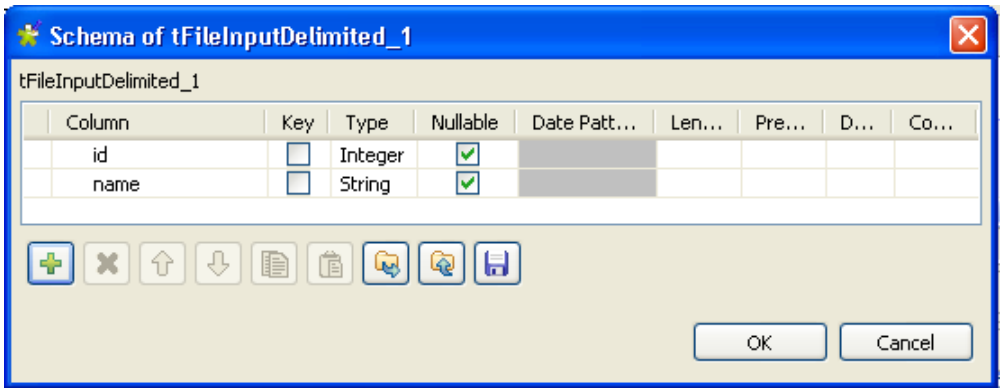
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.



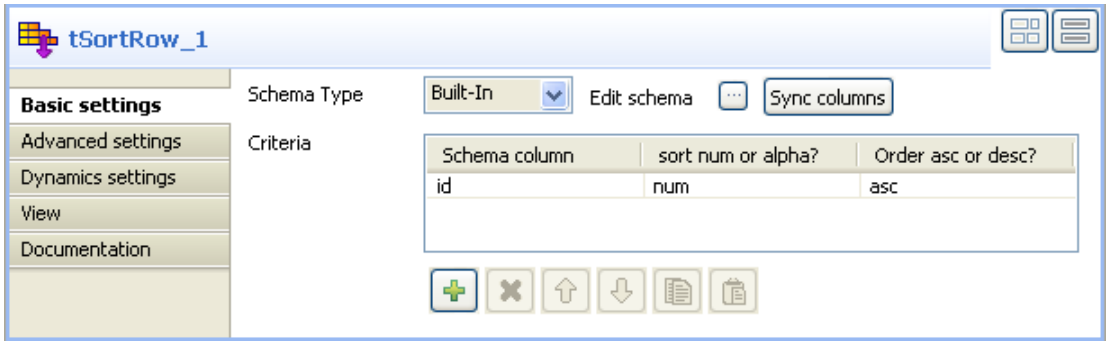
- Set **Property Type** to **Built-In**.
- Fill in a path to the processed file in the **File Name** field. The *name_list* file used in this example holds two columns, id and first name.



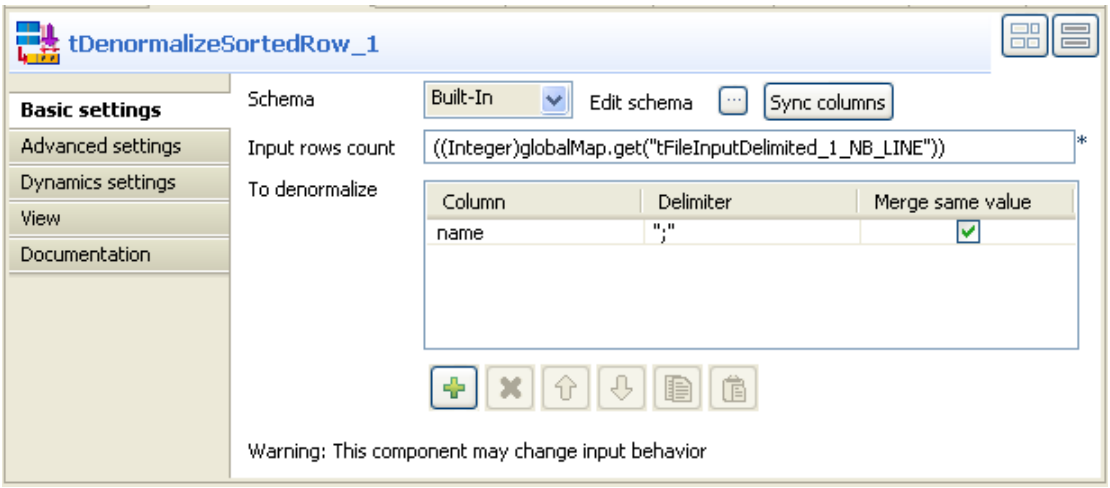
- If needed, define row and field separators, header and footer, and the number of processed rows.
- Set **Schema** to **Built in** and click the three-dot button next to **Edit Schema** to define the data to pass on to the next component. The schema in this example consists of two columns, *id* and *name*.



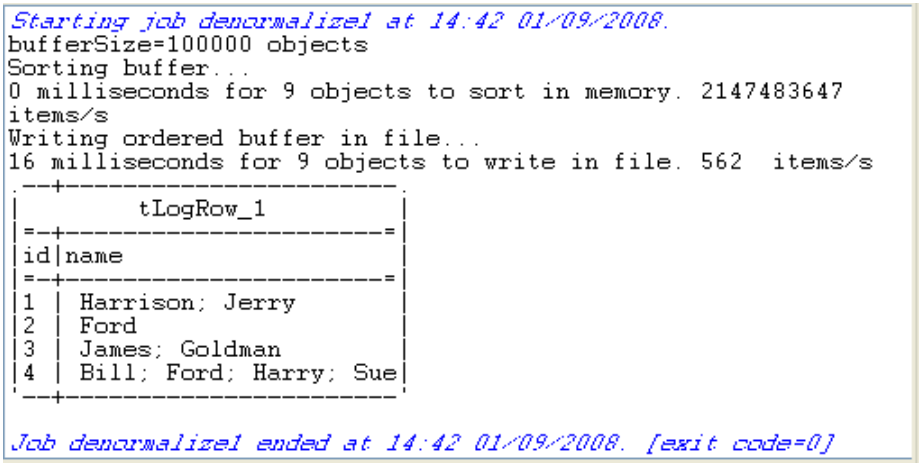
- In the design workspace, select **tSortRow**.
- Click the **Component** tab to define the basic settings for **tSortRow**.



- Set the **Schema Type** to **Built-In** and click **Sync columns** to retrieve the schema from the **tFileInputDelimited** component.
- In the **Criteria** panel, use the plus button to add a line and set the sorting parameters for the schema column to be processed. In this example we want to sort the *id* columns in ascending order.
- In the design workspace, select **tDenormalizeSortedRow**.
- Click the **Component** tab to define the basic settings for **tDenormalizeSortedRow**.



- Set the **Schema Type** to **Built-In** and click **Sync columns** to retrieve the schema from the **tSortRow** component.
- In the **Input rows count**field, enter the number of the input rows to be processed or press **Ctrl+Space** to access the context variable list and select the variable: `tFileInputDelimited_1_NB_LINE`.
- In the **To denormalize** panel, use the plus button to add a line and set the parameters to the column to be denormalize. In this example we want to denormalize the *name* column.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow**, see section *tLogRow*.
- Save your Job and press **F6** to execute it.



The result displayed on the console shows how the *name* column was denormalize.

tExternalSortRow

tExternalSortRow properties

Component family	Processing	
Function	Uses an external sort application to sort input data based on one or several columns, by sort type and order	
Purpose	Helps create metrics and classification table.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	File Name	Name of the file to be processed. Related topic: see *Talend Open Studio for Big Data*. *User Guide*.
	Field separator	Character, string or regular expression to separate fields.
	External command "sort" path	Enter the path to the external file containing the sorting algorithm to use.
	Criteria	Click the plus button to add as many lines as required for the sort to be complete. By default the first column defined in your schema is selected.
		Schema column: Select the column label from your schema, which the sort will be based on. Note that the order is essential as it determines the sorting priority.
		Sort type: Numerical and Alphabetical order are proposed. More sorting types to come.
		Order: Ascending or descending order.
Advanced settings	*Maximum memory*	Type in the size of physical memory you want to allocate to sort processing.
	Temporary directory	Specify the temporary directory to process the sorting command.
	Set temporary input file directory	Select the check box to activate the field in which you can specify the directory to handle your temporary input file.
	Add a dummy EOF line	Select this check box when using the **tAggregateSortedRow** component.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Related scenario

For related use case, see section *tSortRow*.

tExtractDelimitedFields

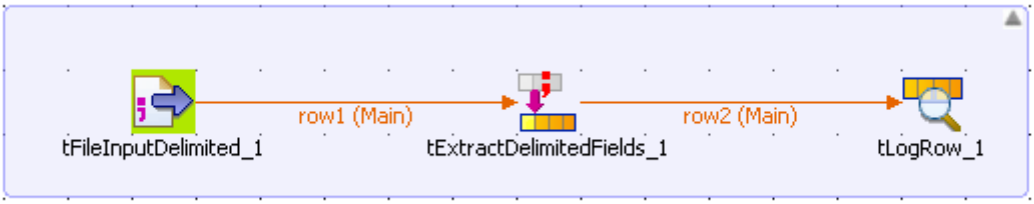
tExtractDelimitedFields properties

Component family	Processing/Fields	
Function	**tExtractDelimitedFields** generates multiple columns from a given column in a delimited file.	
Purpose	**tExtractDelimitedFields** helps to extract 'fields' from within a string to write them elsewhere for example.	
Basic settings	*Field to split*	Select an incoming field from the **Field to split** list to split.
	Field separator	Set field separator. Since this component uses regex to split a filed and the regex syntax uses special characters as operators, make sure to precede the regex operator you use as a field separator by a double backslash. For example, you have to use "\\|" instead of "|".
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **tExtractDelimitedFields**.
		Built-in: You create the schema and store it locally for the component. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Advanced separator (for number)*	Select this check box to modify the separators used for numbers.
	Trim column	Select this check box to remove leading and trailing whitespace from all columns.
	Check each row structure against schema	Select this check box to synchronize every row against the input schema.
	Validate date	Select this check box to check the date format strictly against the input schema.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output components. It allows you to extract data from a delimited field, using a **Row** > **Main** link, and enables you to create a reject flow filtering data which type does not match the defined type.	
Limitation	n/a	

Scenario: Extracting fields from a comma-delimited file

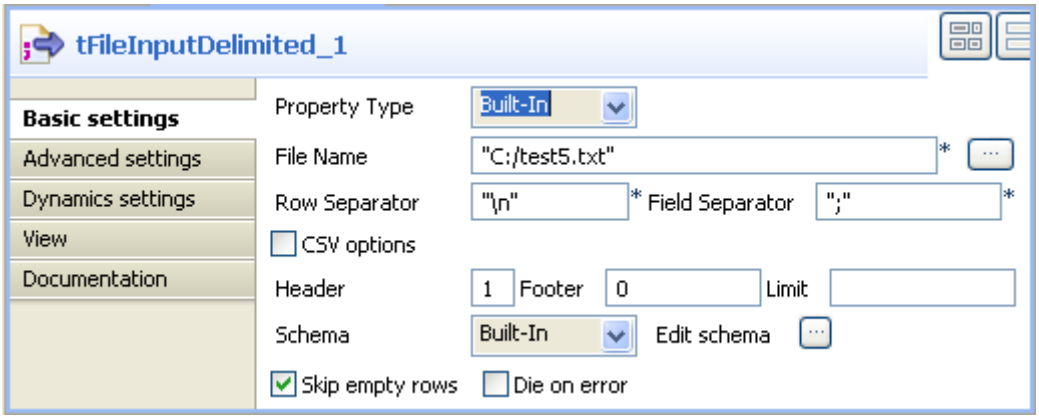
This scenario describes a three-component Job where the **tExtractdelimitedFields** component is used to extract two columns from a comma-delimited file.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tExtractDelimitedFields**, and **tLogRow**.
- Via a right-click each of the three components, connect them using **Row Main** links.

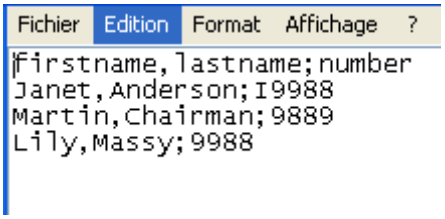


- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.
- In the **Basic settings** view, set **Property Type** to **Built-In**.
- Click the three-dot **[...]** button next to the **File Name** field to select the path to the input file.

The **File Name** field is mandatory.

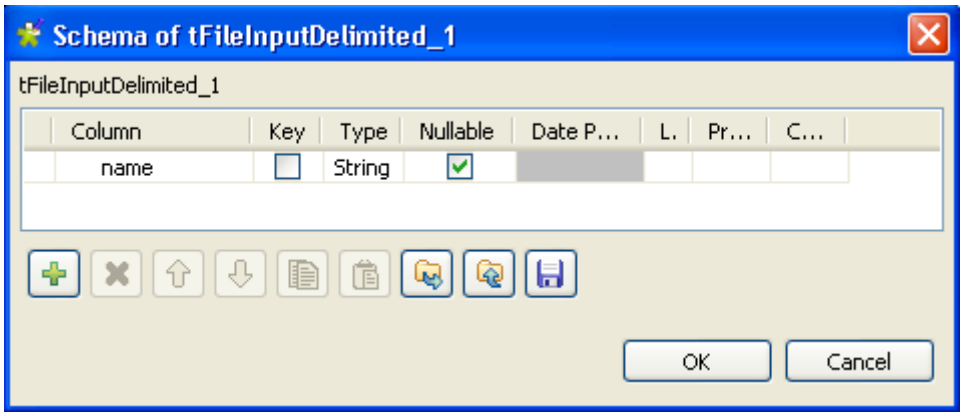


The input file used in this scenario is called *test5*. It is a text file that holds comma-delimited data.

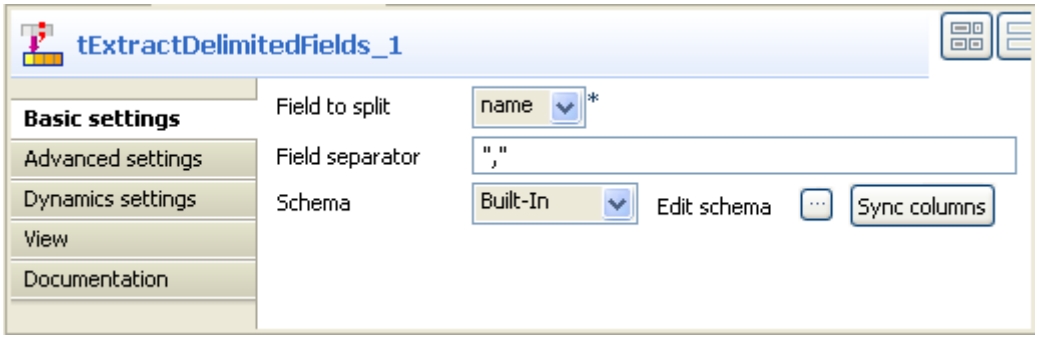


- In the **Basic settings** view, fill in all other fields as needed. For more information, see section *tMDMInput properties*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows

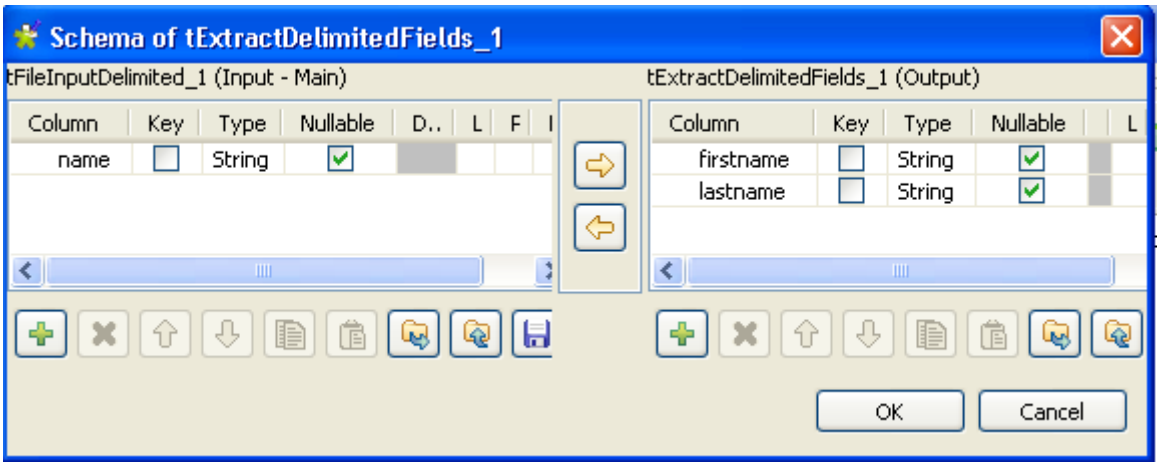
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of one column, *name*.



- In the design workspace, select **tExtractDelimitedFields**.
- Click the **Component** tab to define the basic settings for **tExtractDelimitedFields**.



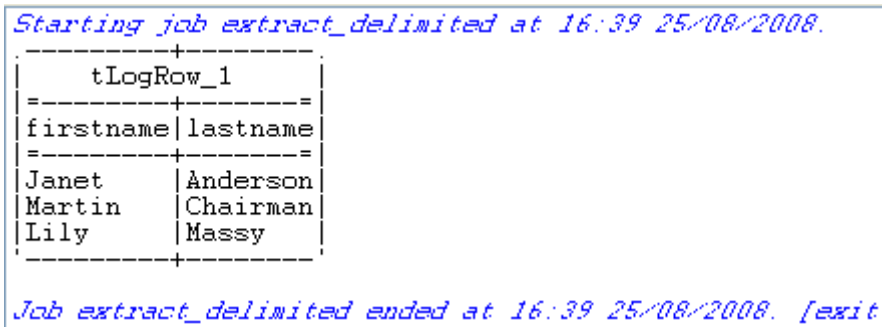
- From the **Field to split** list, select the column to split, *name* in this scenario.
- In the **Field separator** field, enter the corresponding separator.
- Click **Edit schema** to describe the data structure of this processing component.
- In the output panel of the **[Schema of tExtractDelimitedFields]** dialog box, click the plus button to add two columns for the output schema, *firstname* and *lastname*.



In this scenario, we want to split the *name* column into two columns in the output flow, *firstname* and *lastname*.

- Click **OK** to close the **[Schema of tExtractDelimitedFields]** dialog box.

- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*.
- Save your Job and press **F6** to execute it.



First names and last names are extracted and displayed in the corresponding defined columns on the console.

tExtractEBCDICFields

tExtractEBCDICFields properties

Component family	Processing/Fields	
Function	**tExtractEBCDICFields** generates multiple columns from a given column using regex matching.	
Purpose	**tExtractEBCDICFields** allows you to use regular expressions to extract data from a formatted string.	
Basic settings	*Field*	Select an incoming field from the **Field** list to extract.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** connection.
	Schema and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: Select this option to edit the Built-in schema for the data to be processed.
	Edit schema	Click **[...]** to edit the Built-in or Repository schema for the data to be processed.
	Sync columns	Click this button to retrieve the schema defined in the input component. This button is available only when an input component is connected to this component via a **Row** > **Main** connection.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all column	Select this check box to remove leading and trailing whitespaces from defined columns.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output components. It allows you to extract data from a delimited field, using a **Row** > **Main** link, and enables you to create a reject flow filtering data which type does not match the defined type.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario: Extracting name, domain and TLD from e-mail addresses*.

tExtractPositionalFields

tExtractPositionalFields properties

Component family	Processing/Fields	
Function	**tExtractPositionalFields** generates multiple columns from one column using positional fields.	
Purpose	**tExtractPositionalFields** allows you to use a positional pattern to extract data from a formatted string.	
Basic settings	*Field*	Select an incoming field from the **Field** list to extract.
	Customize	Select this check box to customize the data format of the positional file and define the table columns: **Column**: Select the column you want to customize. **Size**: Enter the column size. **Padding char**: Type in between inverted commas the padding character used, in order for it to be removed from the field. A space by default. **Alignment**: Select the appropriate alignment parameter.
	Pattern	Enter the pattern to use as basis for the extraction. A pattern is length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this fields are consistent with the schema defined.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **t**PositionalFields.
		Built-in: You create the schema and store it locally for the component. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Advanced separator (for number)*	Select this check box to modify the separators used for numbers.
	Trim Column	Select this check box to remove leading and trailing whitespace from all columns.
	Check each row structure against schema	Select this check box to synchronize every row against the input schema.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output components. It allows you to extract data from a delimited field, using a **Row** > **Main** link, and enables you to create a reject flow filtering data which type does not match the defined type.	
Limitation	n/a	

Related scenario

For a related scenario, see section *Scenario: Extracting name, domain and TLD from e-mail addresses*.

tExtractRegexFields

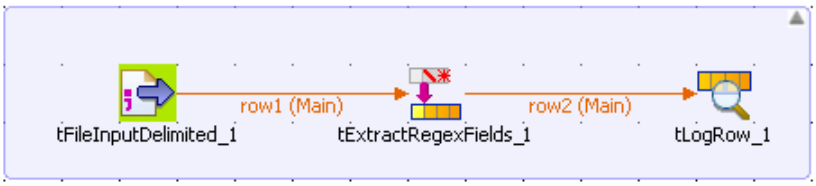
tExtractRegexFields properties

Component family	Processing/Fields	
Function	**tExtractRegexFields** generates multiple columns from a given column using regex matching.	
Purpose	**tExtractRegexFields** allows you to use regular expressions to extract data from a formatted string.	
Basic settings	*Field to split*	Select an incoming field from the **Field to split** list to split.
	Regex	Enter a regular expression according to the programming language you are using.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component connected to **tExtractRegexFields**.
		Built-in: You create and store the schema locally for the component. Related topic: see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Die on error*	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
	Check each row structure against schema	Select this check box to synchronize every row against the input schema.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output components. It allows you to extract data from a delimited field, using a **Row** > **Main** link, and enables you to create a reject flow filtering data which type does not match the defined type.	
Limitation	n/a	

Scenario: Extracting name, domain and TLD from e-mail addresses

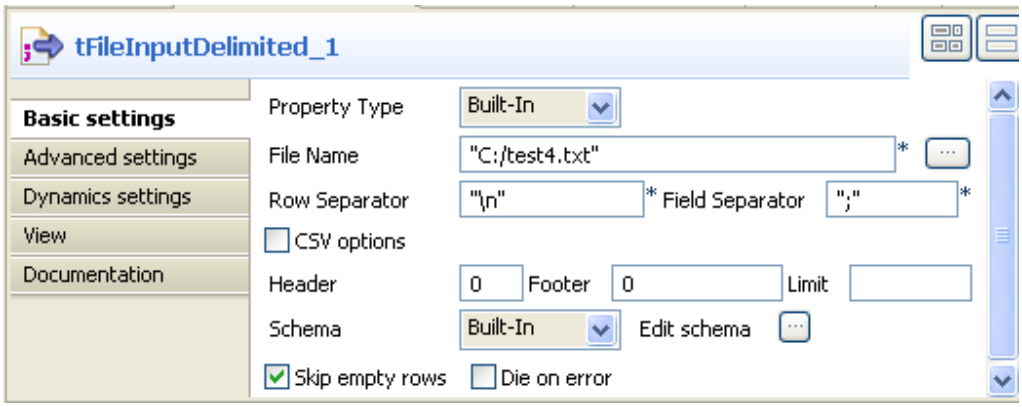
This Java scenario describes a three-component Job where **tExtractRegexFields** is used to specify a regular expression that corresponds to one column in the input data, *email*. The **tExtractRegexFields** component is used to perform the actual regular expression matching. This regular expression includes field identifiers for user name, domain name and Top-Level Domain name portions in each e-mail address. If the given e-mail address is valid, the name, domain and TLD are extracted and displayed on the console in three separate columns. Data in the other two input columns, *id* and *age* is extracted and routed to destination as well.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tExtractRegexFields**, and **tLogRow**.
- Connect the three components using **Row Main** links.



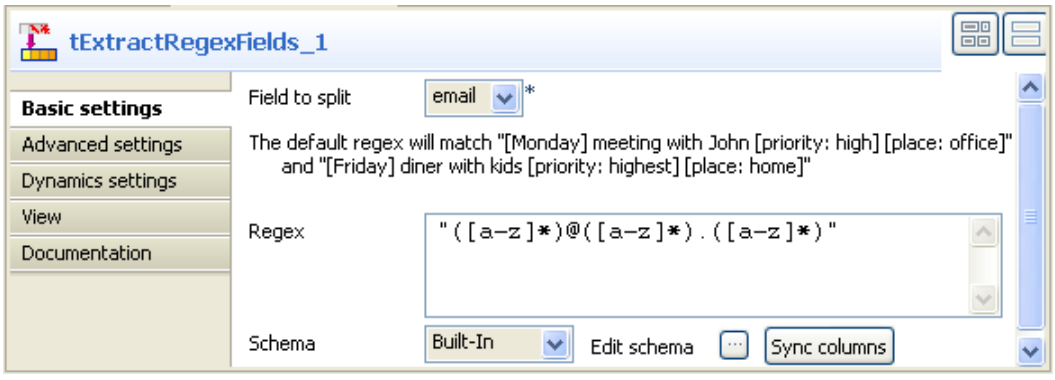
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.
- In the **Basic settings** view, set **Property Type** to **Built-In**.
- Click the three-dot **[...]** button next to the **File Name** field to select the path to the input file.

The **File Name** field is mandatory.

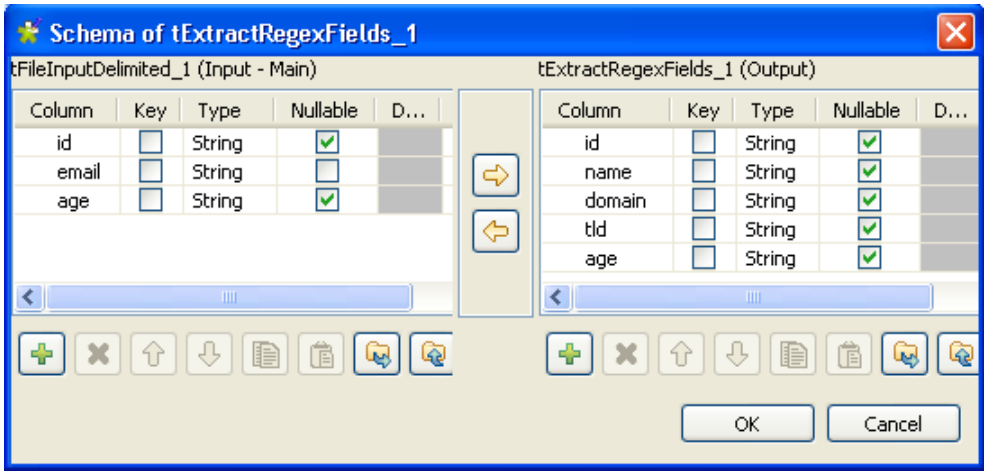


The input file used in this scenario is called *test4*. It is a text file that holds three columns: *id*, *email*, and *age*.

- Fill in all other fields as needed. For more information, see section *tMDMInput properties*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of the three columns, *id*, *email* and *age*.
- In the design workspace, select **tExtractRegexFields**.
- Click the **Component** tab to define the basic settings for **tExtractRegexFields**.
- From the **Field to split** list, select the column to split, *email* in this scenario.
- In the **Regex** panel, enter the regular expression you want to use to perform data matching, java regular expression in this scenario.



- Click **Edit schema** to describe the data structure of this processing component.
- In the output panel of the **[Schema of tExtractRegexFields]** dialog box, click the plus button to add five columns for the output schema.



In this scenario, we want to split the input *email* column into three columns in the output flow, *name*, *domain*, and *tld*. The two other input columns will be extracted as they are.

- Click **OK** to close the **[Schema of tExtractRegexFields]** dialog box.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see section *tLogRow*.
- Save your Job and press **F6** to execute it.

```
Starting job extract_regex_mine at 10:46 29/08/2008.
.--+----+------+---+---.
|       tLogRow_1        |
|=-+----+------+---+--=|
|id|name|domain|tld|age|
|=-+----+------+---+--=|
|1 |name|domain|com|24 |
|2 |name|domain|net|31 |
|3 |name|domain|org|13 |
'--+----+------+---+---'

Job extract_regex_mine ended at 10:46 29/08/2008. [exit code=0]
```

The **tExtractRegexFields** component matches all given e-mail addresses with the defined regular expression and extracts the name, domain, and TLD names and displays them on the console in three separate columns. The two other columns, *id* and *age*, are extracted as they are.

tExtractXMLField

tExtractXMLFieldbelongs to two component families: Processing and XML. For more information on **tExtractXMLField**, see section *tExtractXMLField*.

tFilterColumns

tFilterColumns Properties

Component family	Processing	
Function	Makes specified changes to the schema defined, based on column name mapping.	
Purpose	Helps homogenize schemas either on the columns order or by removing unwanted columns or adding new columns.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
Usage	This component is not startable (green background) and it requires an output component.	

Related Scenario

For more information regarding the **tFilterColumns** component in use, see section *Scenario: multiple replacements and column filtering*.

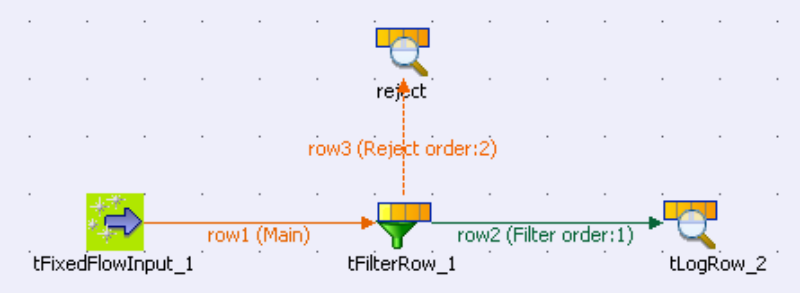
tFilterRow

tFilterRow Properties

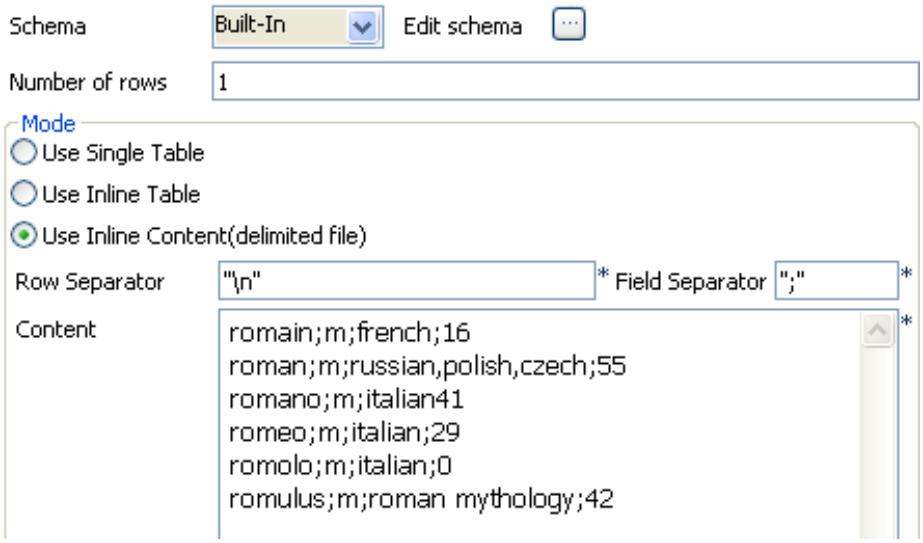
Component family	Processing	
Function	**tFilterRow** filters input rows by setting conditions on the selected columns.	
Purpose	**tFilterRow** helps parametrizing filters on the source data.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. *The schema is read-only.*
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Logical operator used to combine conditions	In the case you want to combine simple filtering and advanced mode, select the operator to combine both modes.
	Conditions	Click the plus button to add as many conditions as needed. The conditions are performed one after the other for each row. **Input column**: Select the column of the schema the function is to be operated on **Function**: Select the function on the list **Operator**: Select the operator to bind the input column with the value **Value**: Type in the filtered value, between quotes if need be.
	Use advanced mode	Select this check box when the operation you want to perform cannot be carried out through the standard functions offered. In the text field, type in the regular expression as required.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component is not startable (green background) and it requires an output component.	

Scenario: Filtering and searching a list of names

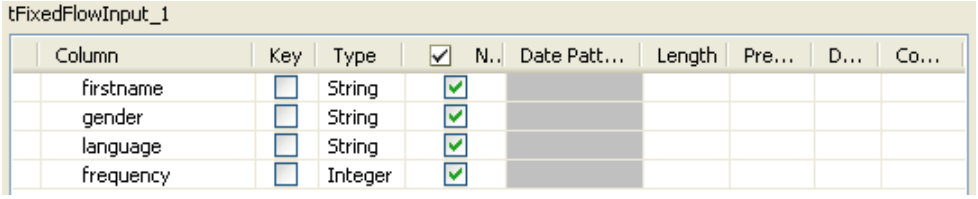
The following scenario is a Java Job that uses a simple condition and a regular expression to filter a list of records. This scenario will output two tables: the first will list all Italian records where first names are shorter than six characters; the second will list all rejected records. An error message for each rejected record will display in the same table to explain why such a record has been rejected.



- Drop **tFixedFlowInput**, **tFilterRow** and **tLogRow** from the **Palette** onto the design workspace.
- Connect the **tFixedFlowInput** to the **tFilterRow**, using a **Row** > **Main** link. Then, connect the **tFilterRow** to the **tLogRow**, using a **Row** > **Filter** link.
- Drop **tLogRow** from the **Palette** onto the design workspace and rename it as *reject*. Then, connect the **tFilterRow** to the *reject*, using a **Row** > **Reject** link.
- Double-click **tFixedFlowInput** to display its **Basic settings** view and define its properties.
- Select the **Use Inline Content(delimited file)** option in the **Mode** area to define the input mode.

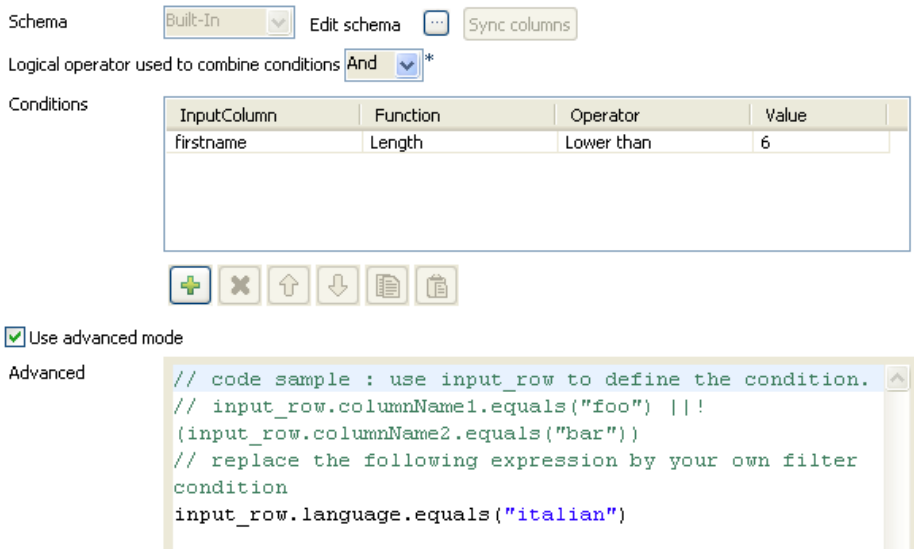


- Set the row and field separators in the corresponding fields. The row separator is a carriage return and the field separator is a semi-colon.
- From the **Schema** list, select **Built-in**. The properties and schema are **Built-in** for this Job. This means, the schema is not stored in the **Repository**.
- Click the three-dot button next to **Edit schema** to define the schema for the input file. In this example, the schema is made of the following four columns: *firstname*, *gender*, *language* and *frequency*. In the **Type** column, select **String** for the first three rows and select **Integer** for frequency.



- Click **OK** to validate and close the editor. A dialog box opens and asks you if you want to propagate the schema. Click **Yes**.

- Type in content in the Content multiline textframe according to the setting in the schema.
- Double-click **tFilterRow** to display its **Basic settings** view and define its properties.



- In the **Conditions** table, fill in the filtering parameters based on the *firstname* column.
- In **InputColumn**, select *firstname*, in **Function**, select **Length,** in **Operator**, select **Lower than**.
- In the **Value** column, type in *6* to filter only first names of which length is lower than six characters.

In the **Value** column, you must type in your values between double quotes for all data types, except for the **Integer** type, which does not need quotes.

- Then to implement the search on names whose language is italian, select the **Use advanced mode** check box and type in the following regular expression that includes the name of the column to be searched: `input_row.language.equals("italian")`
- To combine both conditions (simple and advanced), select **And** as logical operator for this example.
- In the **Basic settings** of **tLogRow** components, select **Table (print values in cells of a table)** in the **Mode** area.
- Save your Job and press **F6** to execute it.

```
Starting job FilterRow at 18:32 16/12/2010.

[statistics] connecting to socket on port 3601
[statistics] connected
.---------+------+---------+---------.
|            tLogRow_2                |
|=--------+------+---------+---------=|
|firstname|gender|language |frequency|
|=--------+------+---------+---------=|
|romeo    |m     |italian  |29       |
'---------+------+---------+---------'

.---------+------+---------------------+---------+-------------------------------------------------.
|                                       reject                                                      |
|=--------+------+---------------------+---------+-------------------------------------------------=|
|firstname|gender|language             |frequency|errorMessage                                     |
|=--------+------+---------------------+---------+-------------------------------------------------=|
|romain   |m     |french               |16       | firstname.length()<6 faild|advanced condition failed|
|roman    |m     |russian,polish,czech |55       |advanced condition failed                        |
|romano   |m     |italian41            |null     | firstname.length()<6 faild|advanced condition failed|
|romolo   |m     |italian              |0        | firstname.length()<6 faild                      |
|romulus  |m     |roman mythology      |42       | firstname.length()<6 faild|advanced condition failed|
'---------+------+---------------------+---------+-------------------------------------------------'
[statistics] disconnected
Job FilterRow ended at 18:32 16/12/2010. [exit code=0]
```

Thus, the first table lists records that have Italian names made up of less than six characters and the second table lists all records that do not match the filter condition “rejected record”. Each rejected record has a corresponding error message that explains the reason of rejection.

tJoin

tJoin properties

Component family	Processing	
Function	**tJoin** joins two tables by doing an exact match on several columns. It compares columns from the main flow with reference columns from the lookup flow and outputs the main flow data and/or the rejected data.	
Purpose	This component helps you ensure the data quality of any source data against a reference data source.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit schema** to make changes to the schema. Note that if you make changes to a remote schema, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Include lookup columns in output	Select this check box to include the lookup columns you define in the output flow.
Key definition	*Input key attribute*	Select the column(s) from the main flow that needs to be checked against the reference (lookup) key column.
	Lookup key attribute	Select the lookup key columns that you will use as a reference against which to compare the columns from the input flow.
	Inner join (with reject output)	Select this check box to join the two tables first and gather the rejected data from the main flow.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is not startable and it requires two input components and one or more output components.	
Limitation/prerequisite	n/a	

Scenario 1: Doing an exact match on two columns and outputting the main and rejected data

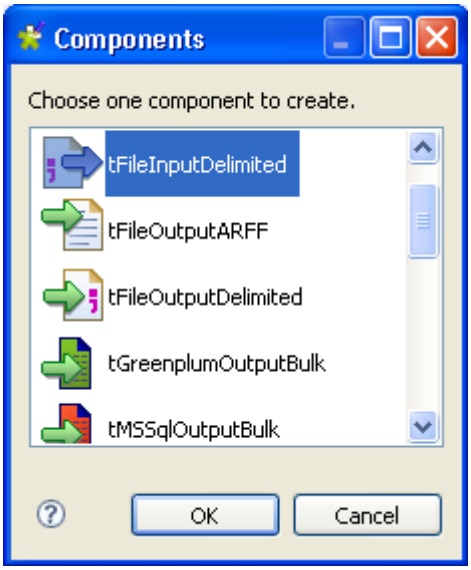
This scenario describes a five-component Job aiming at carrying out an exact match between the *firstnameClient* column of an input file against the data of the reference input file, and the *lastnameClient* column against the data of the reference input file. The outputs of this exact match are written in two separate files: exact data are written in an Excel file, and inaccurate data are written in a delimited file.

In this scenario, we have already stored the input schemas of the input and reference files in the Repository. For more information about storing schema metadata in the **Repository** tree view, see *Talend Open Studio for Big Data User Guide*.

Dropping and linking the components

1. In the **Repository** tree view, expand **Metadata** and the file node where you have stored the input schemas and drop the relevant file onto the design workspace.

   The **[Components]** dialog box appears.

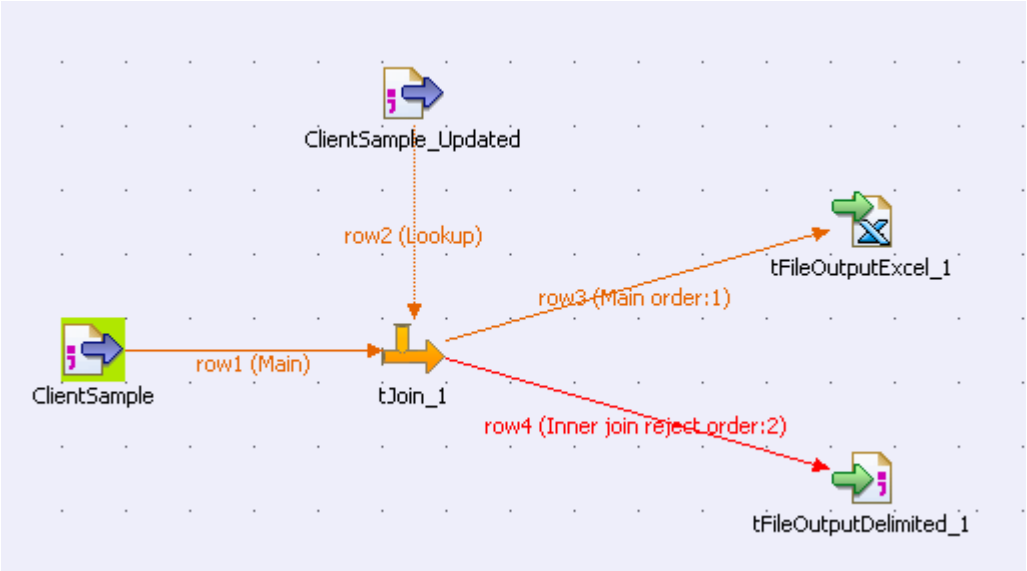


2. Select **tFileInputDelimited** from the list and click **OK** to close the dialog box.

   The **tFileInputDelimited** component displays in the workspace. The input file used in this scenario is called *ClientSample*. It holds four columns including the two columns *firstnameClient* and *lastnameClient* we want to do the exact match on.

3. Do the same for the second input file you want to use as a reference, *ClientSample_Update* in this scenario.

4. Drop the following components from the **Palette** onto the design workspace: **tJoin**, **tFileOutputExcel**, and **tFileOutputDelimited**.



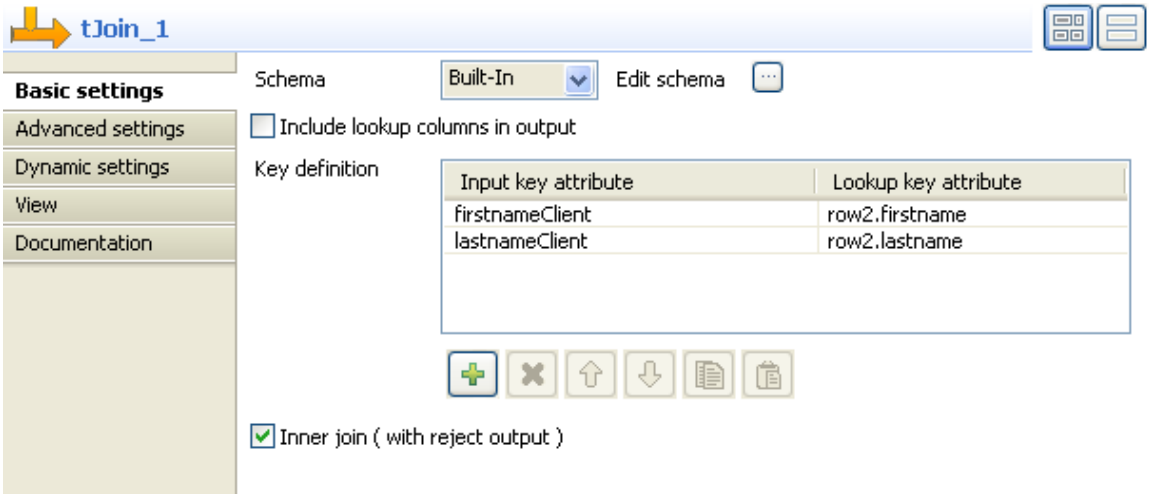
5. Connect the main and reference input files to **tJoin** using **Main** links. The link between the reference input file and **tJoin** appears as a lookup link on the design workspace.

6. Connect **tJoin** to **tFileOutputExcel** using the **Main** link and **tJoin** to **tFileOutputDelimited** using the **Inner join reject** link.

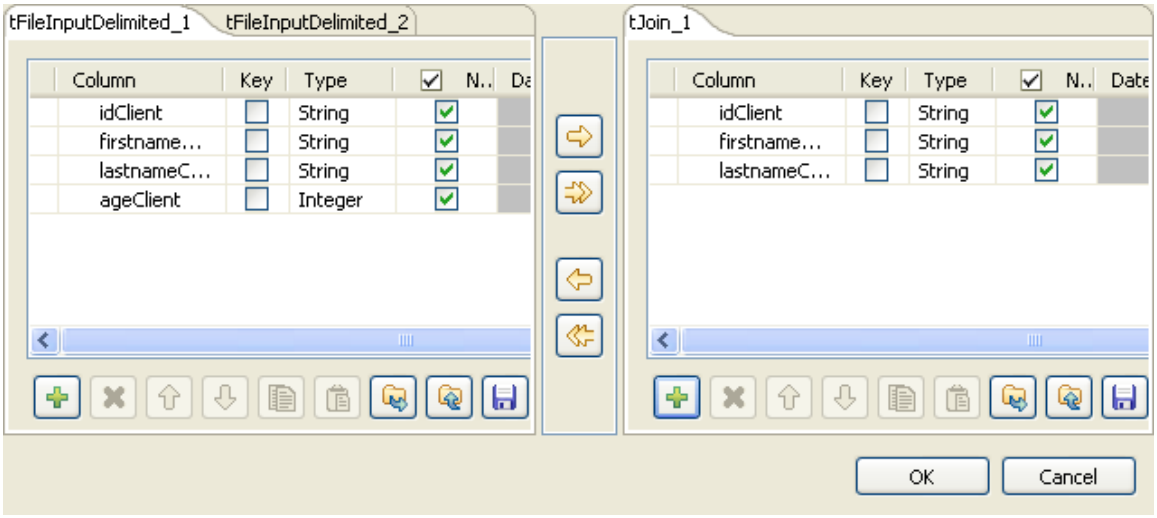
Configuring the components

1. If needed, double-click the main and reference input files to display their **Basic settings** views. All their property fields are automatically filled in. If you do not define your input files in the **Repository**, fill in the details manually after selecting **Built-in** in the **Property Type** field.

2. Double click **tJoin** to display its **Basic settings** view and define its properties.



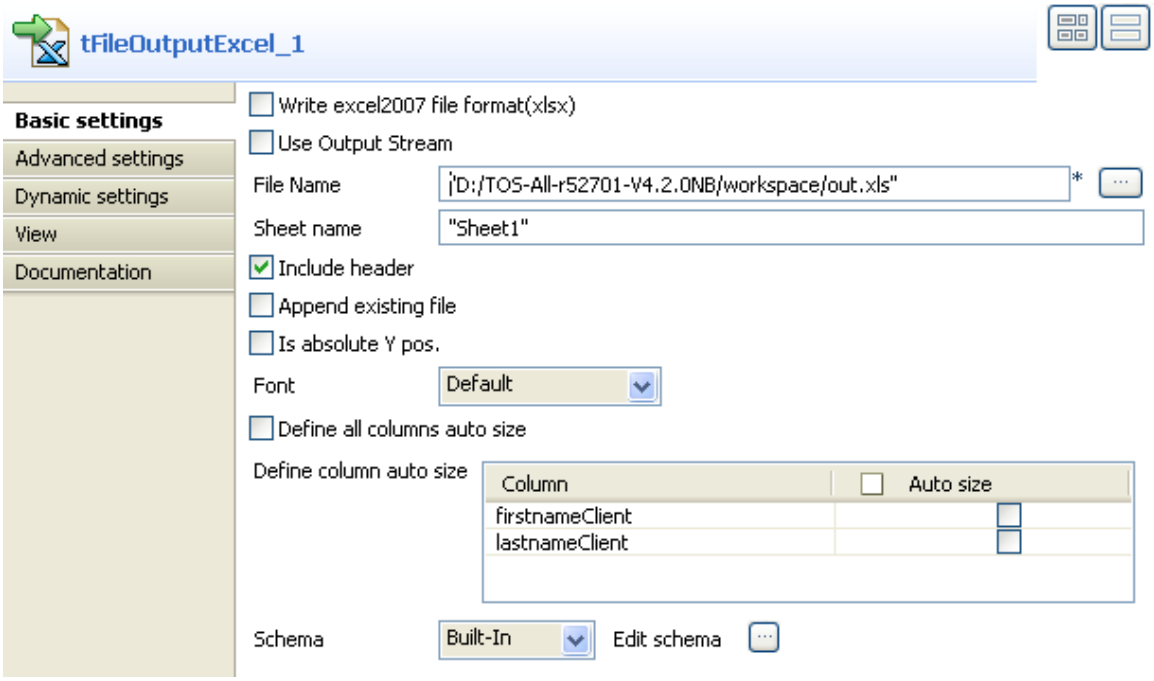
3. Click the **Edit schema** button to open a dialog box that displays the data structure of the input files, define the data you want to pass to the output components, three columns in this scenario, *idClient*, *firstnameClient* and *lastnameClient*, and then click **OK** to validate the schema and close the dialog box.



4. In the **Key definition** area of the **Basic settings** view of **tJoin**, click the plus button to add two columns to the list and then select the input columns and the output columns you want to do the exact matching on from the **Input key attribute** and **Lookup key attribute** lists respectively, *firstnameClient* and *lastnameClient* in this example.

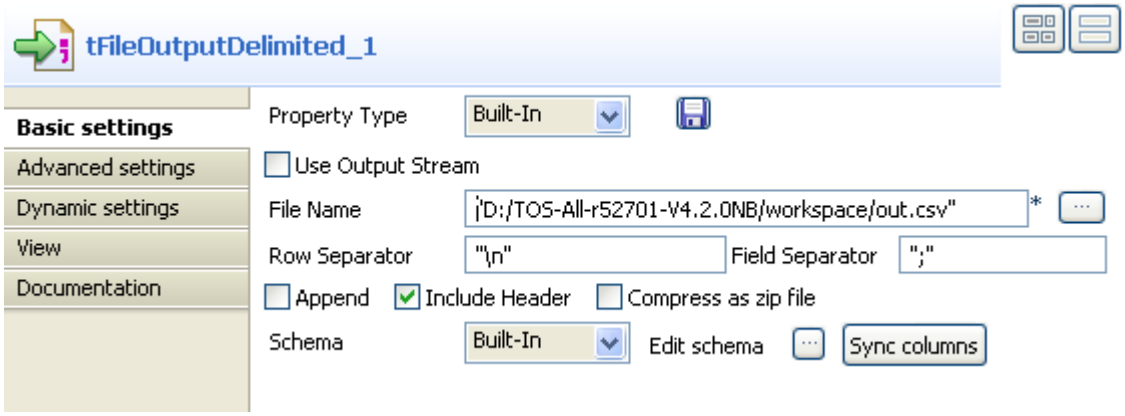
5. Select the **Inner join (with reject output)** check box to define one of the outputs as inner join reject table.

6. Double click **tFileOutputExcel** to display its **Basic settings** view and define its properties.



7. Set the destination file name and the sheet name, and select the **Include header** check box.

8. Double click **tFileOutputDelimited** to display its **Basic settings** view and define its properties.

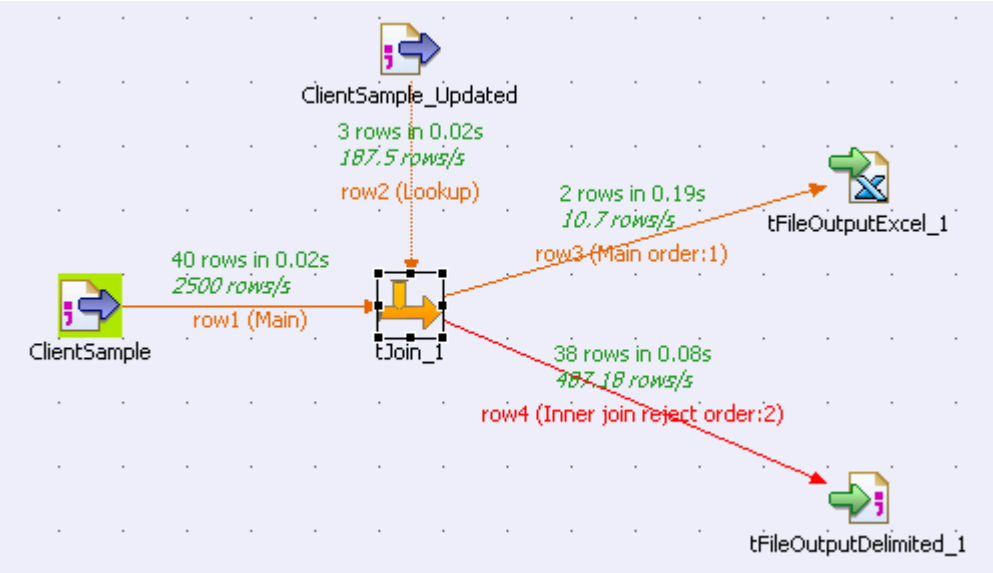


9. Set the destination file name, and select the **Include header** check box.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6**, or click **Run** on the **Run** tab to execute the Job.



The output of the exact match on the *firstnameClient* and *lastnameClient* columns is written to the defined Excel file.

	A	B	C
1	idClient	firstnameClient	lastnameClient
2	28	Herbert	Eisenhower
3	36	Chester	Grant

The rejected data is written to the defined delimited file.

```
1  idClient;firstnameClient;lastnameClient
2  1;Dwight;Madison
3  2;Franklin;Jackson
4  3;Ronald;Buchanan
5  4;Bill;Cleveland
6  5;William;Harrison
7  6;William;Fillmore
8  7;Harry;Adams
9  8;Harry;McKinley
10 9;Herbert;Reagan
11 10;Lyndon;Jefferson
12 11;Bill;Jackson
13 12;John;Hayes
14 13;Ulysses;Reagan
```

tMap

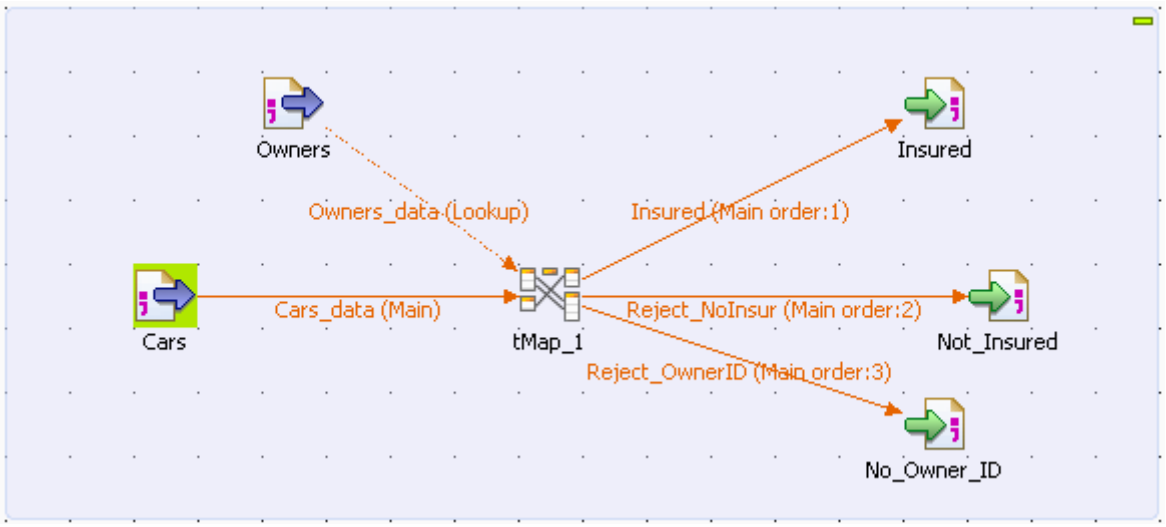
tMap properties

Component family	Processing	
Function	**tMap** is an advanced component, which integrates itself as plugin to *Talend Open Studio for Big Data*.	
Purpose	**tMap** transforms and routes data from single or multiple sources to single or multiple destinations.	
Basic settings	*Preview*	The preview is an instant shot of the Mapper data. It becomes available when Mapper properties have been filled in with data. The preview synchronization takes effect only after saving changes.
	Mapping links display as	**Auto**: the default setting is curves links **Curves**: the mapping display as curves **Lines**: the mapping displays as straight lines. This last option allows to slightly enhance performance.
	Map editor	It allows you to define the **tMap** routing and transformation properties. If you do not want to handle execution errors, you can click the **Property Settings** button at the top of the input area and select the **Die on error** check box (selected by default) in the **[Property Settings]** dialog box. It will kill the Job if there is an error.
Usage	Possible uses are from a simple reorganization of fields to the most complex Jobs of data multiplexing or demultiplexing transformation, concatenation, inversion, filtering and more...	
Limitation	The use of **tMap** supposes minimum Java knowledge in order to fully exploit its functionalities. This component is a junction step, and for this reason cannot be a start nor end component in the Job.	

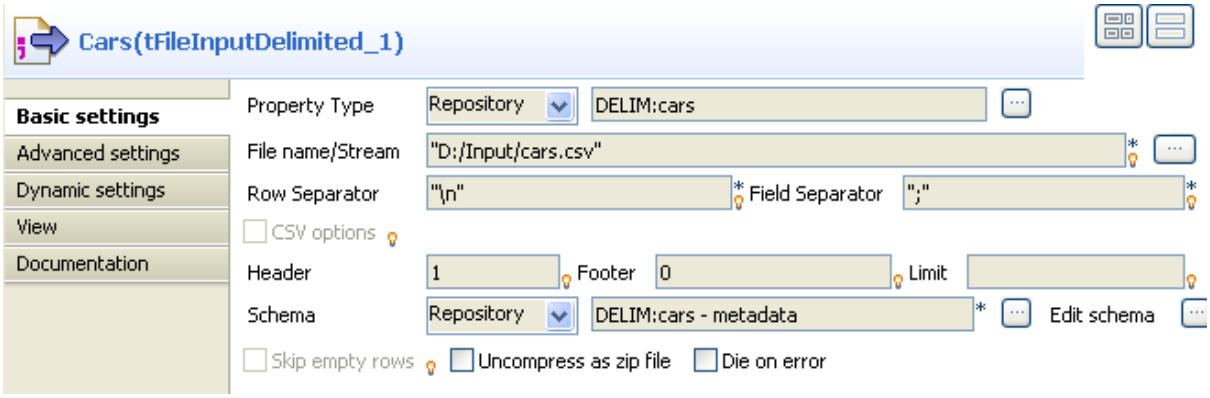
For further information, see *Talend Open Studio for Big Data User Guide*.

Scenario 1: Mapping data using a filter and a simple explicit join

The Job described below aims at reading data from a csv file with its schema stored in the Repository, looking up at a reference file, the schema of which is also stored in the Repository, then extracting data from these two files based on a defined filter to an output file and reject files.



- Click **File** in the **Palette** of components, select **tFileInputDelimited** and drop it onto the design workspace. Rename the component *Cars*, either by double-clicking the label in the design workspace or via the **View** tab of the **Component** view.
- Repeat this operation, and rename this second input component *Owners*.
- Click **Processing** in the **Palette** of components, select **tMap** and drop it onto the design workspace.
- Connect the two input components to the mapping component using **Row** > **Main** connections and label the connections *Cars_data* and *Owners_data* respectively.
- Double-click the **tFileInputDelimited** component labelled *Cars* to display its **Basic settings** view.



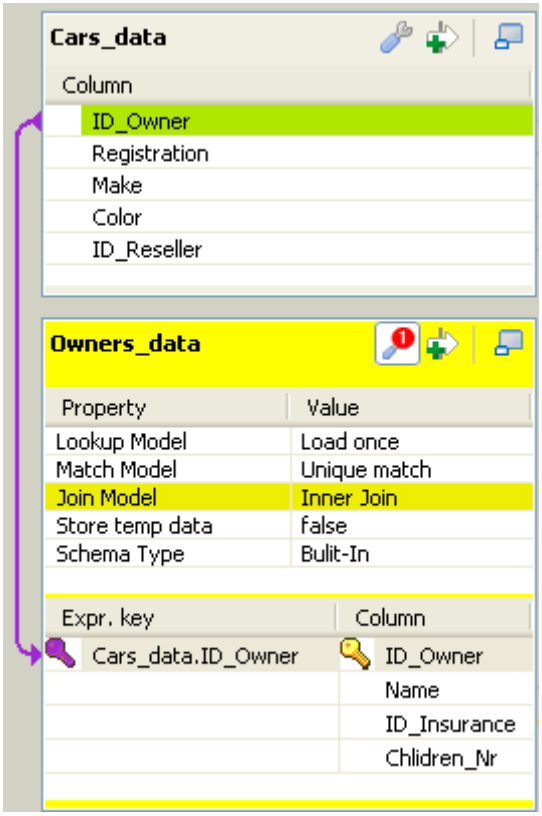
- Select **Repository** from the **Property type** list and select the component's schema, *cars* in this scenario, from the **[Repository Content]** dialog box. The rest fields are automatically filled.
- Double-click the component labelled *Owners* and repeat the setting operation. Select the appropriate metadata entry, *owners* in this scenario.

In this scenario, the input schemas are stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information regarding metadata creation in the Repository, see *Talend Open Studio for Big Data User Guide*.

- Double-click the **tMap** component to open the **Map Editor**.

  Note that the input area is already filled with the defined input tables and that the top table is the main input table, and the respective row connection labels are displayed on the top bar of the table.
- Create a join between the two tables on the *ID_Owner* column by simply dropping the *ID_Owner* column from the *Cars_data* table onto the *ID_Owner* column in the *Owners_data* table.

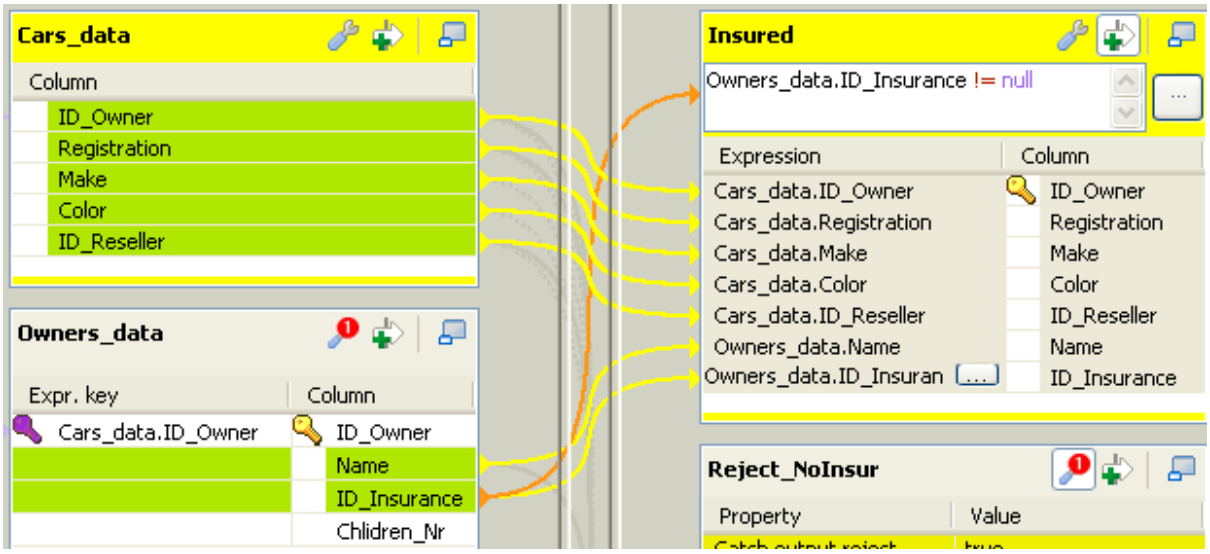
- Define this join as an inner join by clicking the **tMap settings** button, clicking in the **Value** field for **Join Model**, clicking the small button that appears in the field, and selecting **Inner Join** from the **[Options]** dialog box.



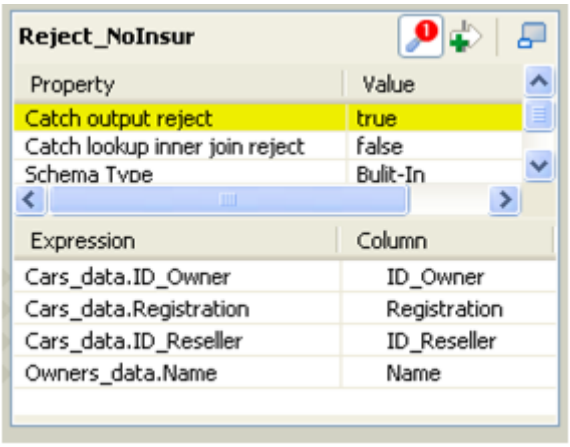
- Click the **[+]** button on the output area of the **Map Editor** to add three output tables: *Insured*, *Reject_NoInsur*, *Reject_OwnerID*.
- Drag all the columns of the *Cars_data* table to the *Insured* table.
- Drag the *ID_Owner*, *Registration*, and *ID_Reseller* columns of the *Cars_data* table and the *Name* column of the *Owners_data* table to the *Reject_NoInsur* table.
- Drag all the columns of the *Cars_data* table to the *Reject_OwnerID* table.

  For more information regarding data mapping, see *Talend Open Studio for Big Data* **User Guide**.
- Click the plus arrow button at the top of the *Insured* table to add a filter row.
- Drag the *ID_Insurance* column of the *Owners_data* table to the filter condition area and enter the formula meaning ‘not undefined’: `Owners_data.ID_Insurance != null`.

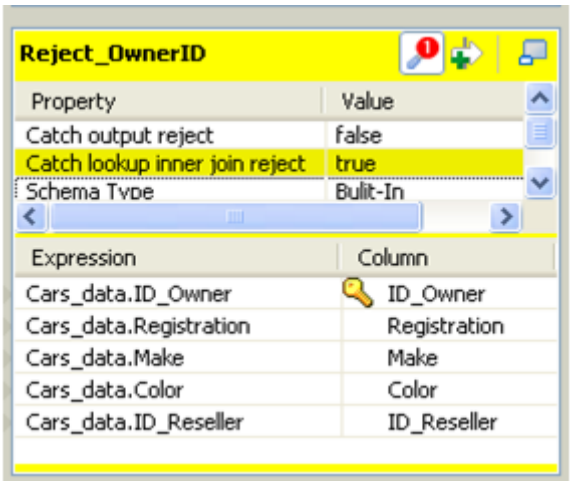
  With this filter, the *Insured* table will gather all the records that include an insurance ID.



- Click the **tMap settings** button at the top of the *Reject_NoInsur* table and set **Catch output reject** to **true** to define the table as a standard reject output flow to gather the records that do not include an insurance ID.

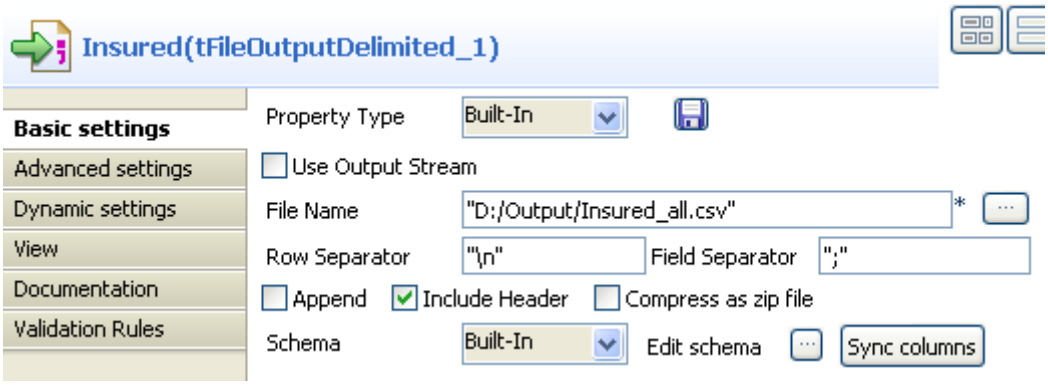


- Click the **tMap settings** button at the top of the *Reject_OwnerID* table and set **Catch lookup inner join reject** to **true** so that this output table will gather the records from the *Cars_data* flow with missing or unmatched owner IDs.



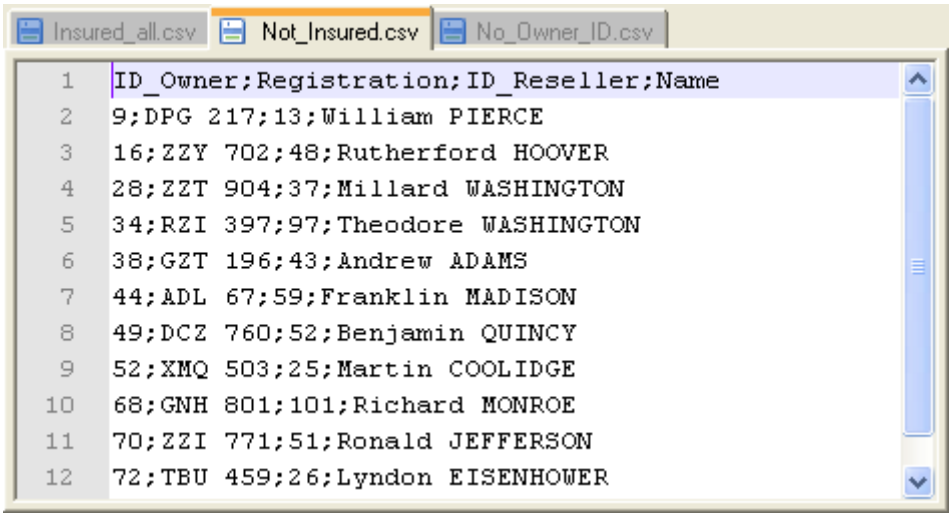
- Click **OK** to validate the mappings and close the **Map Editor**.

- Add three **tFileOutputDelimited** components to the design workspace and connect the **tMap** component to the three output components using the relevant **Row** connections.
- Relabel the three output components accordingly.
- Double-click each of the output components, one after the other, to define their properties. If you want a new file to be created, browse to the destination output folder, and type in a file name including the extension.
- Select the **Include header** check box to reuse the column labels from the schema as header row in the output file.



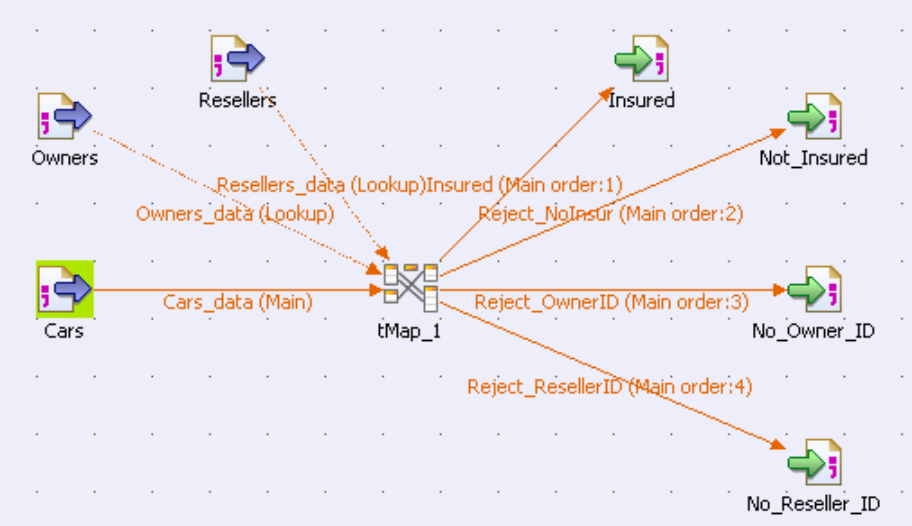
- Save your Job and press **F6** to run it.

  The output files are created, which contain the relevant data as defined.

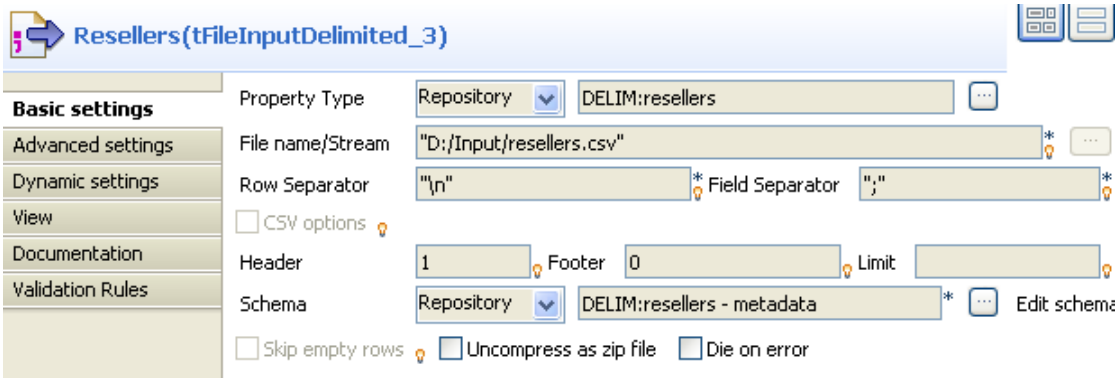


Scenario 2: Mapping data using inner join rejections

This scenario, based on scenario 1, adds one input file containing details about resellers and extra fields in the main output table. Two filters on inner joins are added to gather specific rejections.



- Click **File** in the **Palette** of Components, and drop a **tFileInputDelimited** component to the design workspace, and label the component *Resellers*.
- Connect it to the Mapper using a **Row** > **Main** connection, and label the connection *Resellers_data*.
- Double-click the *Resellers* component to display its **Basic settings** view.

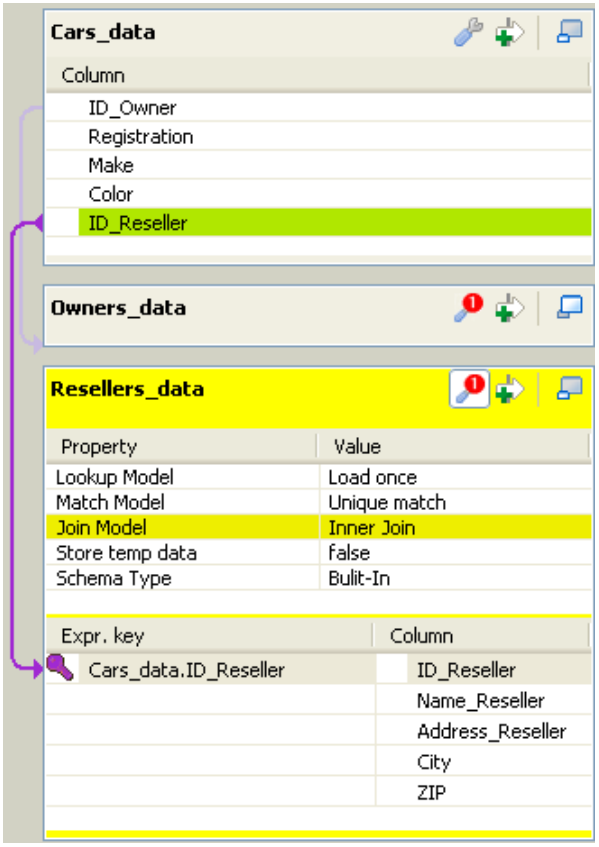


- Select **Repository** from the **Property type** list and select the component's schema, *resellers* in this scenario, from the **[Repository Content]** dialog box. The rest fields are automatically filled.

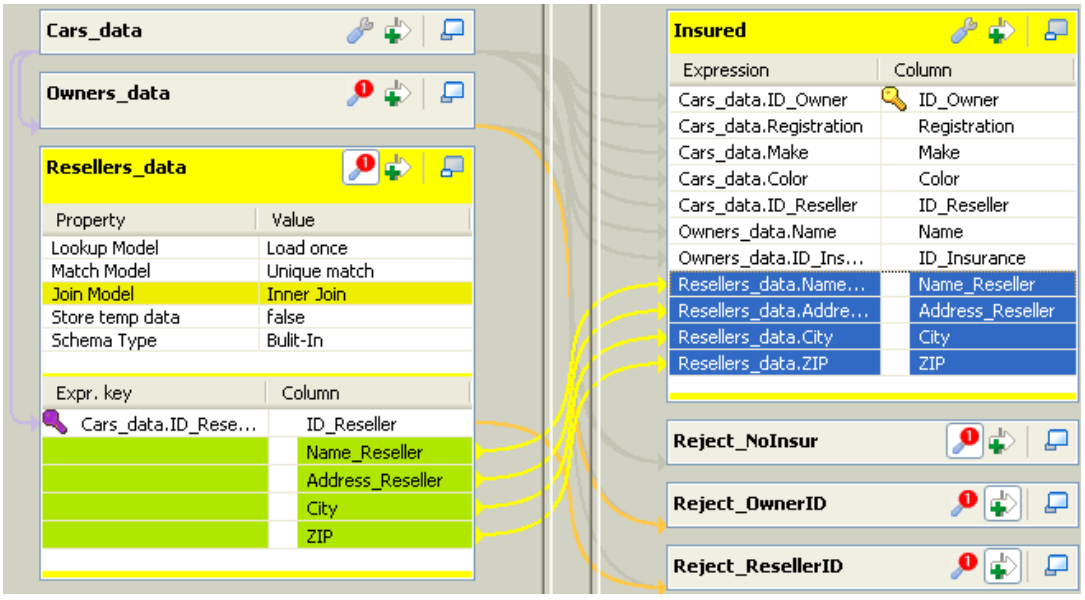
In this scenario, the input schemas are stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information regarding metadata creation in the Repository, see *Talend Open Studio for Big Data User Guide*.

- Double-click the **tMap** component to open the **Map Editor**.

  Note that the schema of the new input component is already added in the Input area.
- Create a join between the main input flow and the new input flow by dropping the *ID_Reseller* column of the *Cars_data* table to the *ID_Reseller* column of the *Resellers_data* table.
- Click the **tMap settings** button at the top of the *Resellers_data* table and set **Join Model** to **Inner Join**.



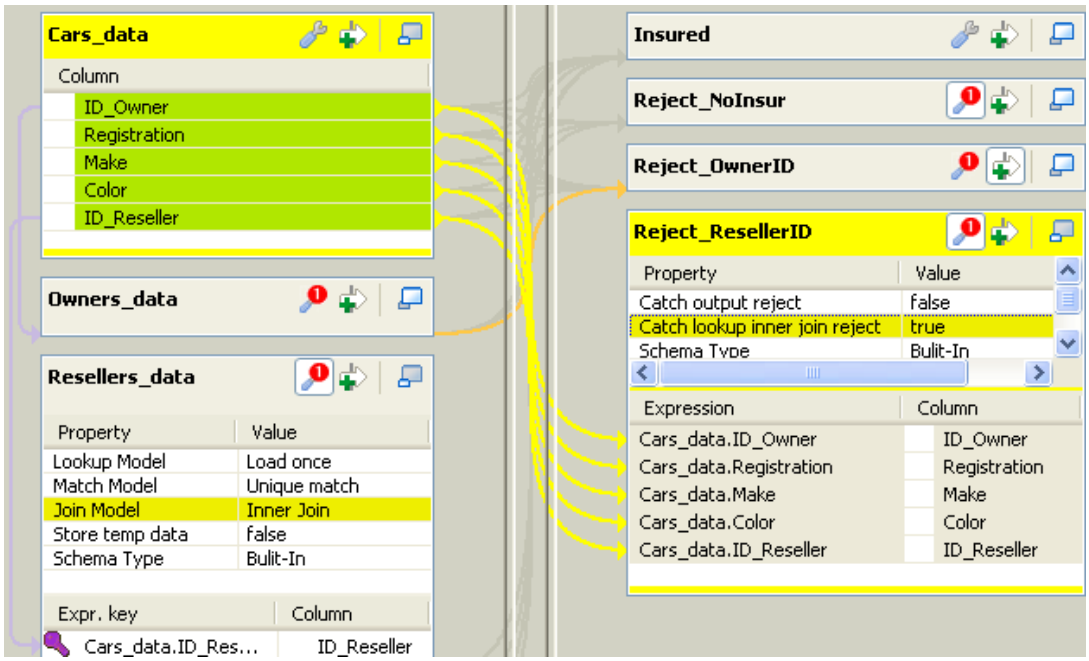
- Drag all the columns except *ID_Reseller* of the *Resellers_data* table to the main output table, *Insured*.



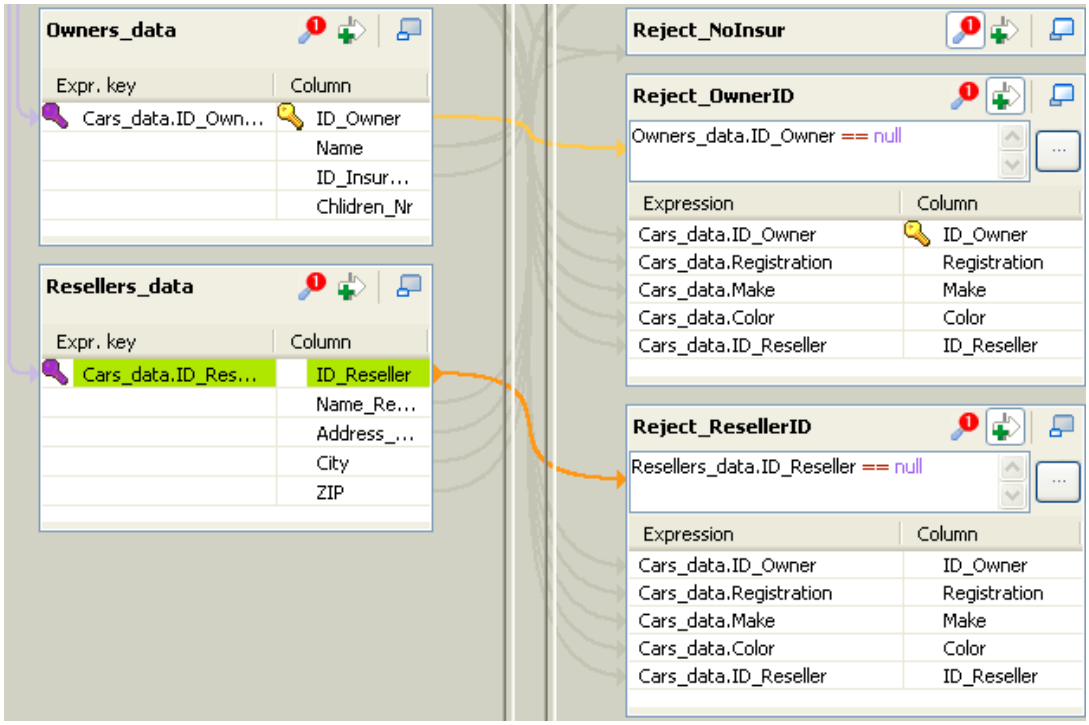
When two inner joins are defined, you either need to define two different inner join reject tables to differentiate the two rejections or, if there is only one inner join reject output, both inner join rejections will be stored in the same output.

- Click the **[+]** button at the top of the output area to add a new output table, and name this new output table *Reject_ResellerID*.
- Drag all the columns of the *Cars_data* table to the *Reject_ResellerID* table.
- Click the **tMap settings** button and select **Catch lookup inner join reject** to **true** to define this new output table as an inner join reject output.

If the defined inner join cannot be established, the information about the relevant cars will be gathered through this output flow.



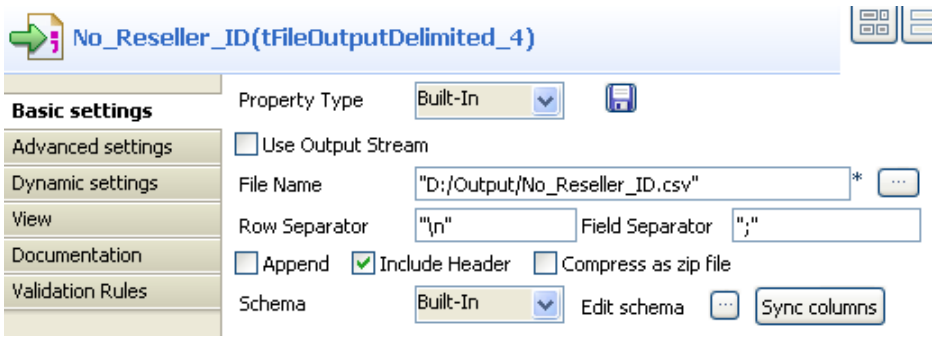
- Now apply filters on the two Inner Join reject outputs, in order for to distinguish the two types of rejection.
- In the first Inner Join output table, *Reject_OwnerID*, click the plus arrow button to add a filter line and fill it with the following formula to gather only owner ID related rejection: `Owners_data.ID_Owner==null`
- In the second Inner Join output table, *Reject_ResellerID*, repeat the same operation using the following formula: `Resellers_data.ID_Reseller==null`



- Click **OK** to validate the map settings and close the **Mapper Editor**.

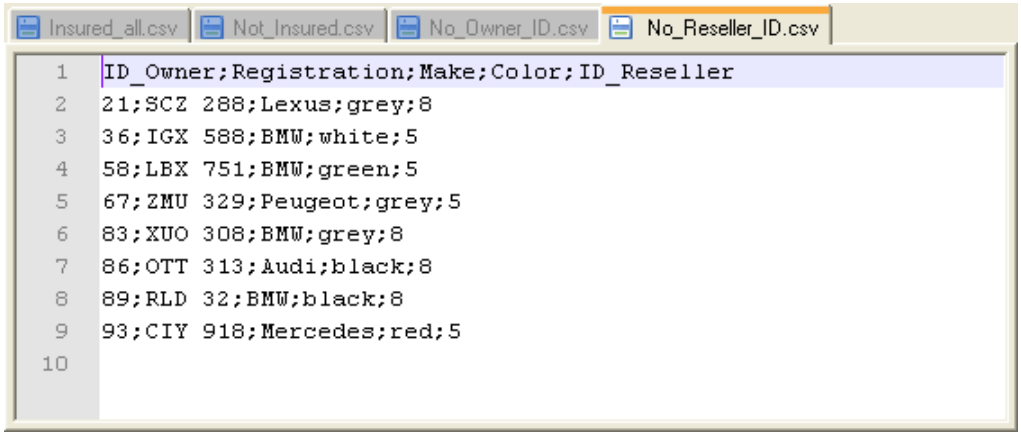
- Drop a new **tFileOutputDelimited** component from the **Palette** to the design workspace, and label the component *No_Reseller_ID*.
- Define the properties of the new **tFileOutputDelimited** component, as shown below.

  In this use case, simple specify the output file path and select the **Include Header** check box, and leave the other parameters as they are.



- Connect the **tMap** component to the new **tFileOutputDelimited** component by using the **Row** connection named *Reject_ResellerID*.
- To demonstrate the work of the Mapper, in this example, remove reseller IDs *5* and *8* from the input file *Resellers.csv*.
- Save your Job and press **F6** to run it.

  The four output files are all created in the specified folder, containing information as defined. The output file *No_Reseller_ID.csv* contains the *cars* information related to reseller IDs *5* and *8*, which are missing in the input file *Resellers.csv*.



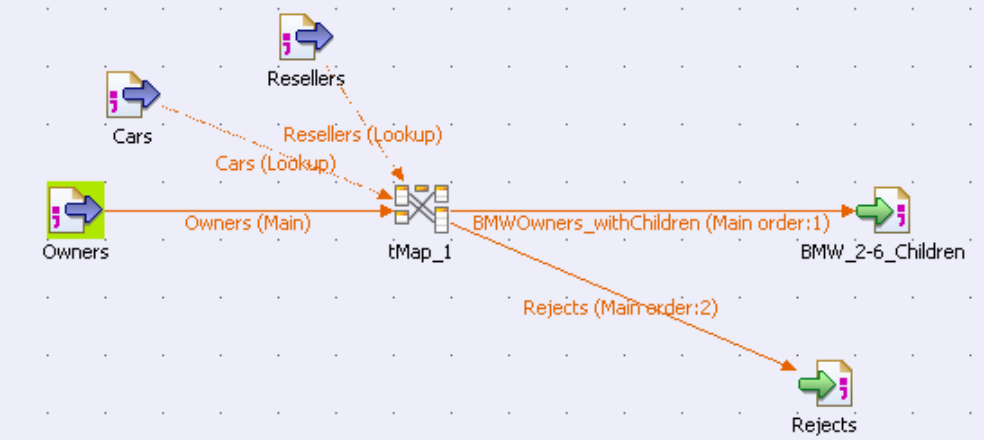
Scenario 3: Cascading join mapping

As third advanced use scenario, based on the scenario 2, add a new Input table containing Insurance details for example.

Set up an Inner Join between two lookup input tables (Owners and Insurance) in the Mapper to create a cascade lookup and hence retrieve Insurance details via the Owners table data.

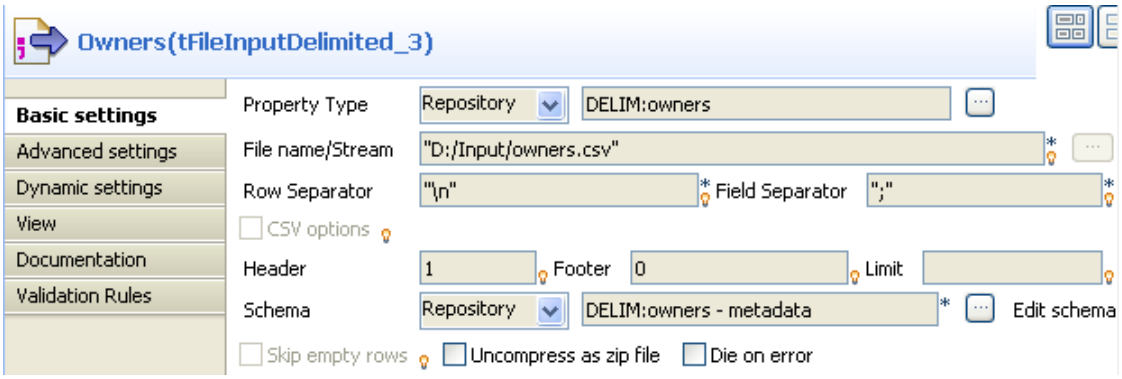
Scenario 4: Advanced mapping using filters, explicit joins and rejections

This scenario introduces a Job that allows you to find BMW owners who have two to six children (inclusive), for sales promotion purpose for example.



- Drop three **tFileInputDelimited** components, a **tMap** component, and two **tFileOutputDelimited** components from the **Palette** onto the design workspace, and label them to best describe their functions.
- Connect the input components to the **tMap** using **Row** > **Main** connections.

  Pay attention to the file you connect first as it will automatically be set as **Main** flow, and all the other connections will be **Lookup** flows. In this example, the connection for the input component *Owners* is the **Main** flow.
- Define the properties of each input components in the respective **Basic settings** view. Define the properties of *Owners*.

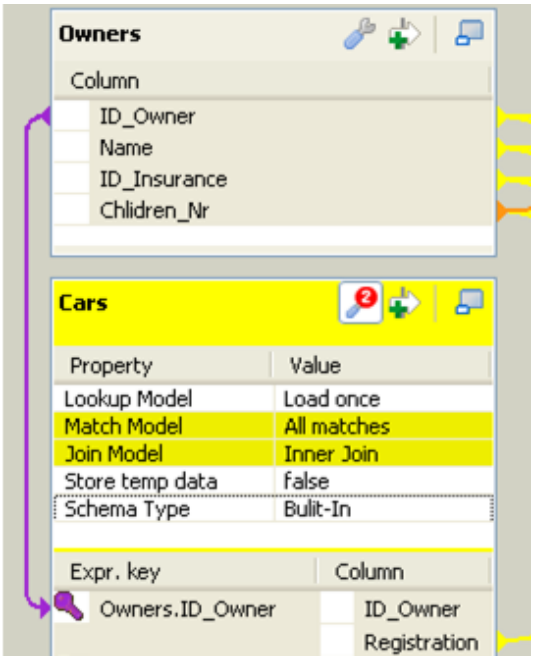


- Select **Repository** from the **Property type** list and select the component's schema, *owners* in this scenario, from the **[Repository Content]** dialog box. The rest fields are automatically filled.

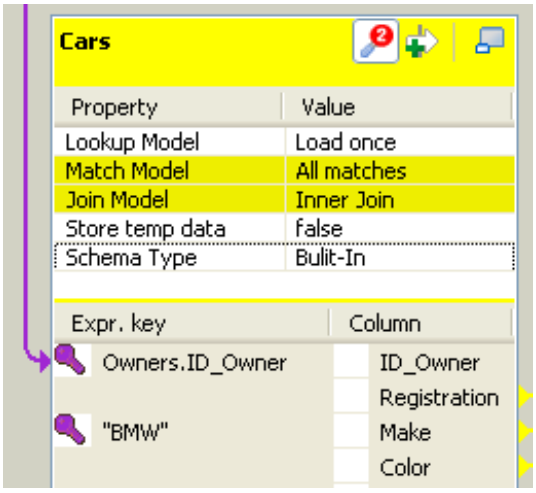
In this scenario, the input schemas are stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information regarding metadata creation in the Repository, see *Talend Open Studio for Big Data User Guide*.

- In the same way, set the properties of the other input components: *Cars* and *Resellers*. These two **Lookup** flows will fill in secondary (lookup) tables in the input area of the **Map Editor**.
- Then double-click the **tMap** component to launch the **Map Editor** and define the mappings and filters.
- Set an explicit join between the **Main** flow *Owner* and the **Lookup** flow *Cars* by dropping the *ID_Owner* column of the *Owners* table to the *ID_Owner* column of the *Cars* table.

The explicit join is displayed along with a hash key.

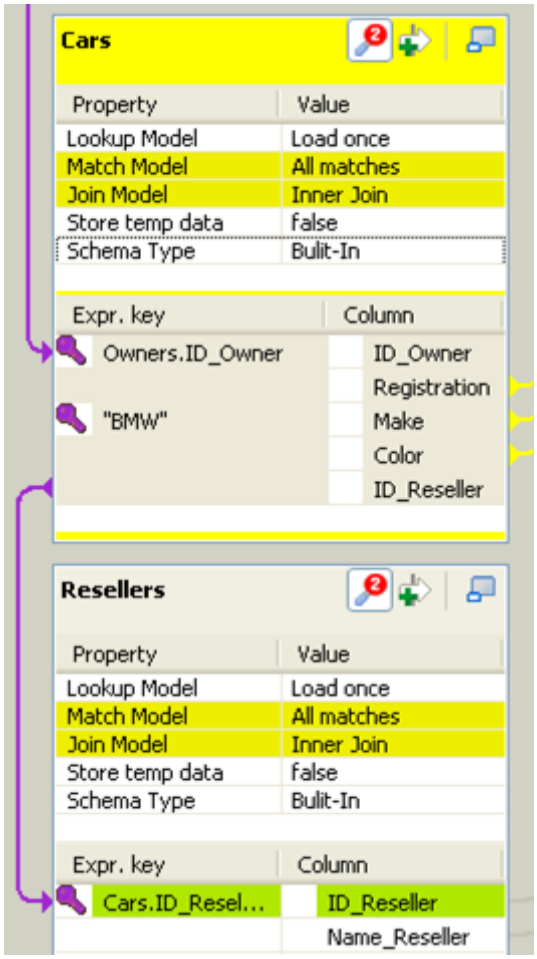


- In the **Expr. Key** field of the *Make* column, type in a filter. In this use case, simply type in `"BMW"` as the search is focused on the owners of this particular make.



- Implement a cascading join between the two lookup tables *Cars* and *Resellers* on the *ID_Reseller* column in order to retrieve resellers information.

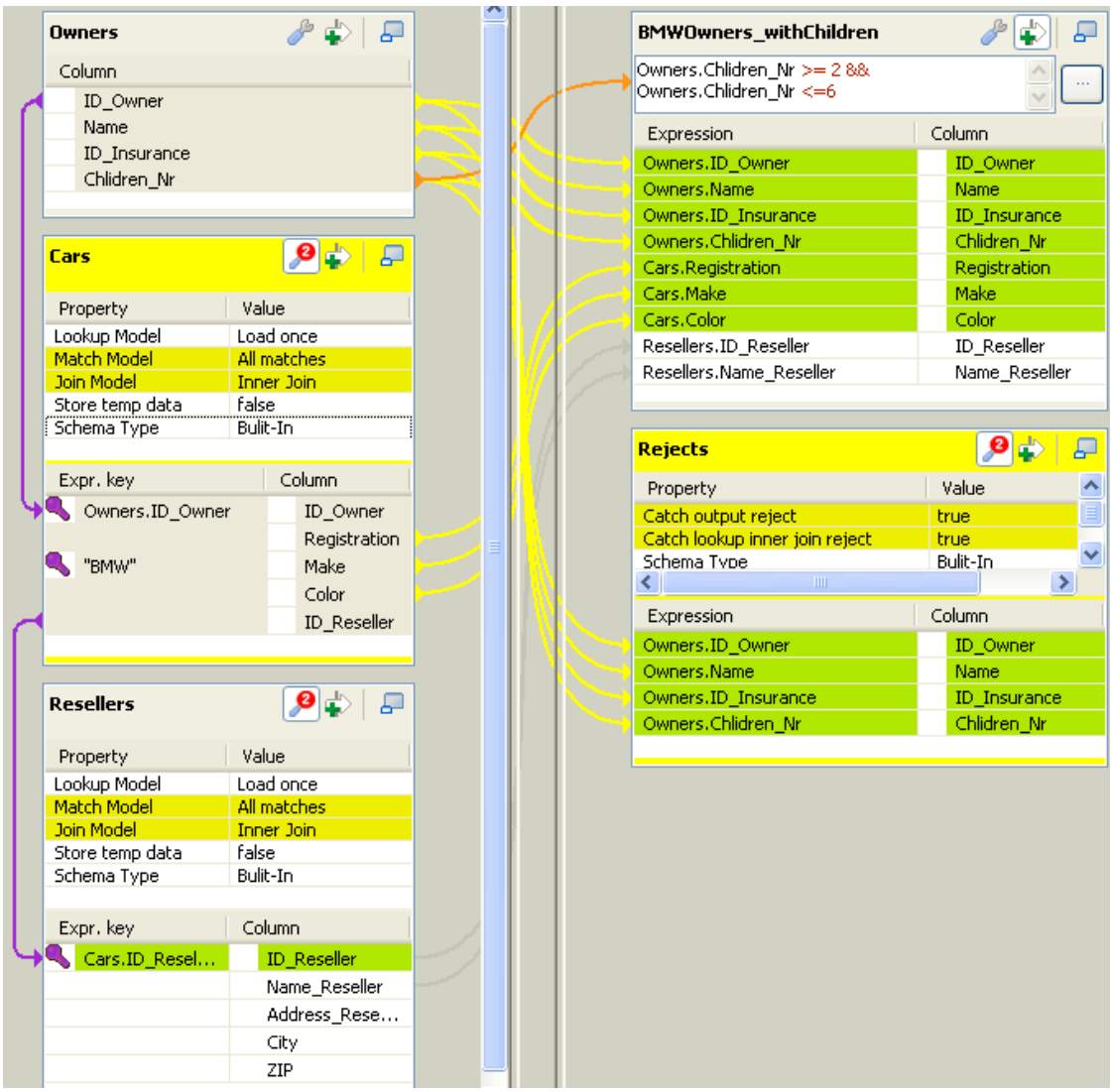
- As you want to reject the null values into a separate table and exclude them from the standard output, click the **tMap settings** button and set **Join Model** to **Inner Join** in each of the **Lookup** tables.



- In the tMap settings, you can set **Match Model** to **Unique match**, **First match**, or **All matches**. In this use case, the **All matches** option is selected. Thus if several matches are found in the Inner Join, i.e. rows matching the explicit join as well as the filter, all of them will be added to the output flow (either in rejection or the regular output).

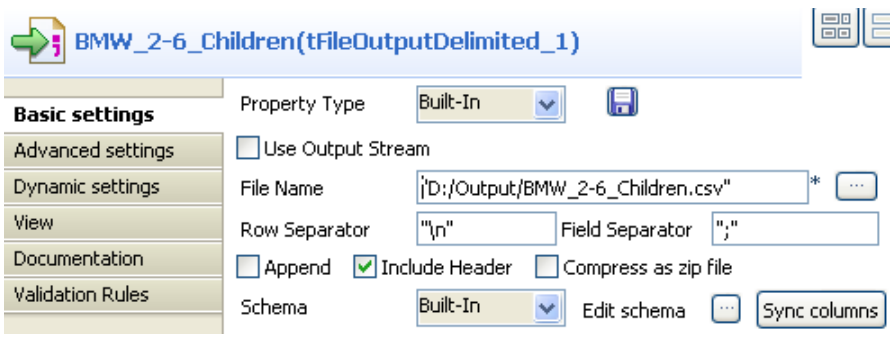
The **Unique match** option functions as a **Last match**. The **First match** and **All matches** options function as named.

- On the output area of the **Map Editor**, click the plus button to add two tables, one for the full matches and the other for the rejections.
- Drag all the columns of the *Owners* table, the *Registration*, *Make* and *Color* columns of the *Cars* table, and the *ID_Reseller* and *Name_Reseller* columns of the *Resellers* table to the main output table.
- Drag all the columns of the *Owners* table to the reject output table.
- Click the **Filter** button at the top of the main output table to display the **Filter** expression area.
- Type in a filter statement to narrow down the number of rows loaded in the main output flow. In this use case, the statement reads: `Owners.Children_Nr >= 2 && Owners.Children_Nr <= 6`.
- In the reject output table, click the **tMap settings** button and set the reject types.
- Set **Catch output reject** to **true** to collect data about BMW car owners who have less than two or more than six children.
- Set **Catch lookup inner join reject** to **true** to collect data about owners of other car makes and owners for whom the reseller information is not found.



- Click **OK** to validate the mappings and close the **Map Editor**.
- On the design workspace, right-click the **tMap** and pull the respective output link to the relevant output components.
- Define the properties of the output components in their respective **Basic settings** view.

  In this use case, simple specify the output file paths and select the **Include Header** check box, and leave the other parameters as they are.



- Save you Job and press **F6** to run it.

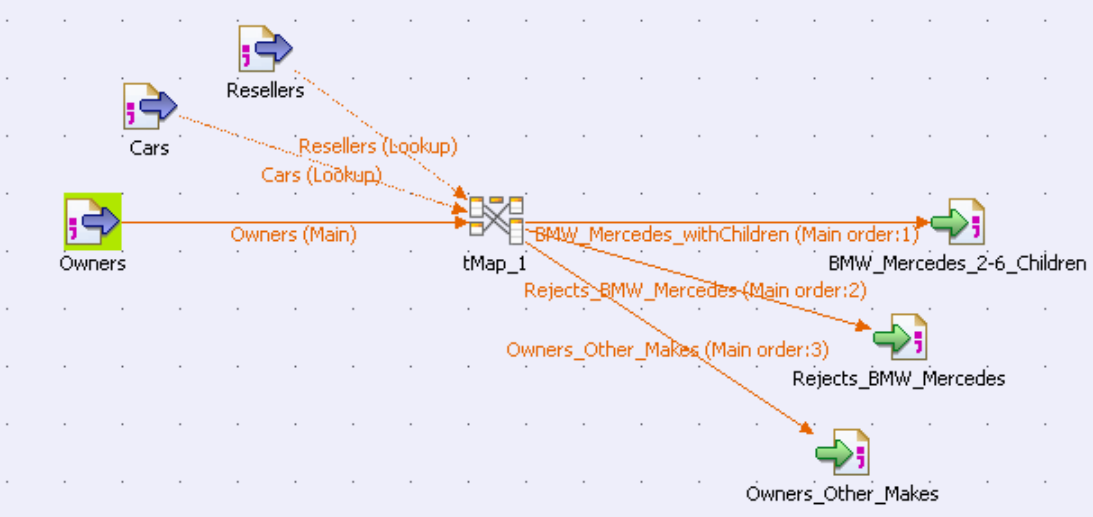
The main output file contains the information related to BMW owners who have two to six children, and the reject output file contains the information about the rest of the car owners.

BMW_2-6_Children.csv | Others.csv

```
1  ID_Owner;Name;ID_Insurance;Chlidren_Nr;Registration;Make;Color;
   ID_Reseller;Name_Reseller
2  15;Benjamin ROOSEVELT;103;2;PJB 72;BMW;yellow;80;Lyndon TAFT
3  29;Bill GRANT;70;2;TJS 538;BMW;grey;94;Grover ROOSEVELT
4  32;Chester VAN BUREN;184;2;PZI 361;BMW;yellow;6;Benjamin
   HARDING
5  38;Andrew ADAMS;16;3;GZT 196;BMW;yellow;43;William WASHINGTON
6  47;Bill ARTHUR;145;6;REZ 45;BMW;white;43;William WASHINGTON
7  70;Ronald JEFFERSON;70;3;ZZI 771;BMW;green;51;Abraham JACKSON
8
```

Scenario 5: Advanced mapping with filters and different rejections

This scenario is a modified version of the preceding scenario. It describes a Job that applies filters to limit the search to BMW and Mercedes owners who have two to six children and divides unmatched data into different reject output flows.



- Take the same Job as in section *Scenario 4: Advanced mapping using filters, explicit joins and rejections*.
- Drop a new **tFileOutputDelimited** component from the **Palette** on the design workspace, and name it *Rejects_BMW_Mercedes* to present its functionality.
- Connect the **tMap** component to the new output component using a **Row** connection and label the connection according to the functionality of the output component.

  This connection label will appear as the name of the new output table in the **Map Editor**.
- Relabel the existing output connections and output components to reflect their functionality.

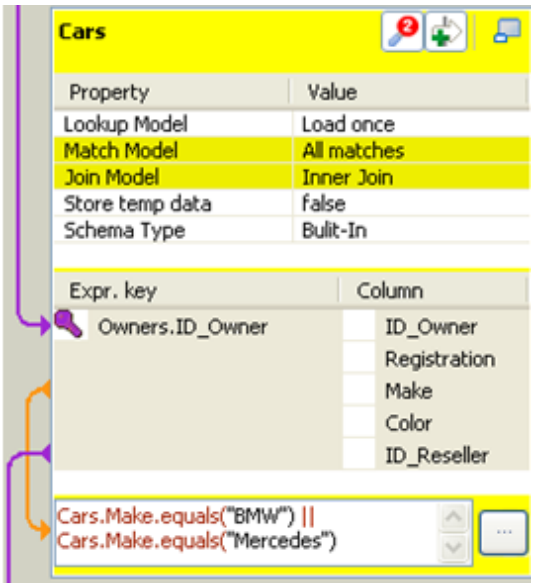
The existing output tables in the **Map Editor** will be automatically renamed according to the connection labels. In this example, relabel the existing output connections *BMW_Mercedes_withChildren* and *Owners_Other_Makes* respectively.

- Double-click the **tMap** component to launch the **Map Editor** to change the mappings and the filters.

  Note that the output area contains a new, empty output table named *Rejects_BMW_Mercedes*. You can adjust the position of the table by selecting it and clicking the **Up** or **Down** arrow button at the top of the output area.

- Remove the **Expr. key** filter (“*BMW*”) from the *Cars* table in the input area.

- Click the **Filters** button to display the **Filter** field, and type in a new filter to limit the search to *BMW* or *Mercedes* car makes. The statement reads as follows: `Cars.Make.equals("BMW") || Cars.Make.equals("Mercedes")`

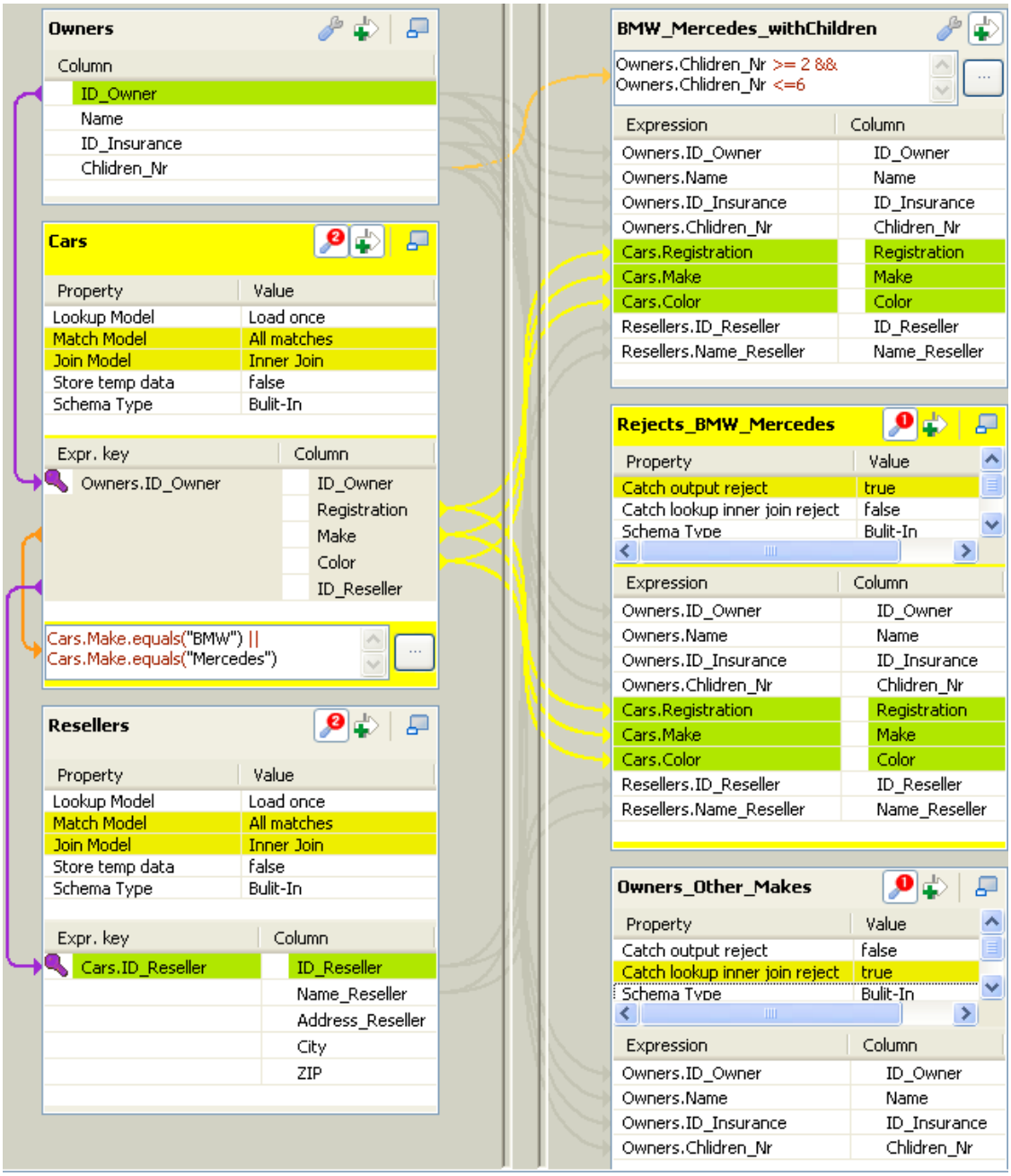


- Select all the columns of the main output table and drop them down to the new output table.

  Alternatively, you can also drag the corresponding columns from the relevant input tables to the new output table.

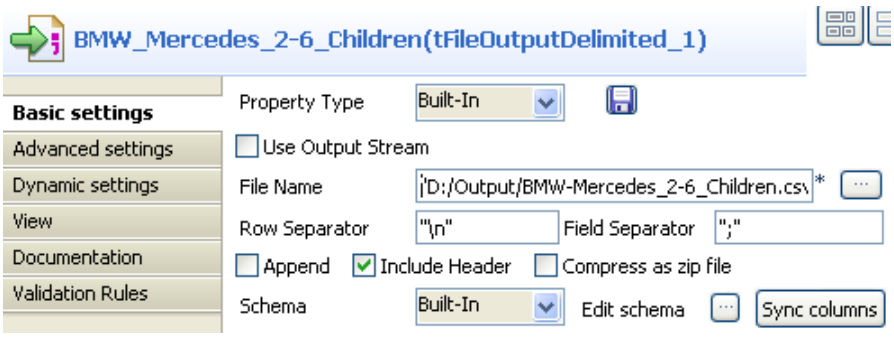
- Click the **tMap settings** button at the top of the new output table and set **Catch output reject** to **true** to collect data about BMW and Mercedes owners who have less than two or more than six children.

- In the *Owners_Other_Makes* table, set **Catch lookup inner join reject** to **true** to collect data about owners of other car makes and owners for whom the reseller information is not found.



- Click **OK** to validate the mappings and close the **Map Editor**.
- Define the properties of the output components in their respective **Basic settings** view.

  In this use case, simple specify the output file paths and select the **Include Header** check box, and leave the other parameters as they are.



- Save the Job and press **F6** to run it.

  The output files contain content of the main output flow shows that the filtered rows have correctly been passed on.

Scenario 6: Advanced mapping with lookup reload at each row

The following scenario describes a Job that retrieves people details from a lookup database, based on a join on the age. The main flow source data is read from a MySQL database table called *people_age* that contains people details such as numeric id, alphanumeric first name and last name and numeric age. The people age is either 40 or 60. The number of records in this table is intentionally restricted.

The reference or lookup information is also stored in a MySQL database table called *large_data_volume*. This lookup table contains a number of records including the city where people from the main flow have been to. For the sake of clarity, the number of records is restricted but, in a normal use, the usefulness of the feature described in the example below is more obvious for very large reference data volume.

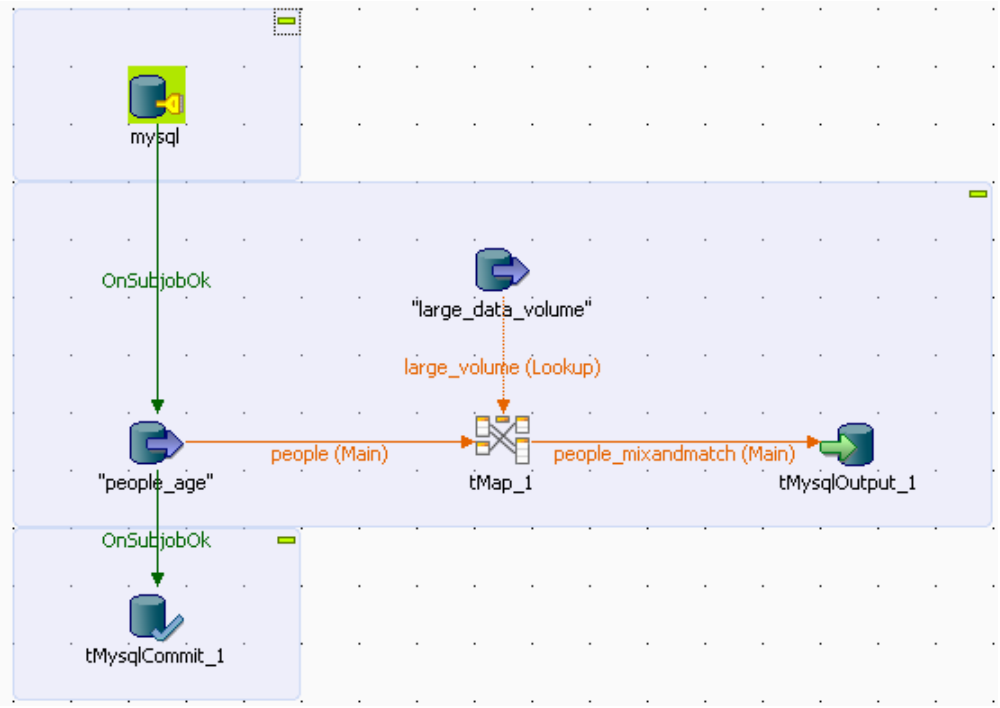
To optimize performance, a database connection component is used in the beginning of the Job to open the connection to the lookup database table in order not to do that every time we want to load a row from the lookup table.

An Expression Filter is applied to this lookup source flow, in order to select only data from people whose age is equal to 60 or 40. This way only the relevant rows from the lookup database table are loaded for each row from the main flow.

Therefore this Job shows how, from a limited number of main flow rows, the lookup join can be optimized to load only results matching the expression key.

Generally speaking, as the lookup loading is performed for each main flow row, this option is mainly interesting when a limited number of rows is processed in the main flow while a large number of reference rows are to be looked up to.

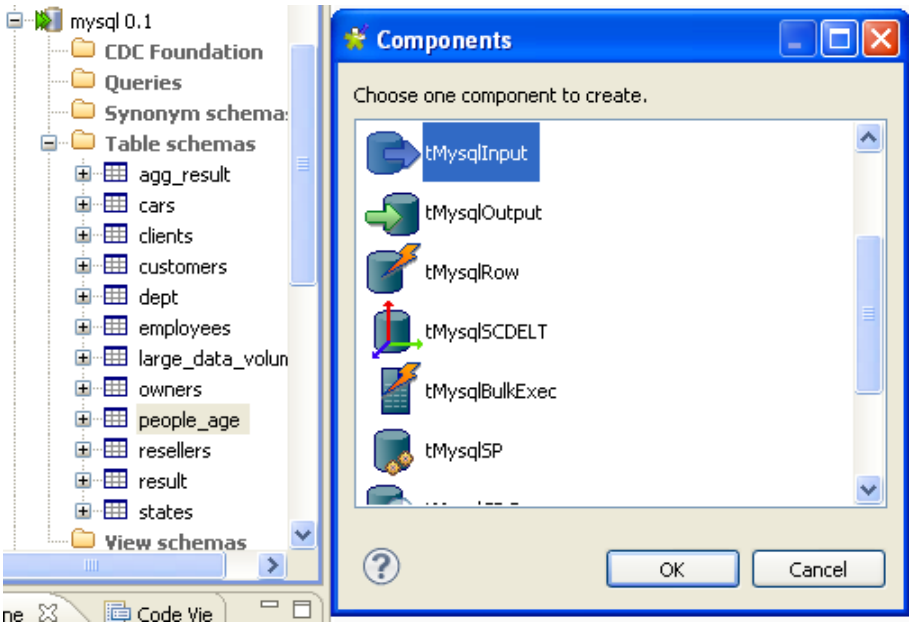
The join is solved on the *age* field. Then, using the relevant loading option in the **tMap** component editor, the lookup database information is loaded for each main flow incoming row.



For this Job, the metadata has been prepared for the source and connection components. For more information on how to set up the DB connection schema metadata, see the relevant section in the *Talend Open Studio for Big Data User Guide*.

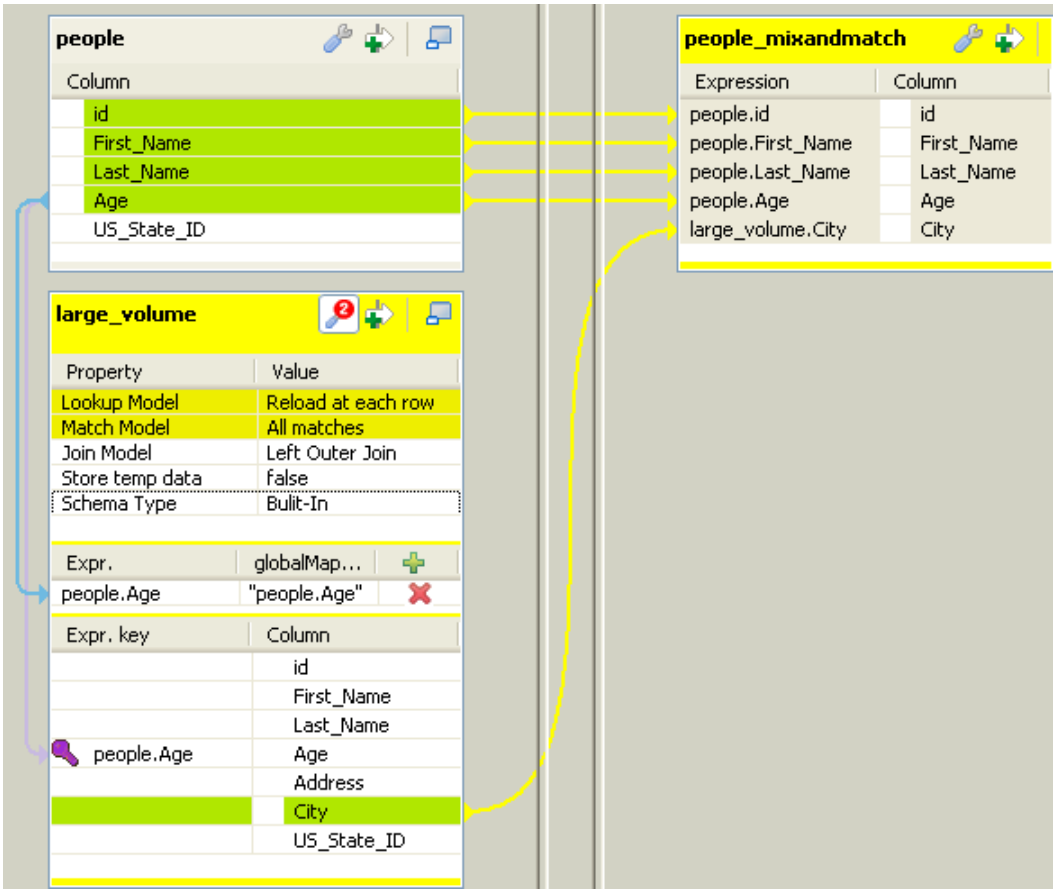
This Job is formed with five components, four database components and a mapping component.

- Drop the DB Connection under the **Metadata** node of the **Repository** to the design workspace. In this example, the source table is called *people_age*.
- Select **tMysqlInput** from the list that pops up when dropping the component.



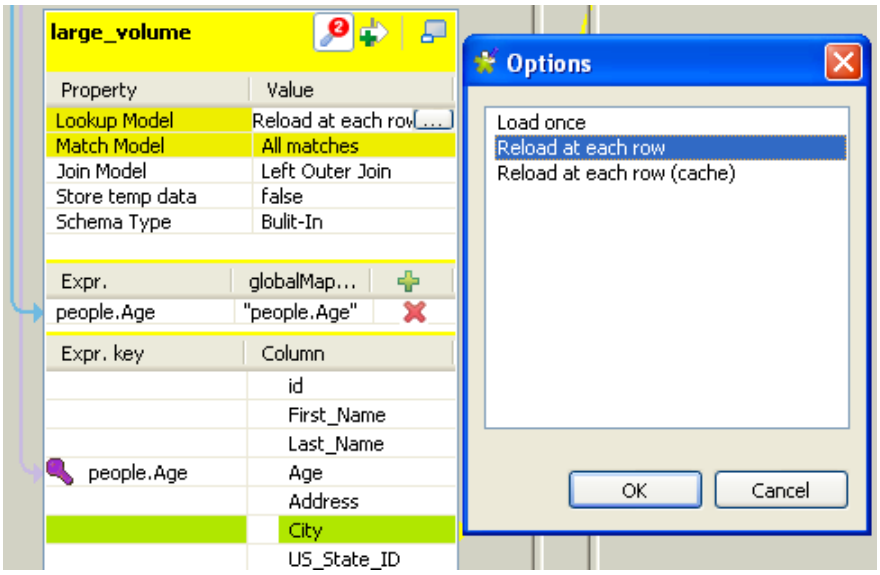
- Drop the lookup DB connection table from the **Metadata** node to the design workspace selecting **tMysqlInput** from the list that pops up. In this Job, the lookup is called *large_data_volume*.

- The same way, drop the DB connection from the **Metadata** node to the design workspace selecting **tMysqlConnection** from the list that pops up. This component creates a permanent connection to the lookup database table in order not to do that every time we want to load a row from the lookup table.
- Then pick the **tMap** component from the **Processing** family, and the **tMysqlOutput** and **tMysqlCommit** components from the **Database** family in the **Palette** to the right hand side of the editor.
- Now connect all the components together. To do so, right-click the **tMysqlInput** component corresponding to the *people* table and drag the link towards **tMap**.
- Release the link over the **tMap** component, the main row flow is automatically set up.
- Rename the **Main** row link to *people*, to identify more easily the main flow data.
- Perform the same operation to connect the lookup table (*large_data_volume*) to the **tMap** component and the **tMap** to the **tMysqlOutput** component.
- A dialog box prompts for a name to the output link. In this example, the output flow is named: *people_mixandmatch.*
- Rename also the lookup row connection link to *large_volume*, to help identify the reference data flow.
- Connect **tMysqlConnection** to **tMysqlInput** using the trigger link **OnSubjobOk**.
- Connect the **tMysqlInput** component to the **tMysqlCommit** component using the trigger link **OnSubjobOk**.
- Then double-click the **tMap** component to open the graphical mapping editor.



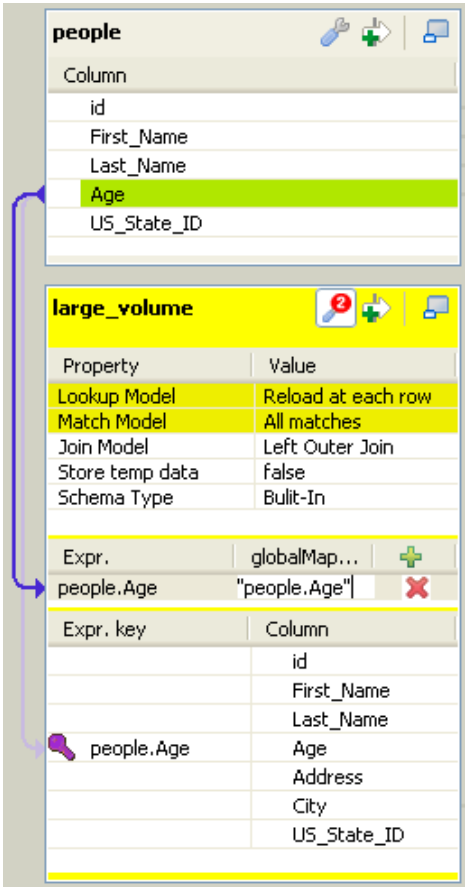
- The **Output** table (that was created automatically when you linked the **tMap** to the **tMySQLOutput** will be formed by the matching rows from the lookup flow (*large_data_volume*) and the main flow (*people_age*).

- Select the main flow rows that are to be passed on to the output and drag them over to paste them in the Output table (to the right hand side of the mapping editor).
- In this example, the selection from the main flow include the following fields: *id*, *first_name*, *last_Name* and *age*.
- From the lookup table, the following column is selected: *city*.
- Drop the selected columns from the input tables (*people* and *large_volume*) to the output table.
- Now set up the join between the main and lookup flows.
- Select the *age* column of the main flow table (on top) and drag it towards the *age* column of the lookup flow table (*large_volume* in this example).
- A key icon appears next to the linked expression on the lookup table. The join is now established.
- Click the **tMap settings** button, click the three-dot button corresponding to **Lookup Model**, and select the **Reload at each row** option from the **[Options]** dialog box in order to reload the lookup for each row being processed.

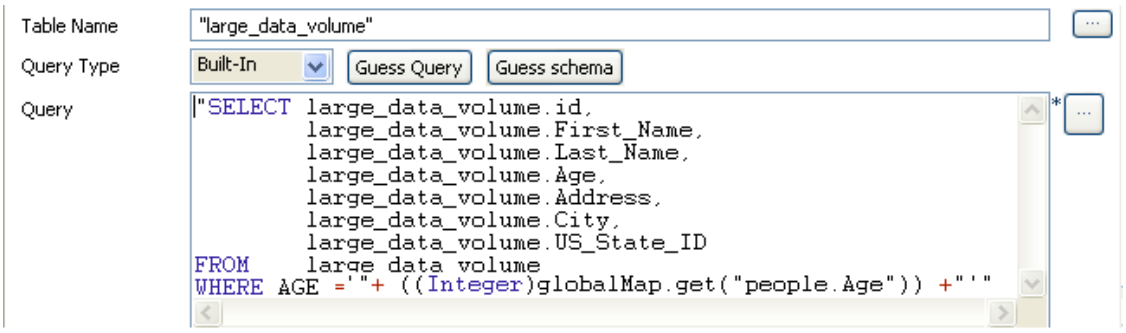


- In the same way, set **Match Model** to **All matches** in the Lookup table, in order to gather all instances of *age* matches in the output flow.
- Now implement the filtering, based on the *age* column, in the Lookup table. The **GlobalMapKey** field is automatically created when you selected the **Reload at each row** option. Indeed you can use this expression to dynamically filter the reference data in order to load only the relevant information when joining with the main flow.

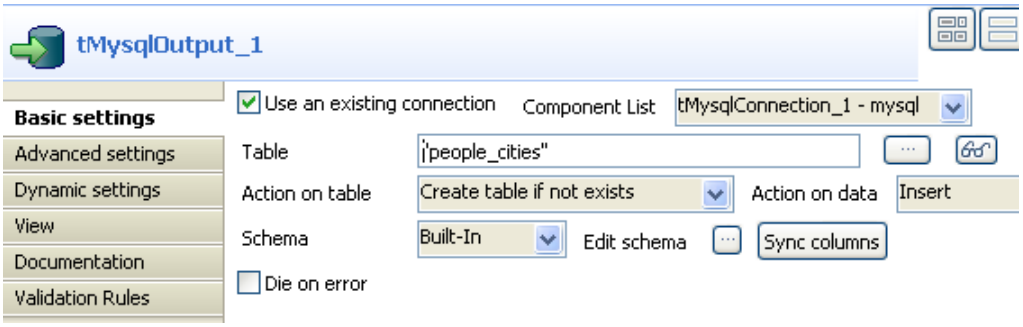
As mentioned in the introduction of the scenario, the main flow data contains only people whose age is either 40 or 60. To avoid the pain of loading all lookup rows, including ages that are different from 40 and 60, you can use the main flow age as global variable to feed the lookup filtering.



- Drop the *Age* column from the main flow table to the **Expr.** field of the lookup table.
- Then in the **globalMap Key** field, put in the variable name, using the expression. In this example, it reads: `“people.Age”`
- Click **OK** to save the mapping setting and go back to the design workspace.
- To finalize the implementation of the dynamic filtering of the lookup flow, you need now to add a WHERE clause in the query of the database input.

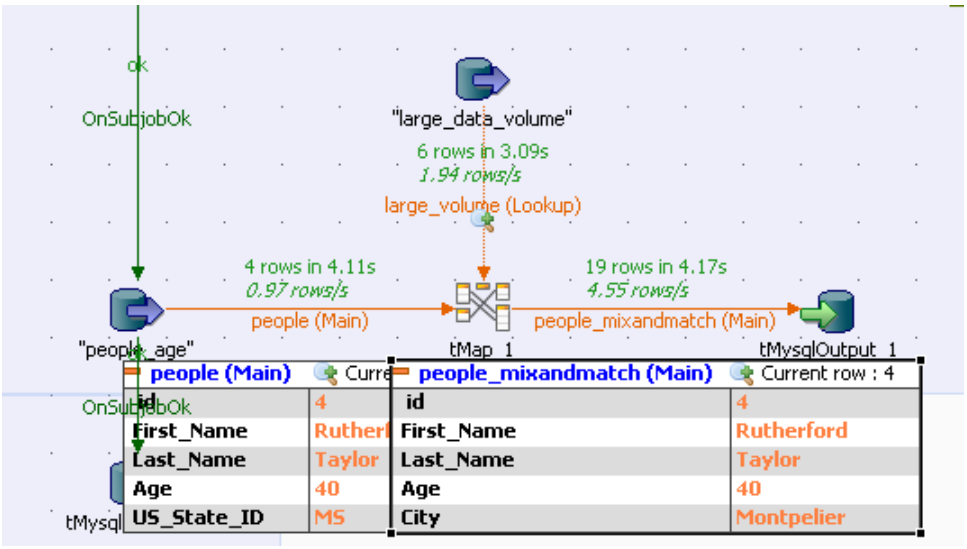


- At the end of the **Query** field, following the `Select` statement, type in the following WHERE clause: `WHERE AGE ='"+((Integer)globalMap.get("people.Age"))+"'"`
- Make sure that the type corresponds to the column used as variable. In this use case, *Age* is of *Integer* type. And use the variable the way you set in the **globalMap key** field of the map editor.
- Double-click the **tMysqloutput** component to define its properties.



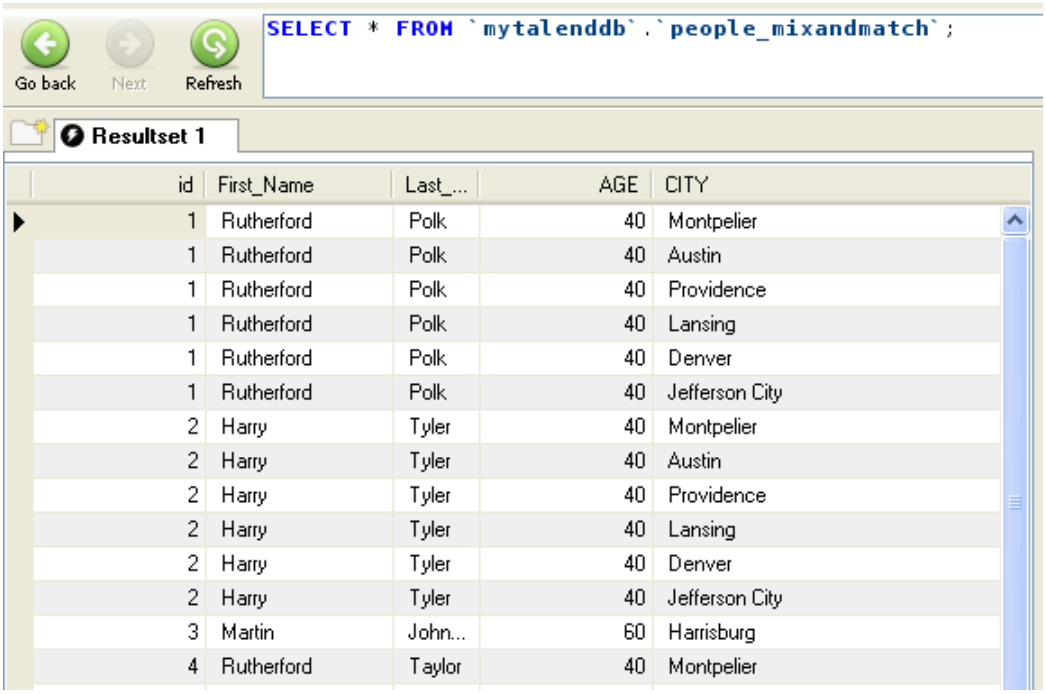
- Select the **Use an existing connection** check box to leverage the created DB connection.
- Define the target table name and relevant DB actions.
- Click the **Run** tab at the bottom of the design workspace, to display the Job execution tab.
- From the **Debug Run** view, click the **Traces Debug** button to view the data processing progress.

For more comfort, you can maximize the Job design view while executing by simply double-clicking on the Job name tab.



The lookup data is reloaded for each of the main flow's rows, corresponding to the age constraint. All *age* matches are retrieved in the lookup rows and grouped together in the output flow.

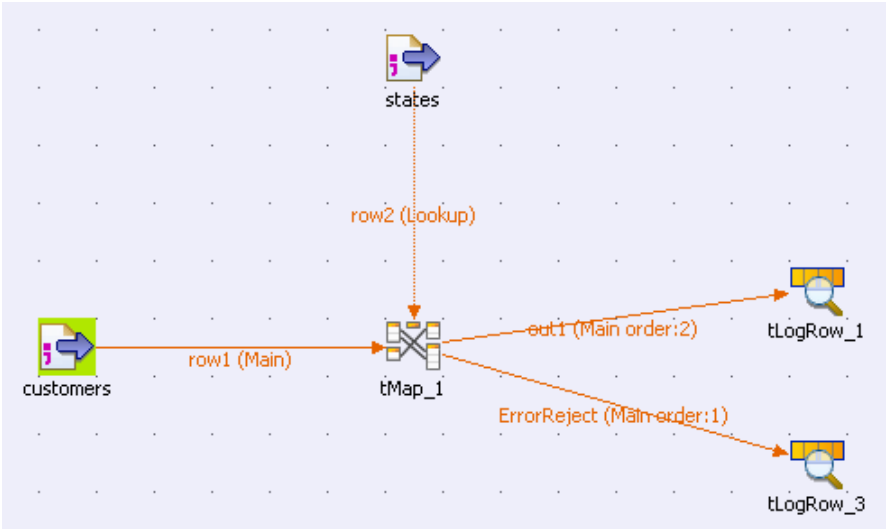
Therefore if you check out the data contained in the newly created *people_mixandmatch* table, you will find all the *age* duplicates corresponding to different individuals whose age equals to 60 or 40 and the city where they have been to.



id	First_Name	Last_...	AGE	CITY
1	Rutherford	Polk	40	Montpelier
1	Rutherford	Polk	40	Austin
1	Rutherford	Polk	40	Providence
1	Rutherford	Polk	40	Lansing
1	Rutherford	Polk	40	Denver
1	Rutherford	Polk	40	Jefferson City
2	Harry	Tyler	40	Montpelier
2	Harry	Tyler	40	Austin
2	Harry	Tyler	40	Providence
2	Harry	Tyler	40	Lansing
2	Harry	Tyler	40	Denver
2	Harry	Tyler	40	Jefferson City
3	Martin	John...	60	Harrisburg
4	Rutherford	Taylor	40	Montpelier

Scenario 7: Mapping with join output tables

The following scenario describes a Job that processes reject flows without separating them from the main flow.



- In the **Repository** tree view, click **Metadata** > **File delimited**. Drag and drop the *customers* metadata onto the workspace.

  The *customers* metadata contains information about customers, such as their ID, their name or their address, etc.

  For more information about centralizing metadata, see *Talend Open Studio for Big Data User Guide*.

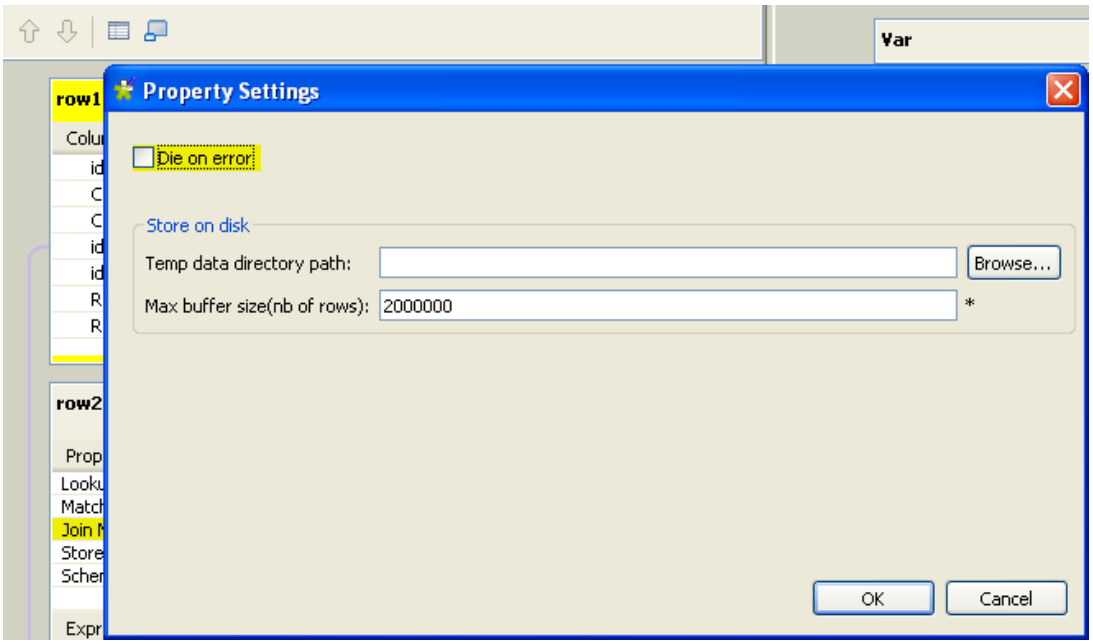
- In the dialog box that asks you to choose which component type you want to use, select **tFileInputDelimited** and click **OK**.

- Drop the *states* metadata onto the design workspace. Select the same component in the dialog box and click **OK**.

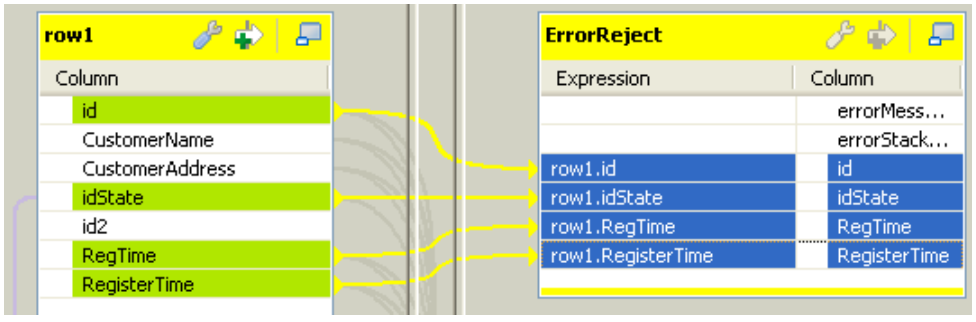
  The *states* metadata contains the ID of the state, and its name.

- Drop a **tMap** and two **tLogRow** components from the **Palette** onto the design workspace.
- Connect the *customers* component to the **tMap**, using a **Row > Main** connection.
- Connect the *states* component to the **tMap**, using a **Row > Main** connection. This flow will automatically be defined as **Lookup**.
- Double-click the **tMap** component to open the **Map Editor**.
- Drop the *idState* column from the main input table to the *idState* column of the lookup table to create a join.
- Click the **tMap settings** button and set **Join Model** to **Inner Join**.
- Click the **Property Settings** button at the top of the input area to open the **[Property Settings]** dialog box, and clear the **Die on error** check box in order to handle the execution errors.

  The **ErrorReject** table is automatically created.



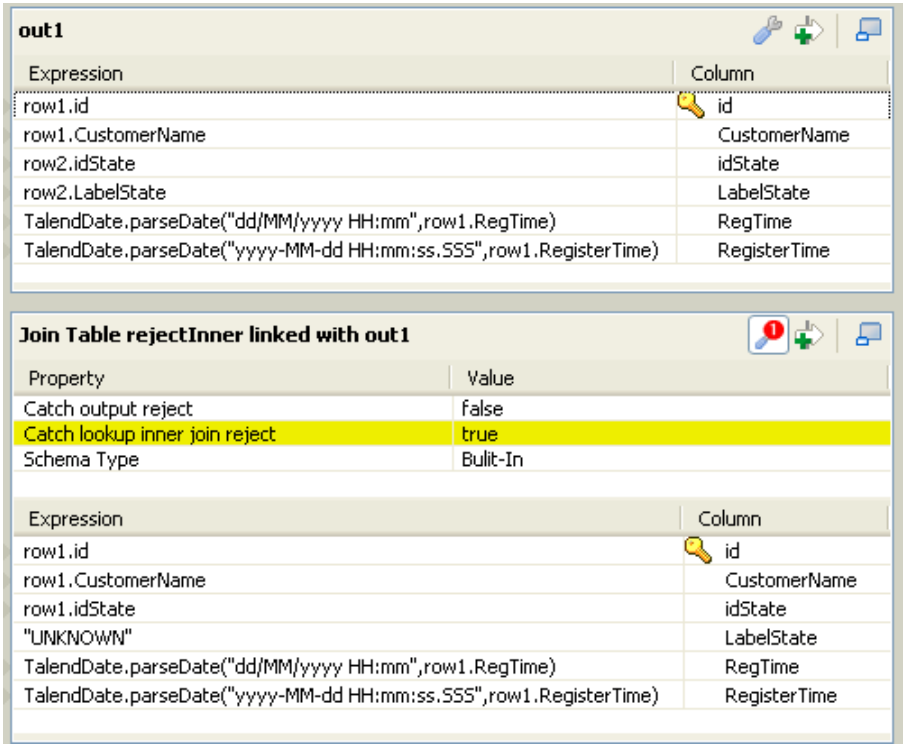
- Select the *id*, *idState*, *RegTime* and *RegisterTime* in the input table and drag them to the **ErrorReject** table.



- Click the **[+]** button at the top right of the editor to add an output table. In the dialog box that opens, select **New output**. In the field next to it, type in the name of the table, *out1*. Click **OK**.
- Drag the following columns from the input tables to the *out1* table: *id*, *CustomerName*, *idState*, and *LabelState*.
- Add two columns, *RegTime* and *RegisterTime*, to the end of the *out1* table and set their date formats: `"dd/MM/yyyy HH:mm"` and `"yyyy-MM-dd HH:mm:ss.SSS"` respectively.

- Click in the **Expression** field for the *RegTime* column, and press **Ctrl+Space** to display the auto-completion list. Find and double-click `TalendDate.parseDate`. Change the pattern to `("dd/MM/yyyy HH:mm",row1.RegTime)`.

- Do the same thing for the *RegisterTime* column, but change the pattern to `("yyyy-MM-dd HH:mm:ss.SSS",row1.RegisterTime)`.



- Click the **[+]** button at the top of the output area to add an output table. In the dialog box that opens, select **Create join table from**, choose *Out1*, and name it *rejectInner*. Click **OK**.

- Click the **tMap settings** button and set **Catch lookup inner join reject** to **true** in order to handle rejects.

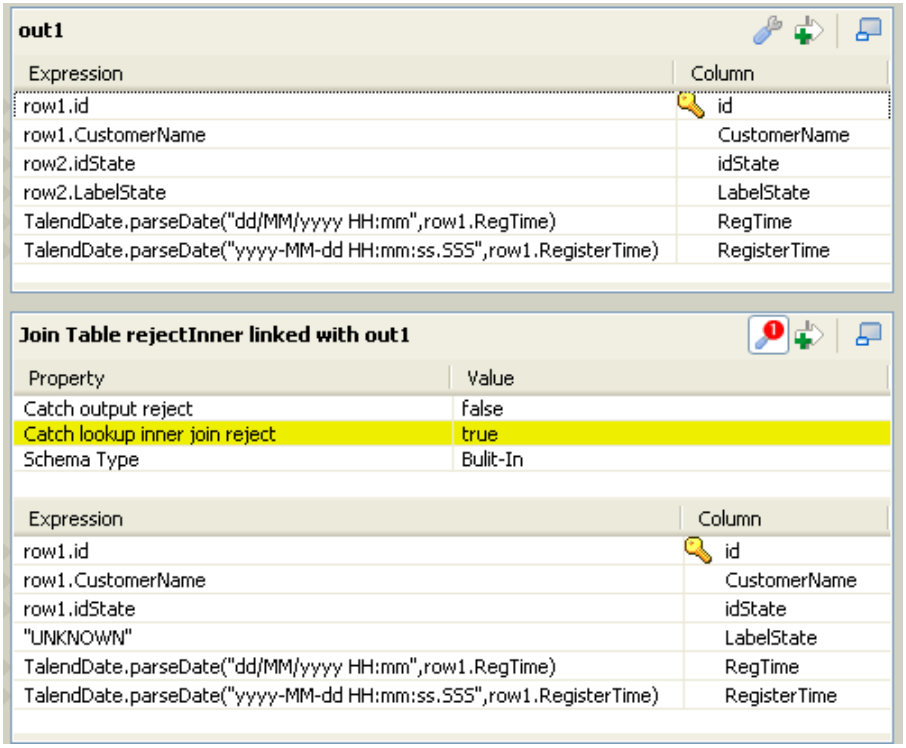
- Drag the *id*, *CustomerName*, and *idState* columns from the input tables to the corresponding columns of the *rejectInner* table.

- Click in the **Expression** field for the *LabelState* column, and type in `"UNKNOWN"`.

- Click in the **Expression** field for the *RegTime* column, press **Ctrl+Space**, and select `TalendDate.parseDate`. Change the pattern to `("dd/MM/yyyy HH:mm",row1.RegTime)`.

- Click in the **Expression** field for the *RegisterTime* column, press **Ctrl+Space**, and select `TalendDate.parseDate`, but change the pattern to `("yyyy-MM-dd HH:mm:ss.SSS",row1.RegisterTime)`.

If the data from *row1* has a wrong pattern, it will be returned by the ErrorReject flow.



- Click **OK** to validate the changes and close the editor.
- Double-click the first **tLogRow** component to display its **Component** view.
- Click **Sync columns** to retrieve the schema structure from the mapper if needed.
- In the **Mode** area, select **Table.**
- Do the same thing with the second **tLogRow**.
- Save your Job and press **F6** to execute it.

  The **Run** console displays the main out flow and the ErrorReject flow. The main output flow unites both valid data and inner join rejects, while the ErrorReject flow contains the error information about rows with unparseable date formats.

```
+------------------------------+-------+--------------+----------------+----------------------+
|                                      tLogRow_1                                              |
+------------------------------+-------+--------------+----------------+----------------------+=|
|CustomerName                  |idState|LabelState    |RegTime         |RegisterTime          |
+------------------------------+-------+--------------+----------------+----------------------+=|
|Griffith Paving and Sealcoatin|7      |Connecticut   |03/11/2006 09:20|2001-01-17 06:26:40.000|
|Bill's Dive Shop              |35     |Ohio          |19/11/2004 15:48|2002-06-07 09:40:00.000|
|Childress Child Day Care      |60     |UNKNOWN       |16/02/2005 08:27|1990-04-01 21:00:00.000|
|Facelift Kitchen and Bath     |0      |UNKNOWN       |22/08/2002 09:55|1972-04-23 18:00:00.000|
|Terrinni & Son Auto and Truck |5      |California    |28/06/2001 09:15|1982-04-19 10:26:40.000|
|Kermit the Pet Shop           |28     |Nevada        |17/08/2003 10:07|2006-05-27 17:00:00.000|
|Mubin Furniture Store         |15     |Iowa          |27/08/2000 02:13|1970-02-27 22:08:16.000|
```

tNormalize

tNormalize Properties

Component family	Processing/Fields	
Function	Normalizes the input flow following SQL standard.	
Purpose	**tNormalize** helps improve data quality and thus eases the data update.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**..
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Column to normalize	Select the column from the input flow which the normalization is based on.
	Item separator	Enter the separator which will delimit data in the input flow.
Advanced settings	*Get rid of duplicated rows from output*	Select this check box to deduplicate rows in the data of the output flow.
	Use CSV parameters	Select this check box to include CSV specific parameters such as **escape mode** and **enclosure** character.
	Discard the trailing empty strings	Select this check box to discard the trailing empty strings.
	Trim resulting values	Select this check box to trim leading and trailing whitespace from the resulting data. When both **Discard the trailing empty string** and **Trim resulting values** check boxes are selected, the former works first.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

Scenario: Normalizing data

This simple scenario illustrates a Job that normalizes a list of tags for Web forum topics, and displays the result in a table on the **Run** console.

This list is not well organized and it contains trailing empty strings, leading and trailing whitespace, and repeated tags, as shown below.

```
ldap,
  db2, jdbc driver,
grid computing,  talend architecture  ,
```

```
content, environment,,
tmap,,
eclipse,
database,java,postgresql,
tmap,
database,java,sybase,
deployment,,
repository,
database,informix,java
```

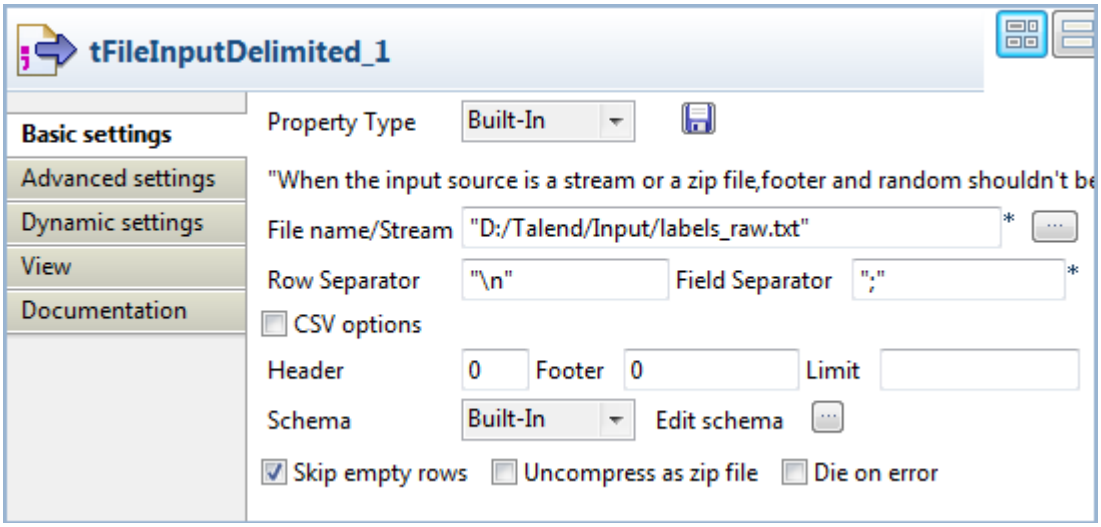
Setting up the Job

1. Drop the following components from the **Palette** to the design workspace: **tFileInputDelimited**, **tNormalize**, **tLogRow**.

2. Connect the components using **Row** > **Main** connections.

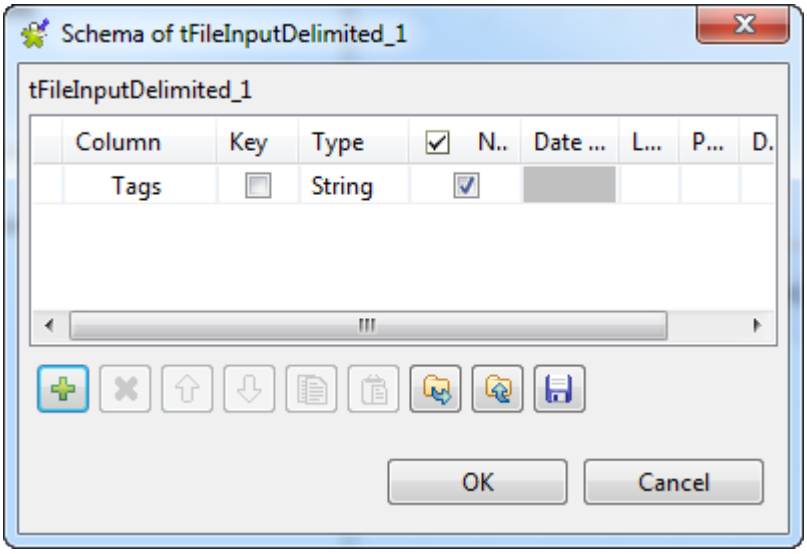
Configuring the components

1. Double-click the **tFileInputDelimited** component to open its **Basic settings** view.

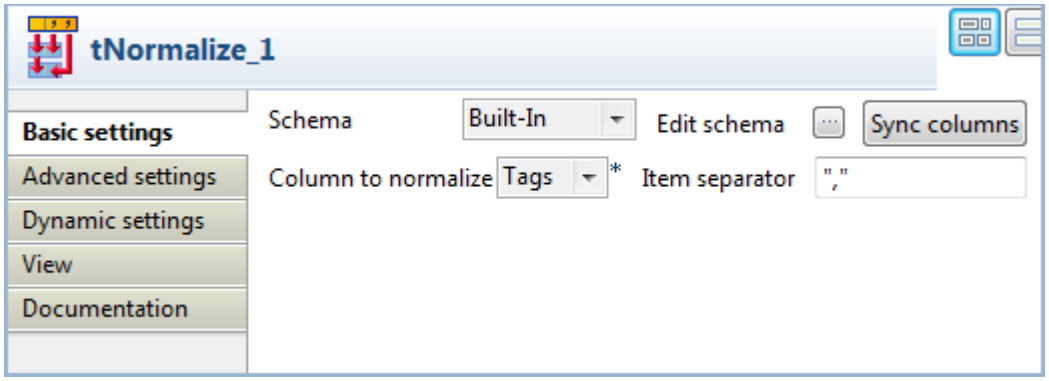


2. In the **File name** field, specify the path to the input file to be normalized.

3. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box, and set up the input schema by adding one column named *Tags*. When done, click **OK** to validate your schema setup and close the dialog box, leaving the rest of the settings as they are.



4. Double-click the **tNormalize** component to open **Basic settings** view.

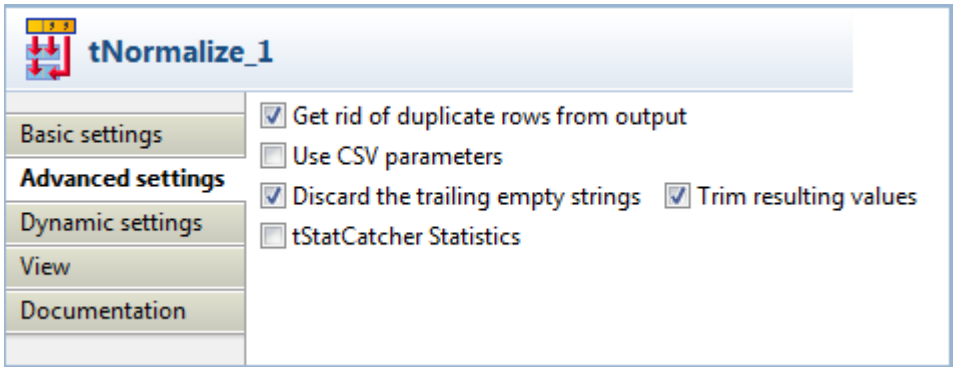


5. Check the schema, and if necessary, click **Sync columns** to get the schema synchronized with the input component.

6. Define the column the normalization operation is based on.

   In this use case, the input schema has only one column, *Tags*, so just accept the default setting.

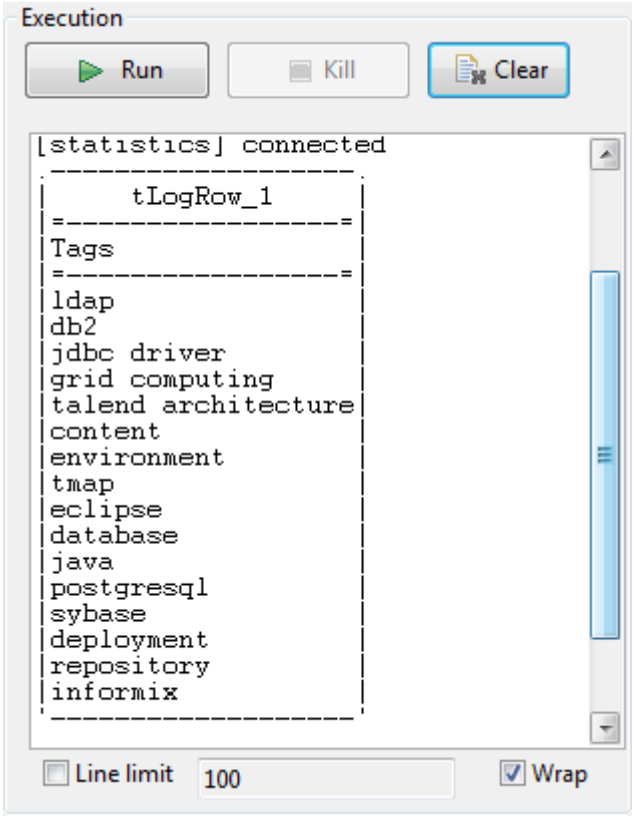
7. In the **Advanced settings** view, select the **Get rid of duplicate rows from output**, **Discard the trailing empty strings**, and **Trim resulting values** check boxes.



8. In the **tLogRow** component, select the **Print values in the cells of table** radio button.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job.
2. Click **Run** on the **Run** tab or press **F6** to execute the Job.



The list is tidied up, with duplicate tags, leading and trailing whitespace and trailing empty strings removed, and the result is displayed in a table cell on the console.

tPigAggregate

tPigAggregate component belongs to two component families: Big Data and Processing. For more information about **tPigAggregate**, see section *tPigAggregate*.

tPigCode

tPigCode component belongs to two component families: Big Data and Processing. For more information about **tPigCode**, see section *tPigCode*.

tPigCross

tPigCross component belongs to two component families: Big Data and Processing. For more information about **tPigCross**, see section *tPigCross*.

tPigDistinct

tPigDistinct component belongs to two component families: Big Data and Processing. For more information about **tPigDistinct**, see section *tPigDistinct*.

tPigFilterColumns

tPigFilterColumns component belongs to two component families: Big Data and Processing. For more information about **tPigFilterColumns**, see section *tPigFilterColumns*.

tPigFilterRow

tPigFilterRow component belongs to two component families: Big Data and Processing. For more information about **tPigFilterRow**, see section *tPigFilterRow*.

tPigJoin

tPigJoin component belongs to two component families: Big Data and Processing. For more information about **tPigJoin**, see section *tPigJoin*.

tPigLoad

tPigLoad component belongs to two component families: Big Data and Processing. For more information about **tPigLoad**, see section *tPigLoad*.

tPigSort

tPigSort component belongs to two component families: Big Data and Processing. For more information about **tPigSort**, see section *tPigSort*.

tPigStoreResult

tPigStoreResult component belongs to two component families: Big Data and Processing. For more information about **tPigStoreResult**, see section *tPigStoreResult*.

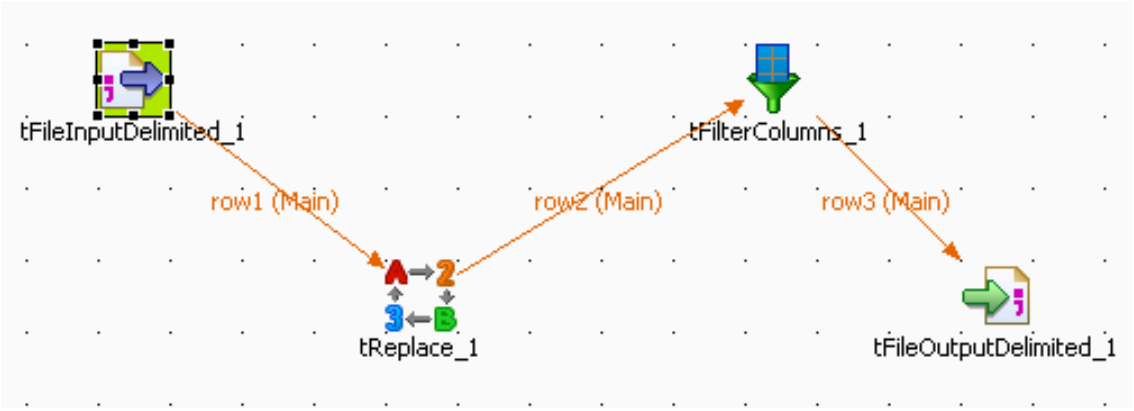
tReplace

tReplace Properties

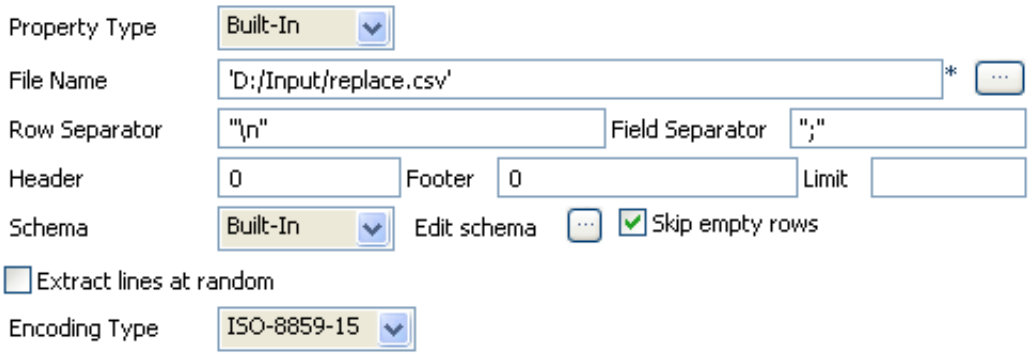
Component family	Processing	
Function	Carries out a Search & Replace operation in the input columns defined.	
Purpose	Helps to cleanse all files before further processing.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Two read-only columns, Value and Match are added to the output schema automatically.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Simple Mode Search / Replace	Click Plus to add as many conditions as needed. The conditions are performed one after the other for each row. **Input column**: Select the column of the schema the search & replace is to be operated on **Search**: Type in the value to search in the input column **Replace with**: Type in the substitution value. **Whole word**: Select this check box if the searched value is to be considered as whole. **Case sensitive**: Select this check box to care about the case. Note that you cannot use regular expression in these columns.
	Use advanced mode	Select this check box when the operation you want to perform cannot be carried out through the simple mode. In the text field, type in the regular expression as required.
Usage	This component is not startable as it requires an input flow. And it requires an output component.	

Scenario: multiple replacements and column filtering

This following Job searches and replaces various typos and defects in a csv file then operates a column filtering before producing a new csv file with the final output.



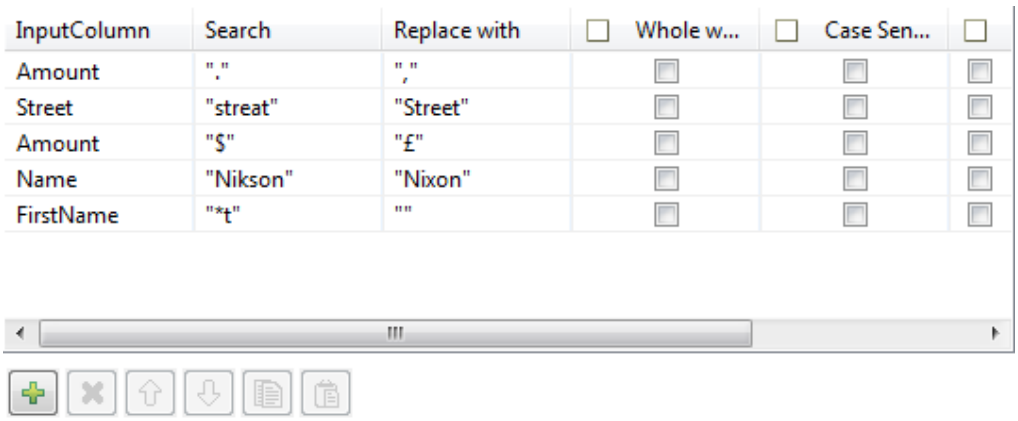
- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tReplace**, **tFilterColumn** and **tFileOutputDelimited**.
- Connect the components using **Main Row** connections via a right-click each component.
- Select the **tFileInputDelimited** component and set the input flow parameters.



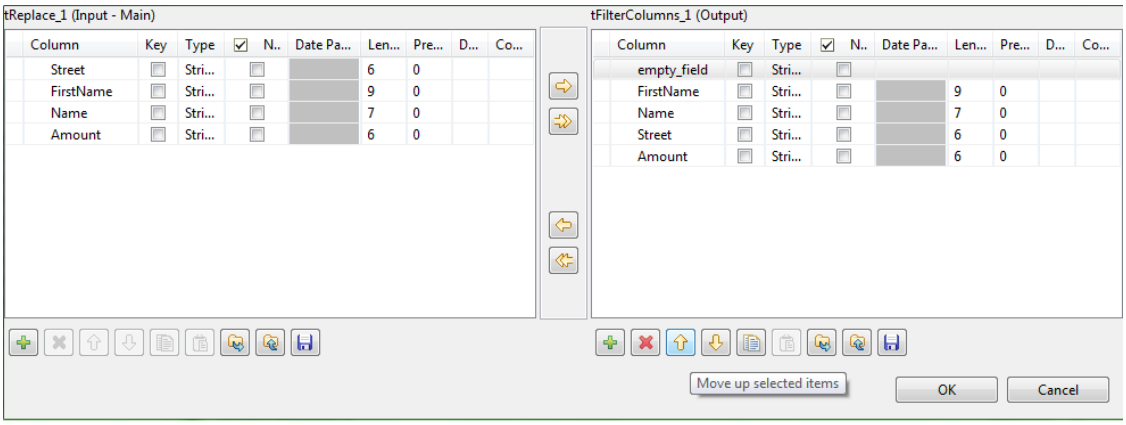
- The **Property type** for this scenario is **Built-in**. Therefore the following fields are to be set manually unlike the Properties stored centrally in the repository, that are retrieved automatically.
- The **File** is a simple csv file stored locally. The **Row Separator** is a carriage return and the **Field Separator** is a semi-colon. In the **Header** is the name of the column, and no **Footer** nor **Limit** are to be set.
- The file contains characters such as: `*t, .` or `Nikson` which we want to turn into `Nixon`, and `streat`, which we want to turn into `Street`.

Street	FirstName	Name	Amount
streat	John	Kennedy	98.30$
streat	Richad	Nikson	78.23$
streat	Richard	Nikson	78.2$
streat	toto	Nikson	78.23$
streat	Richard	Nikson	78.23$
street	Georges *t	bush	99.99$

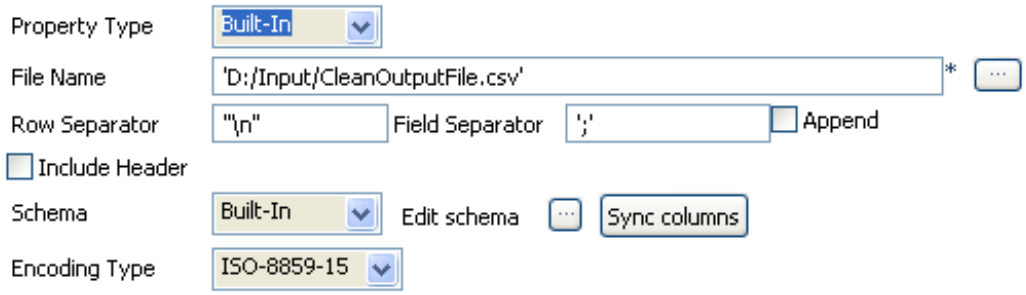
- The schema for this file is built in also and made of four columns of various types (string or int).
- Now select the **tReplace** component to set the search & replace parameters.



- The schema can be synchronized with the incoming flow.
- Select the **Simple mode** check box as the search parameters can be easily set without requiring the use of regexp.
- Click the plus sign to add some lines to the parameters table.
- On the first parameter line, select *Amount* as **InputColumn**. Type *"."* in the **Search** field, and *","* in the **Replace** field.
- On the second parameter line, select *Street* as **InputColumn**. Type *"streat"* in the **Search** field, and *"Street"* in the **Replace** field.
- On the third parameter line, select again *Amount* as **InputColumn**. Type *"$"* in the **Search** field, and *"£"* in the **Replace** field.
- On the fourth paramater line, select *Name* as **InputColumn**. Type *"Nikson"* in the **Search** field, and *"Nixon"* in the **Replace** field.
- On the fifth parameter line, select *Firstname* as **InputColumn**. Type *"*t"* in the **Search** field, and replace them with nothing between double quotes.
- The advanced mode isn't used in this scenario.
- Select the next component in the Job, **tFilterColumn**.



- The **tFilterColumn** component holds a schema editor allowing to build the output schema based on the column names of the input schema. In this use case, add one new column named *empty_field* and change the order of the input schema columns to obtain a schema as follows: *empty_field, Firstname, Name, Street, Amount*.
- Click **OK** to validate.



- Set the **tFileOutputDelimited** properties manually.
- The schema is built-in for this scenario, and comes from the preceding component in the Job.
- Save the Job and press **F6** to execute it.

	John	Kennedy	Street	98,30£
	Richad	Nixon	Street	78,23£
	Richard	Nixon	Street	78,2£
	toto	Nixon	Street	78,23£
	Richard	Nixon	Street	78,23£
	Georges	bush	street	99,99£

The first column is empty, the rest of the columns have been cleaned up from the parasitical characters, and *Nikson* was replaced with *Nixon*. The street column was moved and the decimal delimiter has been changed from a dot to a comma, along with the currency sign.

tSampleRow

tSampleRow properties

Component family	Processing	
Function	**tSampleRow** filters rows according to line numbers.	
Purpose	**tSampleRow** helps to select rows according to a list of single lines and/or a list of groups of lines.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Range	Enter a range using the relevant syntax to choose a list of single lines and/or a list of groups of lines.
Usage	This component handles flows of data therefore it requires input and output components.	
Limitation	n/a	

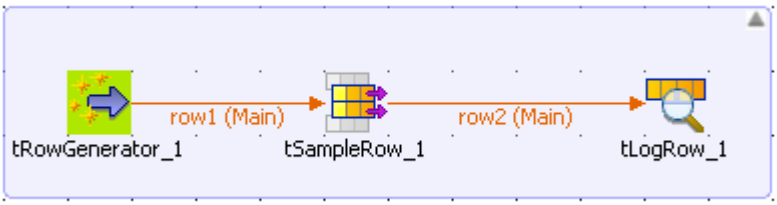
Scenario: Filtering rows and groups of rows

This scenario describes a three-component Job. A **tRowGenerator** is used to create random entries which are directly sent to a **tSampleRow** where they will be filtered according to a defined range. In this scenario, we suppose the input flow contains names of salespersons along with their respective number of sold products and their years of presence in the enterprise. The result of the filtering operation is displayed on the **Run** console.

Dropping and linking the components

1. Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tSampleRow**, and **tLogRow**.

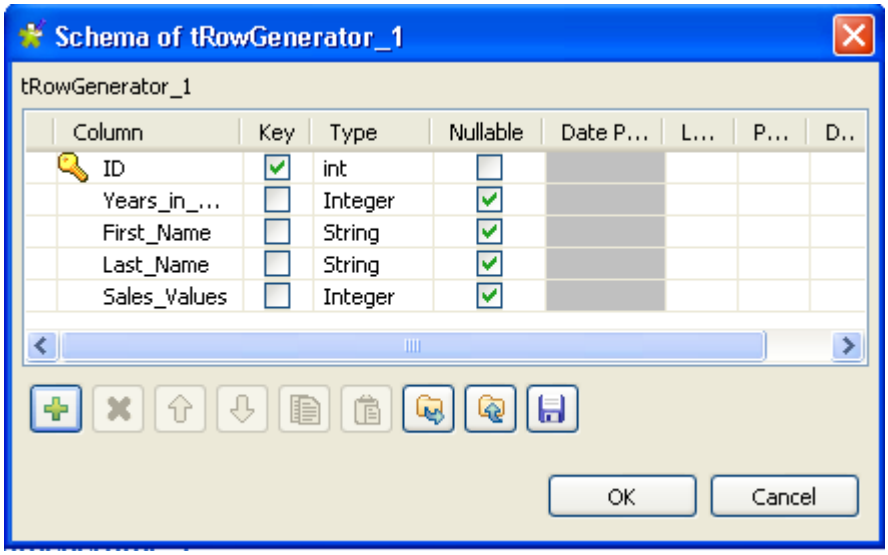
2. Connect the three components using **Row** > **Main** links.



Configuring the components

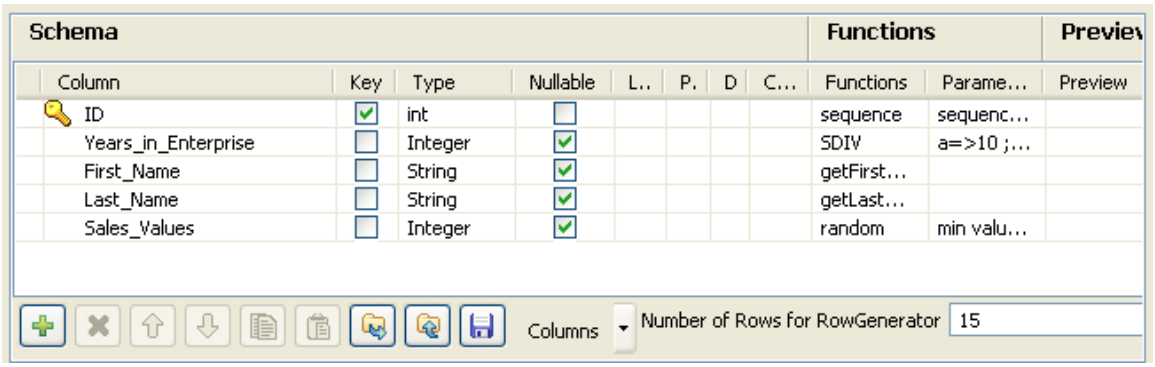
1. In the design workspace, select **tRowgenerator**, and click the **Component** tab to define the basic settings for **tRowGenerator**.

2. Click the **[...]** button next to **Edit Schema** to define the data you want to use as input. In this scenario, the schema is made of five columns.

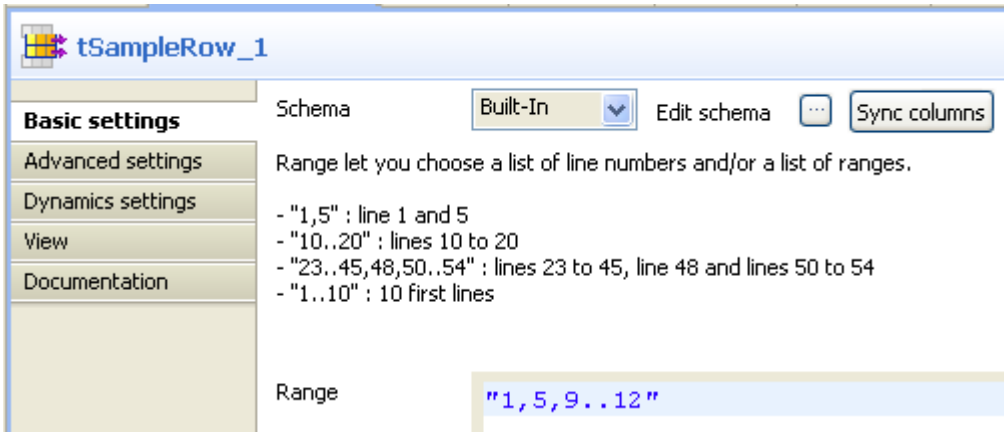


3. In the **Basic settings** view, click **RowGenerator Editor** to define the data to be generated.

4. In the **RowGenerator Editor**, specify the number of rows to be generated in the **Number of Rows for RowGenerator** field and click **OK**. The **RowGenerator Editor** closes.



5. In the design workspace, select **tSampleRow** and click the **Component** tab to define the basic settings for **tSampleRow**.



6. In the **Basic settings** view, set the **Schema** to **Built-In** and click **Sync columns** to retrieve the schema from the **tRowGenerator** component.

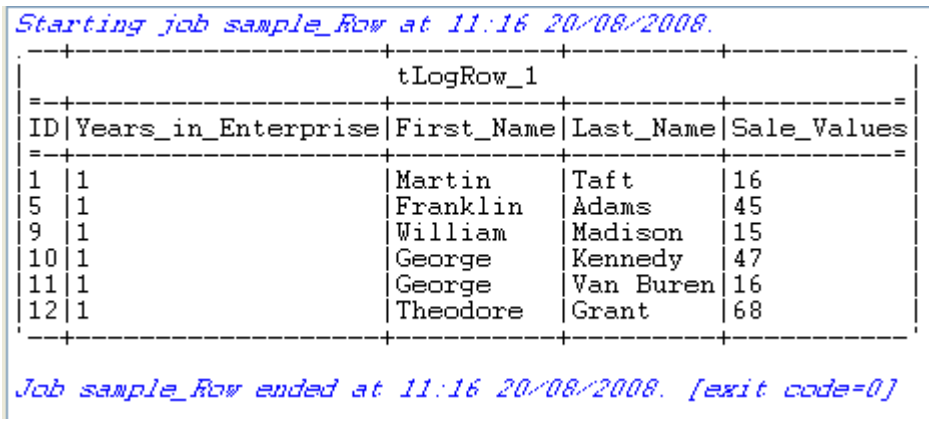
7. In the **Range** panel, set the filter to select your rows using the correct syntax as explained. In this scenario, we want to select the first and fifth lines along with the group of lines between 9 and 12.

8. In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow**, see section *tLogRow*.

Saving and execting the Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6**, or click **Run** on the **Run** tab to execute the Job.



The filtering result displayed on the console shows the first and fifth rows and the group of rows between 9 and 12.

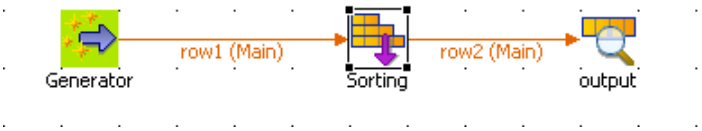
tSortRow

tSortRow properties

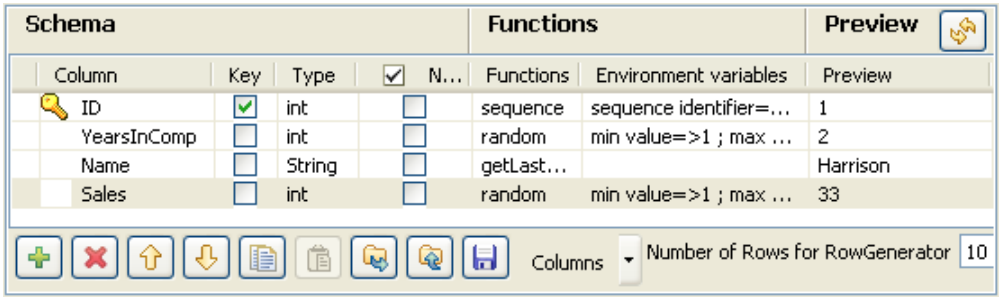
Component family	Processing	
Function	Sorts input data based on one or several columns, by sort type and order	
Purpose	Helps creating metrics and classification table.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Criteria	Click + to add as many lines as required for the sort to be complete. By default the first column defined in your schema is selected.
		Schema column: Select the column label from your schema, which the sort will be based on. Note that the order is essential as it determines the sorting priority.
		Sort type: Numerical and Alphabetical order are proposed. More sorting types to come.
		Order: Ascending or descending order.
Advanced settings	Sort on disk	Customize the memory used to temporarily store output data. **Temp data directory path**: Set the location where the temporary files should be stored. **Create temp data directory if not exists**: Select this checkbox to create the directory if it does not exist. **Buffer size of external sort**: Type in the size of physical memory you want to allocate to sort processing.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Scenario 1: Sorting entries

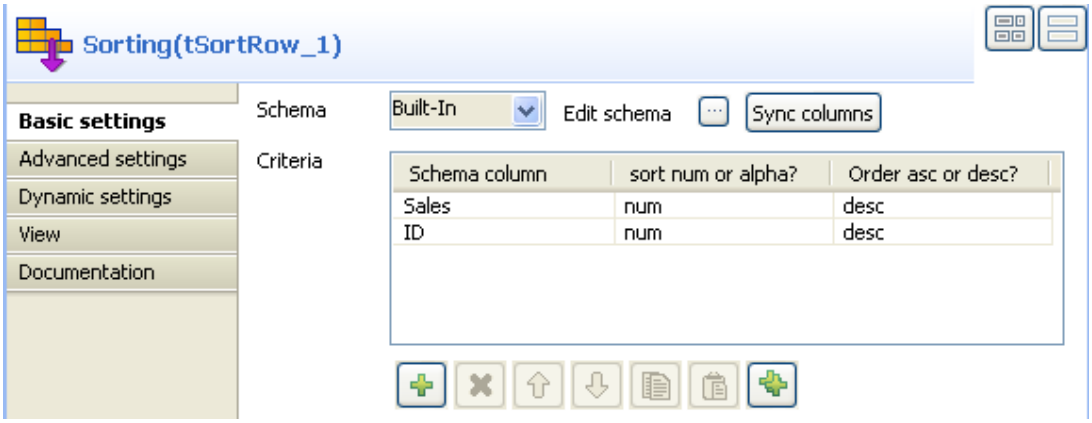
This scenario describes a three-component Job. A **tRowGenerator** is used to create random entries which are directly sent to a **tSortRow** to be ordered following a defined value entry. In this scenario, we suppose the input flow contains names of salespersons along with their respective sales and their years of presence in the company. The result of the sorting operation is displayed on the **Run** console.



- Drop the three components required for this use case: **tRowGenerator**, **tSortRow** and **tLogRow** from the **Palette** to the design workspace.
- Connect them together using **Row main** links.
- On the **tRowGenerator** editor, define the values to be randomly used in the Sort component. For more information regarding the use of this particular component, see section *tRowGenerator*



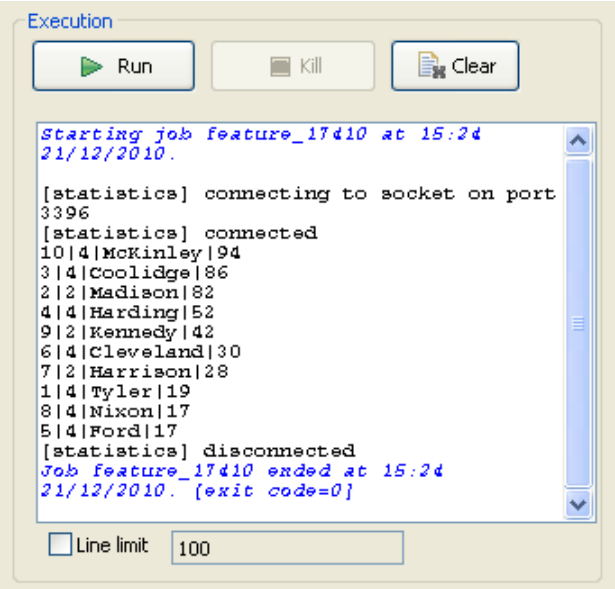
- In this scenario, we want to rank each salesperson according to its *Sales* value and to its number of years in the company.
- Double-click **tSortRow** to display the **Basic settings** tab panel. Set the sort priority on the Sales value and as secondary criteria, set the number of years in the company.



- Use the plus button to add the number of rows required. Set the type of sorting, in this case, both criteria being integer, the sort is numerical. At last, given that the output wanted is a rank classification, set the order as descending.
- Display the **Advanced Settings** tab and select the **Sort on disk** check box to modify the temporary memory parameters. In the **Temp data directory path** field, type the path to the directory where you want to store the temporary data. In the **Buffer size of external sort** field, set the maximum buffer value you want to allocate to the processing.

The default buffer value is 1000000 but the more rows and/or columns you process, the higher the value needs to be to prevent the Job from automatically stopping. In that event, an "out of memory" error message displays.

- Make sure you connected this flow to the output component, **tLogRow**, to display the result in the Job console.
- Press **F6** to run the Job. The ranking is based first on the Sales value and then on the number of years of experience.


Execution
Run
Kill
Clear
Starting job feature_17d10 at 15:2d 21/12/2010.
[statistics] connecting to socket on port 3396
[statistics] connected
10|4|McKinley|94
3|4|Coolidge|86
2|2|Madison|82
4|4|Harding|52
9|2|Kennedy|42
6|4|Cleveland|30
7|2|Harrison|28
1|4|Tyler|19
8|4|Nixon|17
5|4|Ford|17
[statistics] disconnected
Job feature_17d10 ended at 15:2d 21/12/2010. [exit code=0]
Line limit
100

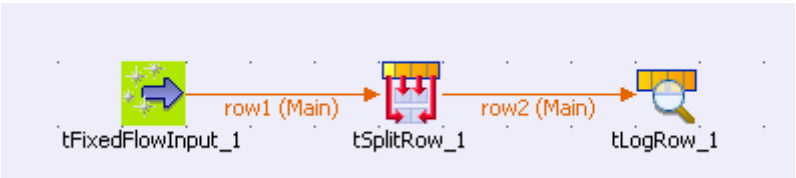
tSplitRow

tSplitRow properties

Component family	Processing/Fields	
Function	**tSplitRow** splits one row into several rows.	
Purpose	This component helps splitting one input row into several output rows.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Columns mapping	Click the plus button to add as many lines as needed by mappings from input columns onto output columns.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	This component splits one input row into multiple output rows by mapping input columns onto output columns.	
Limitation	n/a	

Scenario 1: Splitting one row into two rows

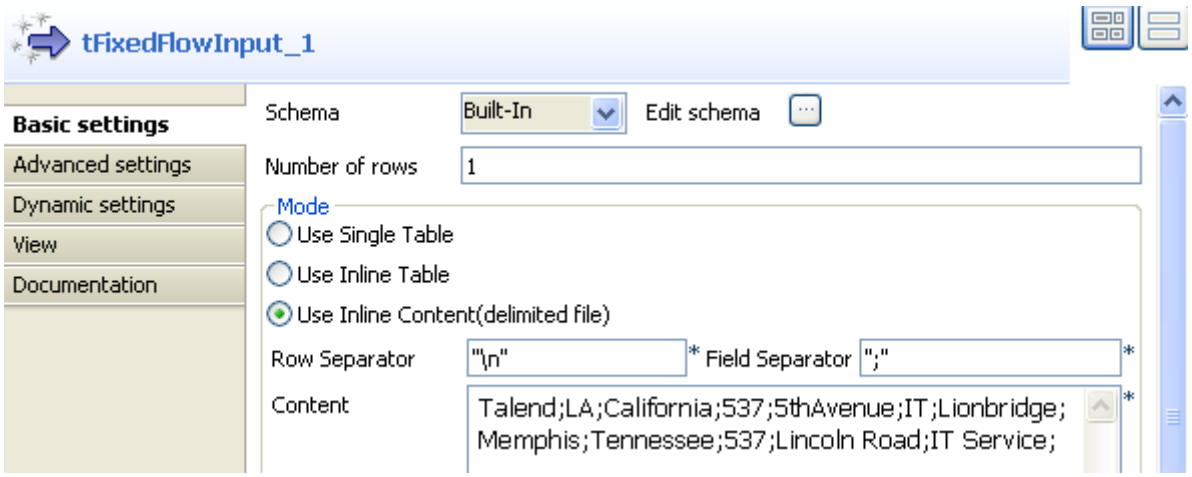
This scenario describes a three-component Job. A row of data containing information of two companies will be split up into two rows.



1. Drop the following components required for this use case: **tFixedFlowInput**, **tSplitRow** and **tLogRow** from the **Palette** to the design workspace.

2. Connect them together using **Row Main** connections.

3. Double-click **tFixedFlowInput** to open its **Basic settings** view.

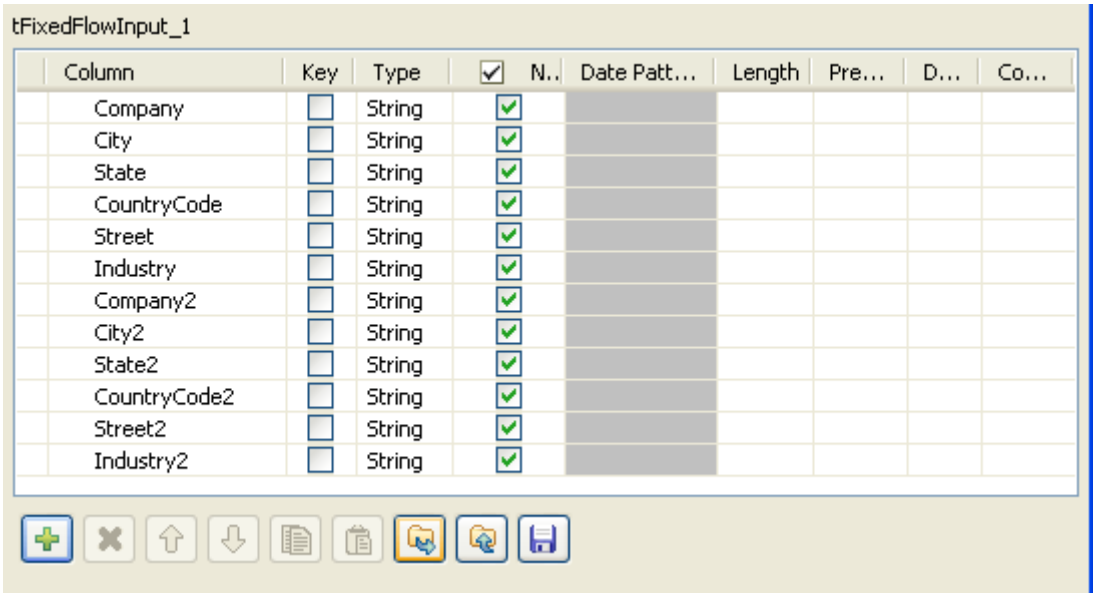


4. Select **Use Inline Content(delimited file)** in the **Mode** area.

5. Fill the **Content** area with the following scripts:

   Talend;LA;California;537;5thAvenue;IT;Lionbridge;Memphis;Tennessee;537;Lincoln Road;IT Service;

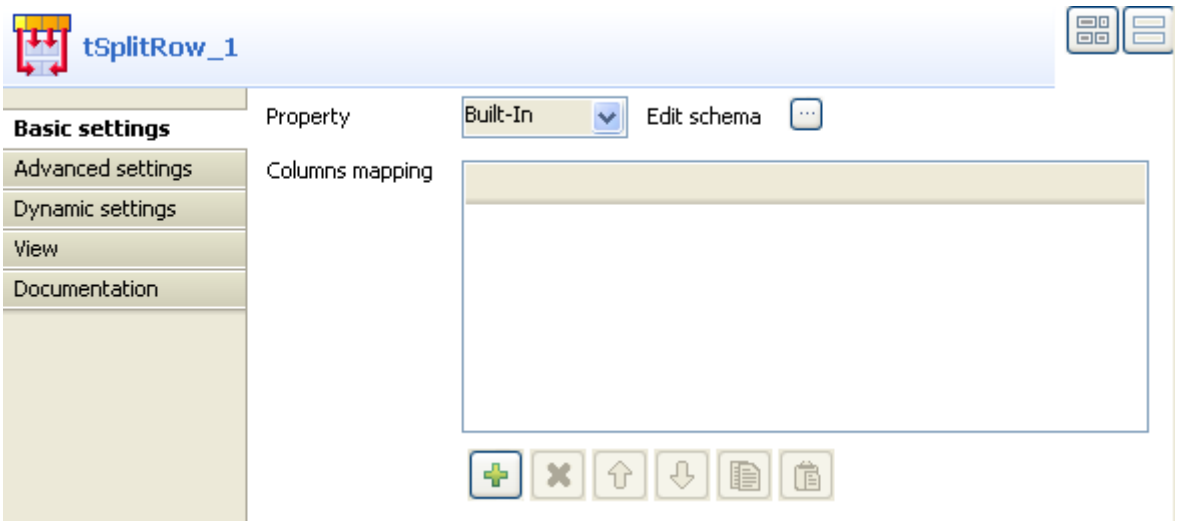
6. Click **Edit schema** to open a dialog box to edit the schema for the input data.



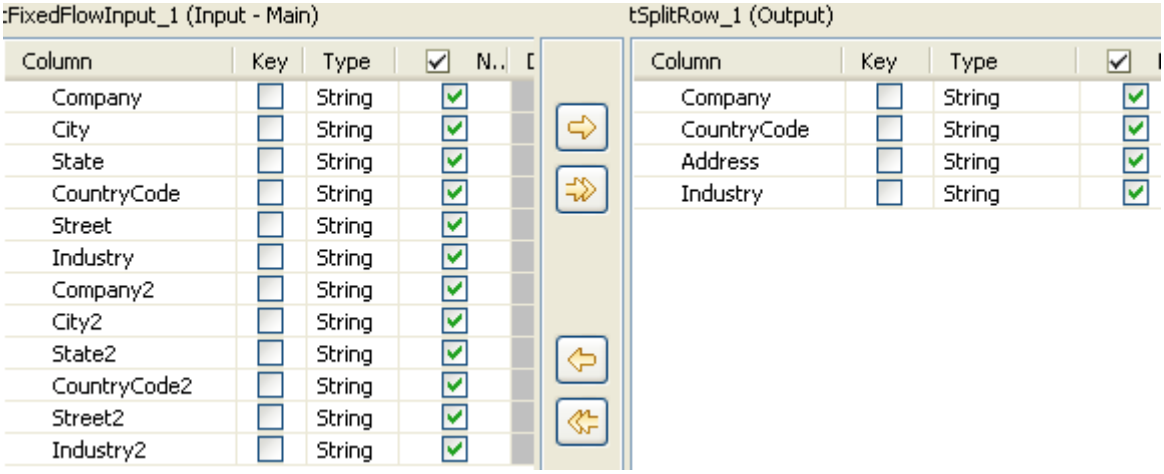
7. Click the plus button to add twelve lines for the input columns: *Company*, *City*, *State*, *CountryCode*, *Street*, *Industry*, *Company2*, *City2*, *State2*, *CountryCode2*, *Street2* and *Industry2*.

8. Click **OK** to close the dialog box.

9. Double-click **tSplitRow** to open its **Basic settings** view.

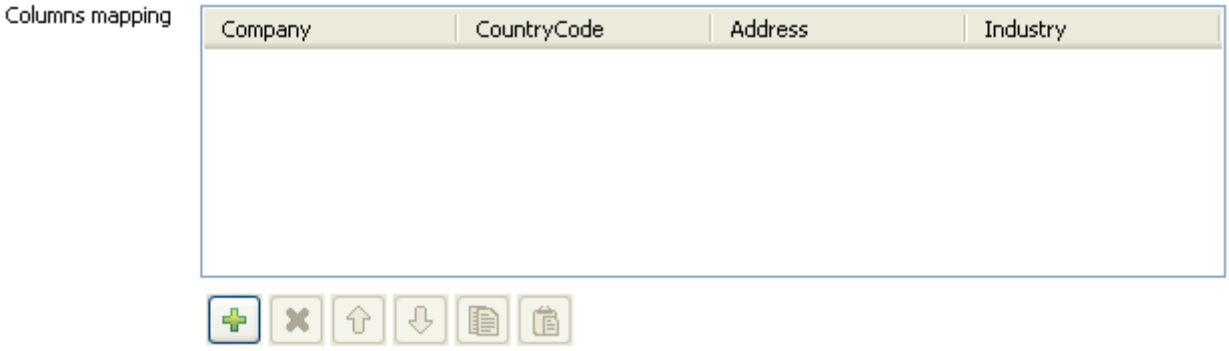


10.Click **Edit schema** to set the schema for the output data.



11.Click the plus button beneath the *tSplitRow_1(Output)* table to add four lines for the output columns: *Company*, *CountryCode*, *Address* and *Industry*.

12.Click **OK** to close the dialog box. Then an empty table with column names defined in the preceding step will appear in the **Columns mapping** area:



13.Click the plus button beneath the empty table in the **Columns mapping** area to add two lines for the output rows.

14.Fill the table in the **Columns mapping** area by columns with the following values:

Company: *row1.Company*, *row1.Company2*;

Country: *row1.CountryCode*, *row1.CountryCode2*;

Address: *row1.Street+","+row1.City+","+row1.State, row1.Street2+","+row1.City2+","+row1.State2*;

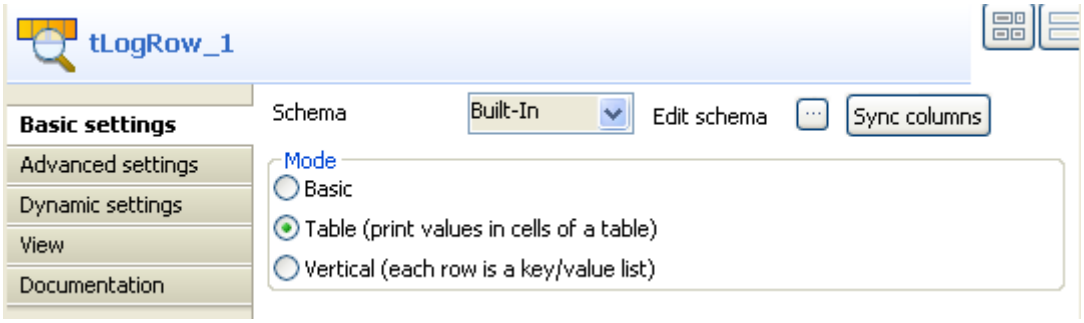
Industry: *row1.Industry, row1.Industry2*;

Columns mapping

Company	CountryCode	Address	Industry
row1.Company	row1.CountryCode	row1.Street+row1.City+row1.State	row1.Industry
row1.Company2	row1.CountryCode2	row1.Street2+row1.City2+row1.State2	row1.Industry2

The value in *Address* column, for example, *row1.Street+","+row1.City+","+row1.State*, will display an absolute address by combining values in *Street* column, *City* column and *State* column together. The "*row1*" used in the values of each column refers to the input row from **tFixedFlowInput**.

15.Double-click **tLogRow** to open its **Basic settings** view.



16.Click **Sync columns** to retrieve the schema defined in the preceding component.

17.Select **Table** in the **Mode** area.

18.Save the Job and press **F6** to run it.

```
[statistics] connecting to socket on port 3706
[statistics] connected
.----------+-----------+-----------------------------+----------.
|                              tLogRow_1                          |
|=---------+-----------+-----------------------------+---------=|
|Company   |CountryCode|Address                      |Industry  |
|=---------+-----------+-----------------------------+---------=|
|Talend    |537        |5thAvenue,LA,California      |IT        |
|Lionbridge|537        |Lincoln Road,Memphis,Tennessee|IT Service|
'----------+-----------+-----------------------------+----------'
[statistics] disconnected
Job Split ended at 16:21 27/10/2011. [exit code=0]
```

The input data in one row is split into two rows of data containing the same company information.

tWriteJSONField

tWriteJSONField properties

Component family	Processing/Fields	
Function	**tWriteJSONField** outputs JSON objects to the defined field of an output file.	
Purpose	**tWriteJSONField** reads data from an input file, assembles it into JSON objects and outputs them to the defined field of an output file.	
Basic settings	*Output Column*	Select the destination field in the output component where you want to write the JSON objects.
	Configure JSON Tree	Opens the interface that supports the creation of the JSON data structure you want to write to a field. For more information about the interface, see the JSON configuration steps in section *Scenario 2: Upserting records in a collection*.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In *Talend Open Studio for Big Data*, the schema is always **Built-in**.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the output component.
	Group by	Define the aggregation set, the columns you want to use to regroup the data.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	n/a	

Related Scenarios

For related scenarios, see:

- section *Scenario 2: Upserting records in a collection*.
- section *Scenario: Extracting the structure of an XML file and inserting it into the fields of a database table*.

tXMLMap

tXMLMap properties

Component family	Processing/XML	
Function	**tXMLMap** is an advanced component fine-tuned for transforming and routing XML data flow (data of the **Document** type), especially when processing numerous XML data sources, with or without joining flat data.	
Purpose	**tXMLMap** transforms and routes data from single or multiple sources to single or multiple destinations.	
Basic settings	*Map editor*	It allows you to define the **tXMLMap** routing and transformation properties.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at the Job level as well as at each component level.
Usage	Possible uses are from a simple reorganization of fields to the most complex jobs of data multiplexing or demultiplexing transformation, concatenation, inversion, filtering and so on. When needs be, you can define sophisticated outputting strategy for the output XML flows using group element, aggregate element, empty element and many other features such as **All in one**. For further information about these features, see *Talend Open Studio for Big Data User Guide*. It is used as an intermediate component and fits perfectly the process requiring many XML data sources, such as, the ESB request-response processes.	
Limitation	The limitations to be kept in mind are: - The use of this component supposes minimum Java and XML knowledge in order to fully exploit its functionalities. - This component is a junction step, and for this reason cannot be a start nor end component in the Job. - At least one loop element is required for each XML data flow involved.	

The following sections present several generic use cases about how to use the **tXMLMap** component, while if you need some specific examples using this component along with the **ESB** components to build data services, see the user guide for the *Talend ESB Studio*.

If you need further information about the principles of mapping multiple input and output flows, see *Talend Open Studio for Big Data User Guide*.

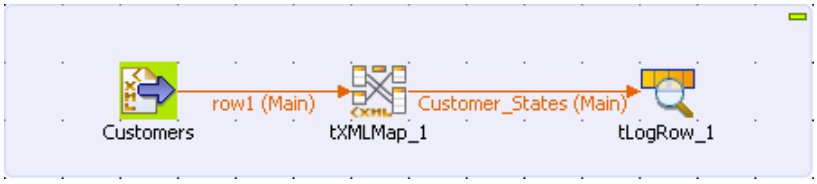
Scenario 1: Mapping and transforming XML data

In this scenario, a three-component Job is run to map and transform data from one XML source, *customer.xml* and generate a XML output flow which could be reused for various purposes in the future, such as, for a ESB request.

These three components are:

- **tFileInputXML**: this component is used to provide input data to **tXMLMap**.
- **tXMLMap**: this component maps and transforms the received XML data flows into one single XML data flow.

- **tLogRow**: this component is used to display the output data.



To replicate this scenario, proceed as the following sections illustrate.

Dropping and linking the components

To do this, proceed as follows:

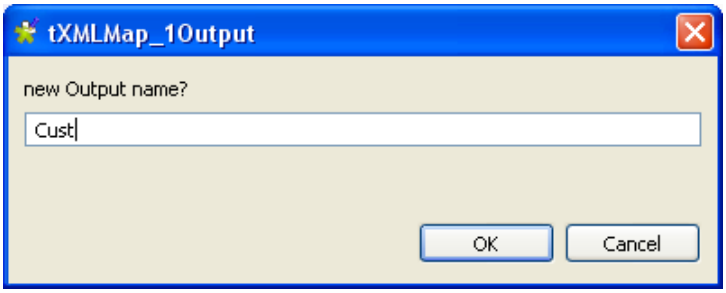
1. From the **Palette**, drop **tFileInputXML**, **tXMLMap** and **tLogRow** into the **Design** workspace.

   A component used in the workspace can be labelled the way you need. In this scenario, this input component is labelled *Customers* for **tFileInputXML**. For further information about how label a component, see *Talend Open Studio for Big Data User Guide*

2. Double click the **tFileInputXML** component labelled *Customers* to open its contextual menu.

3. From this menu, select **Row** > **Main** link to connect this component to **tXMLMap**..

4. Repeat this operation to connect **tXMLMap** to **tLogRow** using **Row** > ***New output* (Main)** link. A dialog box pops up to prompt you to name this output link. In this scenario, name it as *Customer_States*.

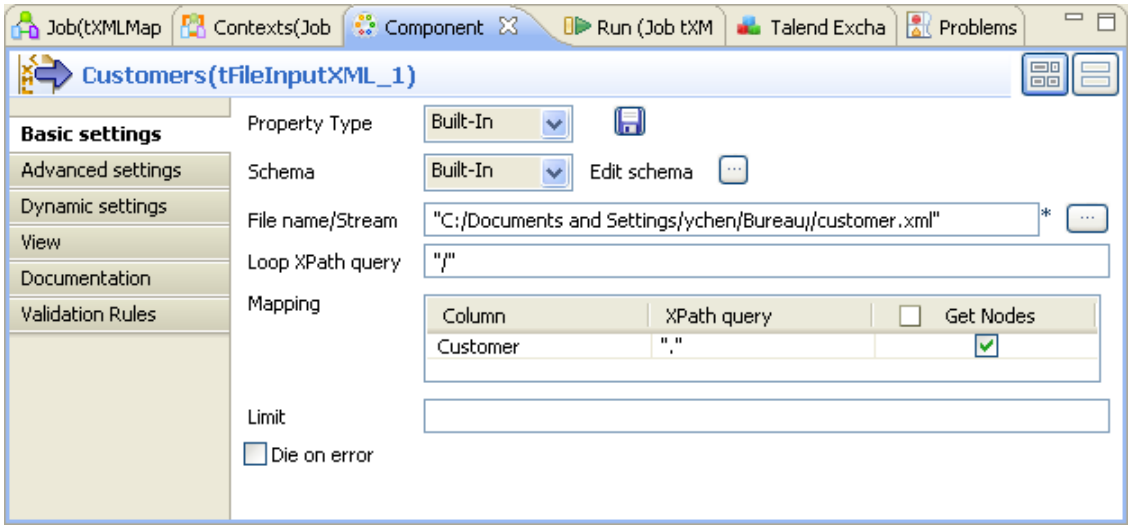


Then you can continue to configure each component.

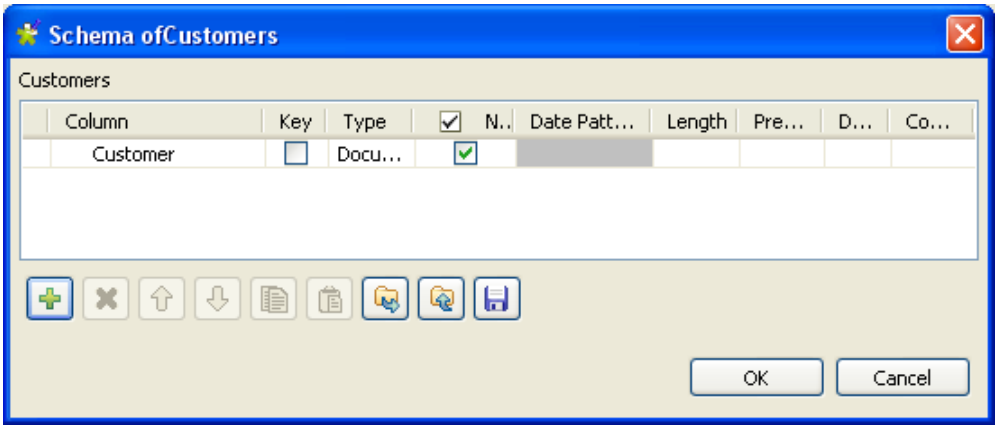
Configuring the input flow

To do this, proceed as follows:

1. Double-click the **tFileInputXML** component labelled *Customers* to display its **Basic settings** view.



2. Next to **Edit schema**, click the three-dot button to open the schema editor.



3. In the schema editor, click the plus button to add one row.

4. In the **Column** column, type in a new name for this row. In this scenario, it is *Customer*.

5. In the **Type** column, select the data type of this row. In this scenario, it is *Document*. The document data type is essential for making full use of **tXMLMap**. For further information about this data type, see *Talend Open Studio for Big Data User Guide*.

6. Click **OK** to validate this editing and accept the propagation prompted by the popup dialog box. One row is added automatically to the **Mapping** table.

7. In the **File name / Stream** field, browse to, or type in the path to the XML source that provides the customer data.

8. In the **Loop XPath query** field, type in "/" to replace the default one. This means the source data is queried from the root.

9. In the **XPath query** column of the **Mapping** table, type in the XPath. In this scenario, type in ".", meaning that all of the data from source are queried.

10. In the **Get Nodes** column of the **Mapping** table, select the check box.

In order to build the **Document** type data flow, it is necessary to get the nodes from this component.

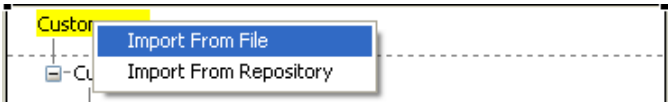
Configuring tXMLMap for transformation

To do this, proceed as follows:

1. Double-click the **tXMLMap** component to open the **Map Editor**.

   Note that the input area is already filled with the defined input tables and that the top table is the main input table.

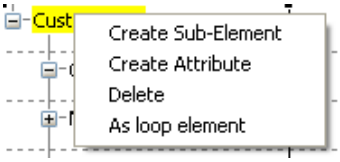
2. In the left table, right-click **Customer** to open the contextual menu.



3. From this contextual menu, select **Import From File** and in the pop-up dialog box, browse to the corresponding source file in order to import therefrom the XML structure used by the data to be received by **tXMLMap**. In this scenario, the source file is *Customer.xml*, which is the data input to **tFileInputXML** (*Customers*).

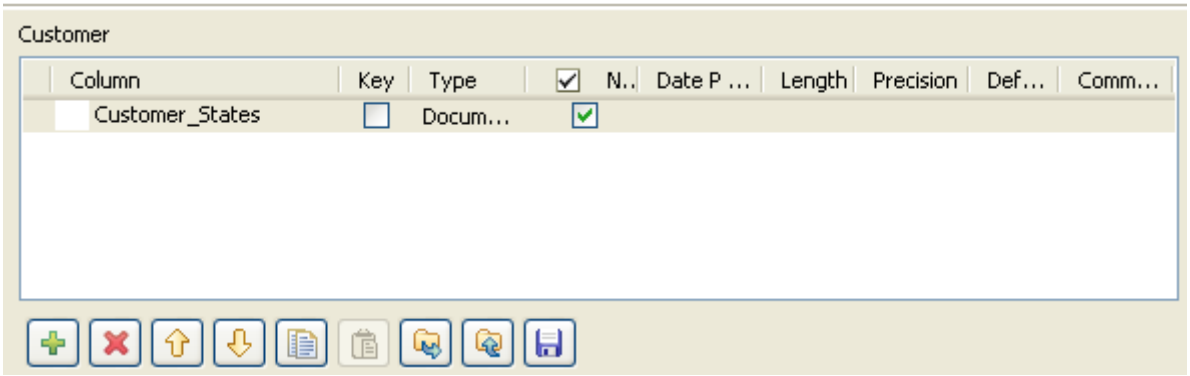
You can also import an XML tree from an XSD file. When importing either an input or an output XML tree structure from an XSD file, you can choose an element as the root of your XML tree. For more information on importing an XML tree from an XSD file, see *Talend Open Studio for Big Data User Guide*.

4. In the imported XML tree, right click the *Customer* node and select **As loop element** to set it as the loop element.

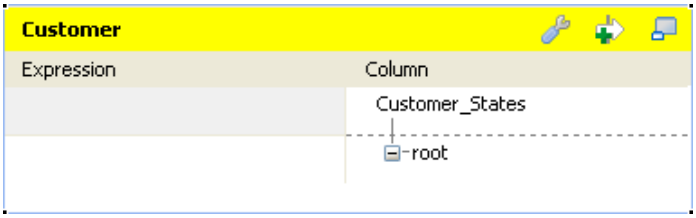


5. On the lower part of this map editor, click the **schema editor** tab to display the corresponding view.

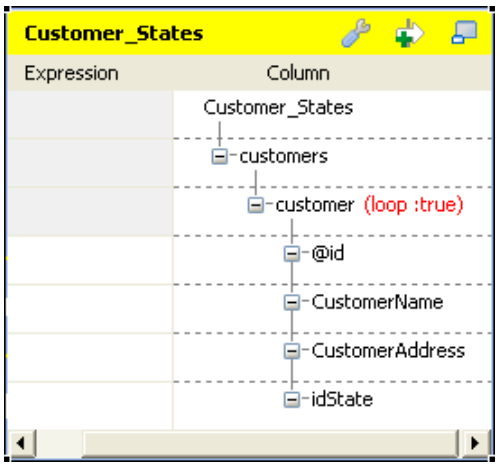
6. On the right side of this view, click the plus button to add one row to the *Customer* table and rename this row as *Customer_States*.



7. In the **Type** column of this *Customer_States* row, select **Document** as the data type. The corresponding XML root is added automatically to the top table on the right side which represents the output flow.



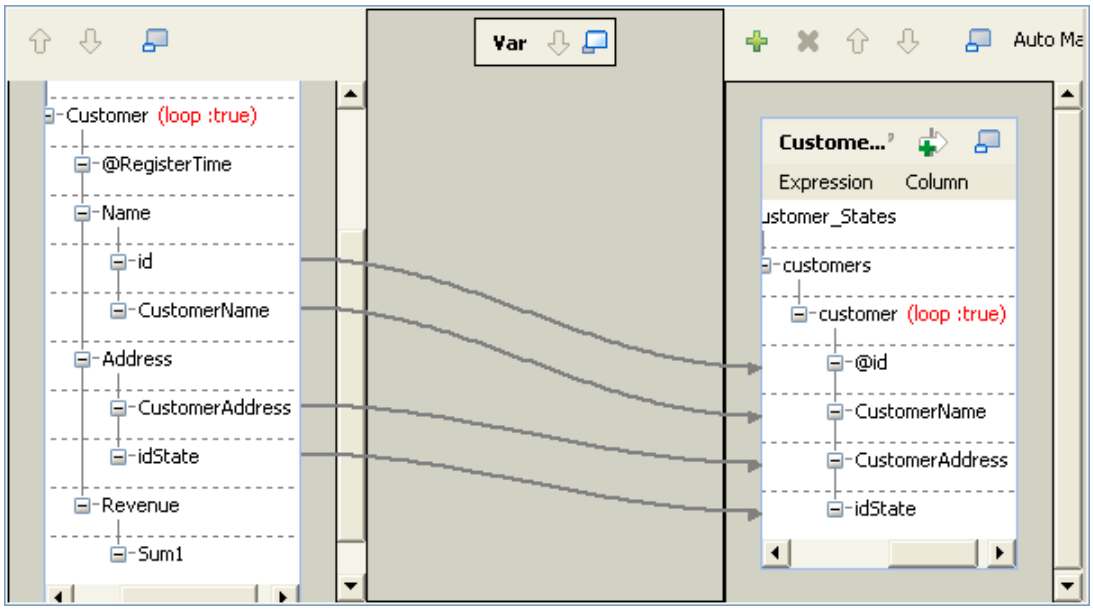
8. On the right side in the top table labelled *Customer_States*, import the XML data structure that you need to use from the corresponding XML source file. In this scenario, it is *Customer_State.xml*.



9. Right click the *customer* node and select **As loop element** from the contextual menu.

   Then you can begin to map the input flow to the output flow.

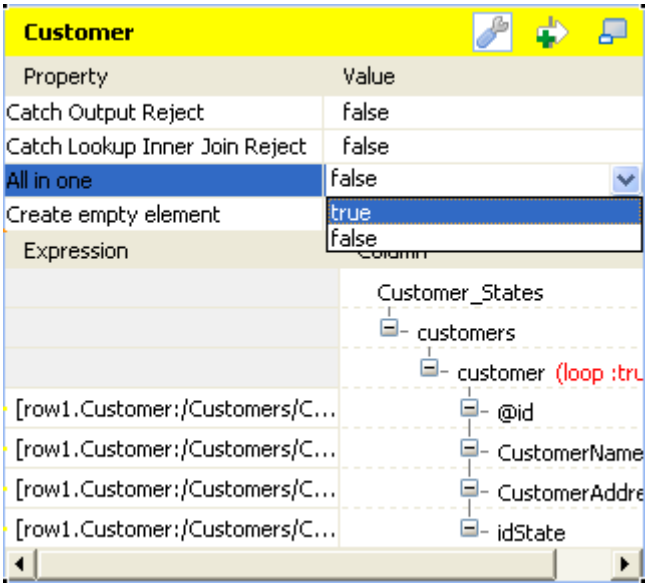
10. In the top table on the input side (left) of the map editor, click the *id* node and drop it to the **Expression** column in the row corresponding to the output row you need map. In this scenario, it is the *@id* node.



11. Do the same to map *CustomerName* to *CustomerName*, *CustomerAddress* to *CustomerAddress* and *idState* to *idState* from the left side to the right side.

In the real project, you may have to keep empty elements in your output XML tree. If so, you can use **tXMLMap** to manage them. For further information about how to manage empty elements using **tXMLMap**, see *Talend Open Studio for Big Data User Guide*.

12. If required to generate single XML flow, click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **true**. In this example, this option is set as **true**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*.



13.Click **OK** to validate the mappings and close the **Map Editor**.

If you close the **Map Editor** without having set the required loop elements as described earlier in this scenario, the root element will be automatically set as loop element.

Then you can run this Job.

Executing the Job

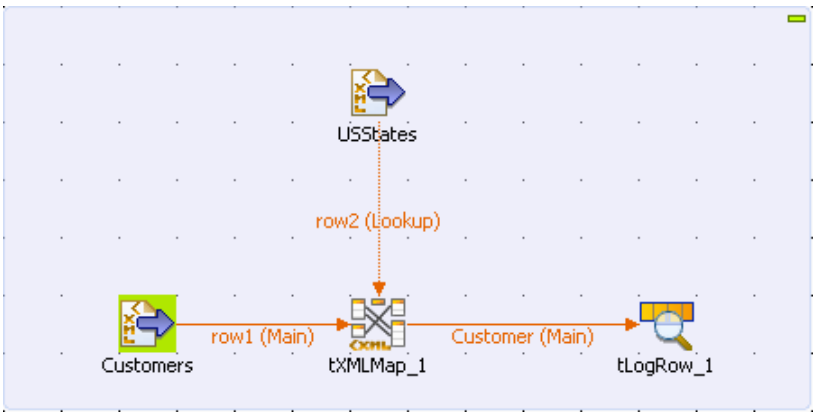
To execute this Job, Press **F6**.

```
Starting job tXMLMap at 11:23 28/04/2011.

[statistics] connecting to socket on port 3382
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</CustomerAddress><idState>7</idState></
customer><customer id="2"><CustomerName>Bill's Dive Shop</CustomerName><CustomerAddress>511
Maple Ave.  Apt. 1B</CustomerAddress><idState>35</idState></customer><customer
id="3"><CustomerName>Childress Child Day Care</CustomerName><CustomerAddress>662 Lyons
Circle</CustomerAddress><idState>1</idState></customer><customer id="4"><CustomerName>Facelift
Kitchen and
Bath</CustomerName><CustomerAddress>unknown</CustomerAddress><idState>0</idState></customer><cus
```

Scenario 2: Launching a lookup in a second XML flow to join complementary data

Based on the previous scenario, this scenario shows how to use lookup in an XML flow to join the data of interest to a given XML flow. The XML data for lookup is held in the *USstates.xml* file.

To do this, a **tFileInputXML** component is added to the previous Job in order to load and send the complementary data to **tXMLMap**. Thus this Job looks like as follows:



To replicate this scenario, proceed as the following sections illustrate.

Configuring the data flow for lookup

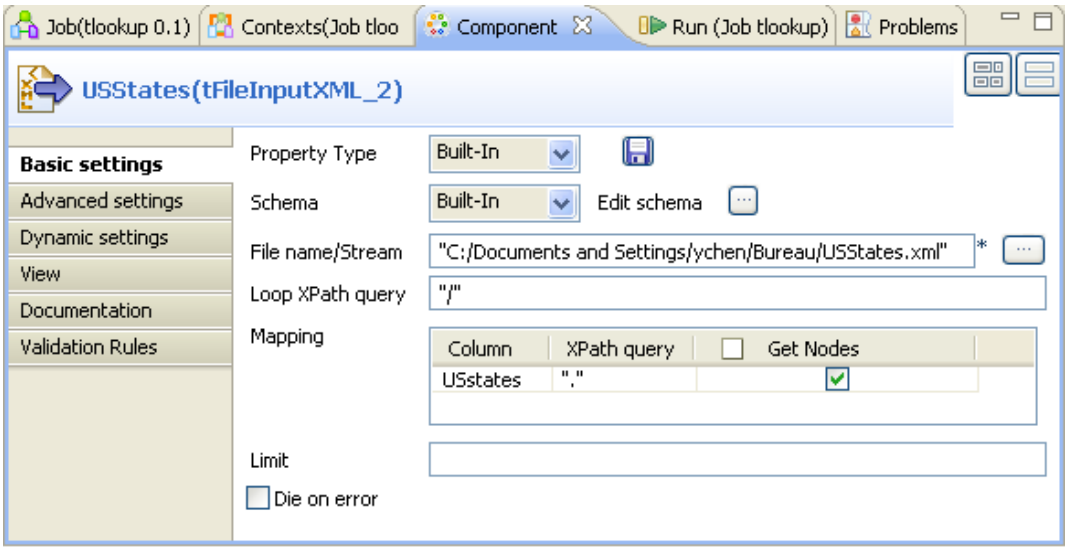
To do this, proceed as follows:

1. From the **Palette**, drop **tFileInputXML** into the Design workspace.

   A component used in the workspace can be labelled the way you need. In this scenario, the newly added **tFileInputXML** is labelled *USstates*. For further information about how to label a component, see *Talend Open Studio for Big Data User Guide*

2. Double click the **tFileInputXML** component labelled *USstates* to open its contextual menu and select **Row** > **Main** connection to connect this component to **tXMLMap**. As you create this connection in the second place, this connection is of type **Lookup**.

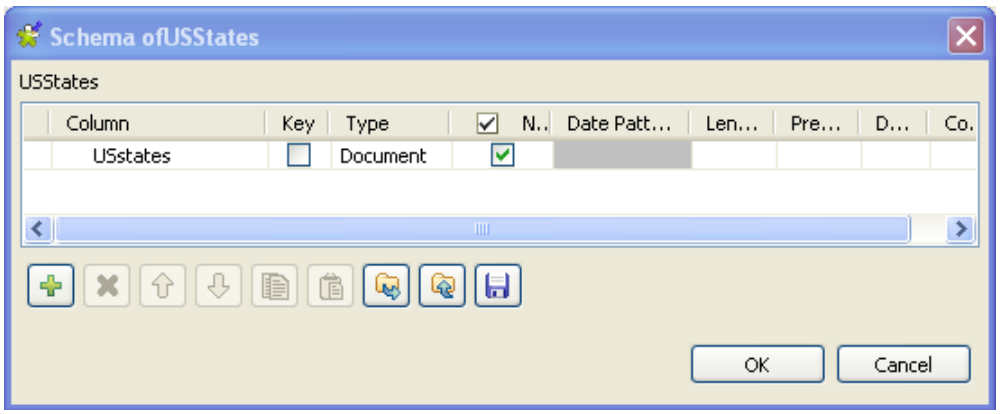
3. Double click the **tFileInputXML** component labelled *USstates* to open its **Component** view.



4. Next to **Edit schema**, click the three-dot buttons to open the schema editor.

5. Click the plus button to add one rows and rename it, for example, as *USState*.

6. In the **Type** column, select the **Document** option from the drop-down list.



7. Click **OK** to validate this editing and accept the propagation prompted by the pop-up dialog box.
8. In the **File name/Stream** field, browse to or type in the path to the *USStates.xml* file.

The input schemas could be stored in the **Metadata** node of the **Repository** tree view for easy retrieval. For further information regarding metadata creation in the Repository, see *Talend Open Studio for Big Data User Guide*.

9. In the **Loop XPath query** field, type in `"/"` to replace the default value. This means the loop is based on the root.
10. In the **Mapping** table, where one row is already added automatically, enter `"."` in the **XPath query** column to retrieve US States data from the source file.
11. In the **Get Nodes** column, select the check box. This retrieves the XML structure for the **Document** type data.

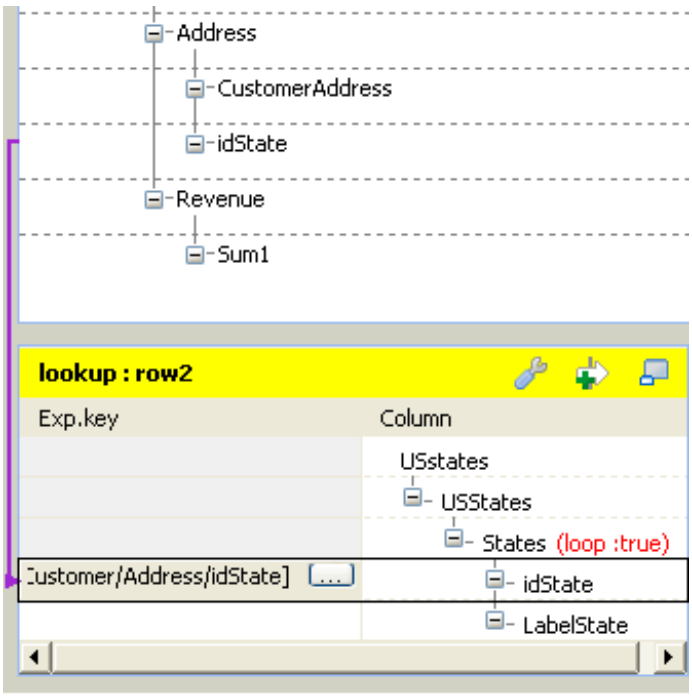
Configuring the transformation

To do this, proceed as follows

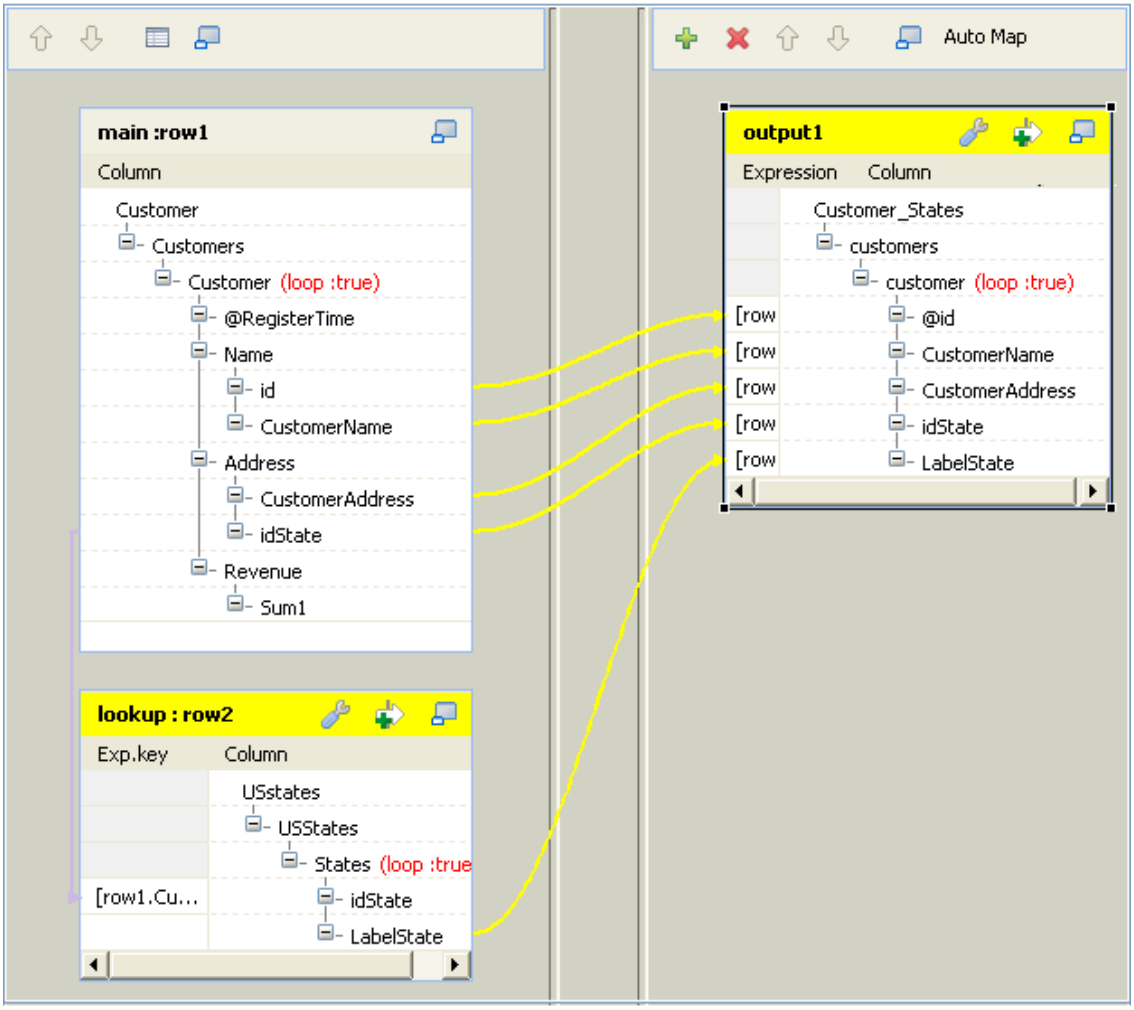
1. Double-click the **tXMLMap** component to open the **Map Editor**.

   Note that the input area is already filled with the defined input tables and that the top table is the main input table.

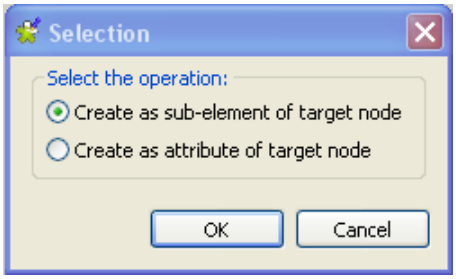
2. In the top table, click the *idState* node and drop it, in the lower table, to the **Exp.key** column in the row corresponding to the *idState* row. This creates a join between the two tables on the *idState* data, among which the *idState* node from the main flow provides the lookup key.



Then you can begin to modify the mapping you have done in the previous scenario to join the complementary data into the input flow. This mapping then should look like as follows:

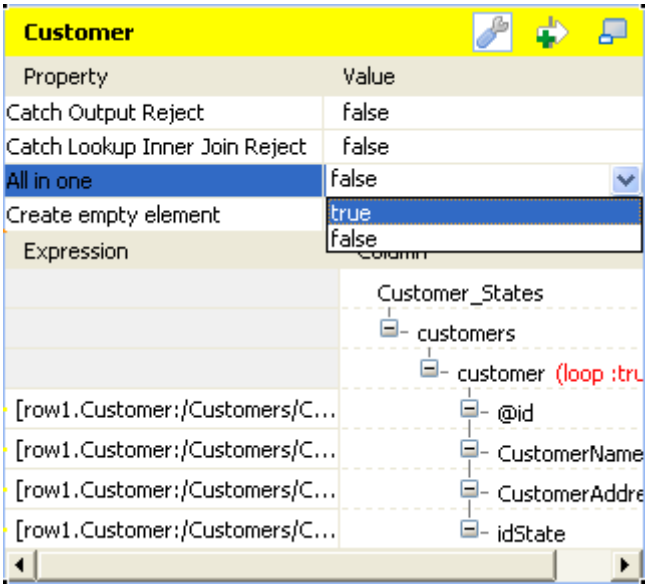


3. In the **lookup** table on the input side (left) of the map editor, click the *LabelState* row and drop it on the *customer* node on the output side. A dialog box pops up.



4. In this dialog box, select **Create as sub-element of target node** and click **OK**. This operation adds a new sub-element to the output XML tree and maps it with *LabelState* on the input inside at the same time.

5. If required to generate single XML flow, click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **true**. In this example, this option is set as **true**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*.



6. Click **OK** to validate the mappings and close the **Map Editor**.

7. Press **F6** to run this Job.

The **Run** view presents the execution result which may read as follows:

```
[statistics] connecting to socket on port 3683
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</CustomerAddress>
<idState>7</idState><LabelState>Connecticut</LabelState></customer><custom
er id="56"><CustomerName>Glenn Oaks Office
Supplies</CustomerName><CustomerAddress>1859 Green Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Connecticut</LabelSta
te></customer><customer id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave.  Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</LabelState></cu
stomer><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio</LabelState></c
ustomer><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Florida</LabelState></
customer></customers>
[statistics] disconnected
```

The US state labels that correspond to the state IDs provided as the lookup key by the main data flow are selected and outputted.

A step-by-step tutorial related to this Join topic is available on the Talend Technical Community Site. For further information, see http://talendforge.org/tutorials/tutorial.php?language=english&idTuto=101.

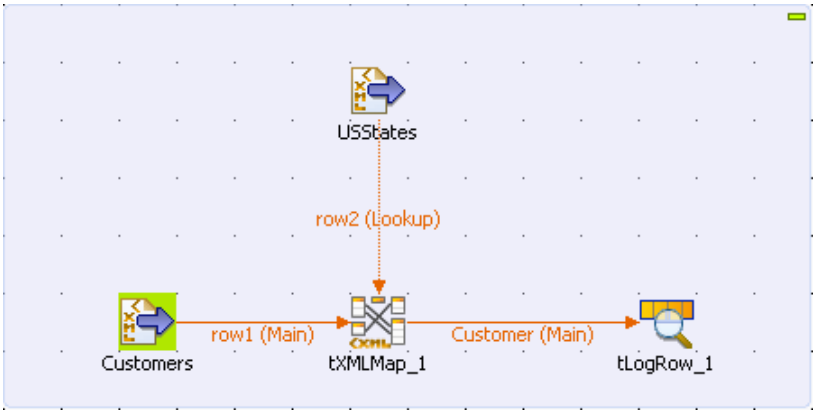
Scenario 3: Mapping data using a filter

Based on section *Scenario 2: Launching a lookup in a second XML flow to join complementary data*, this scenario presents how to apply filter condition(s) to select the data of interest using **tXMLMap**.

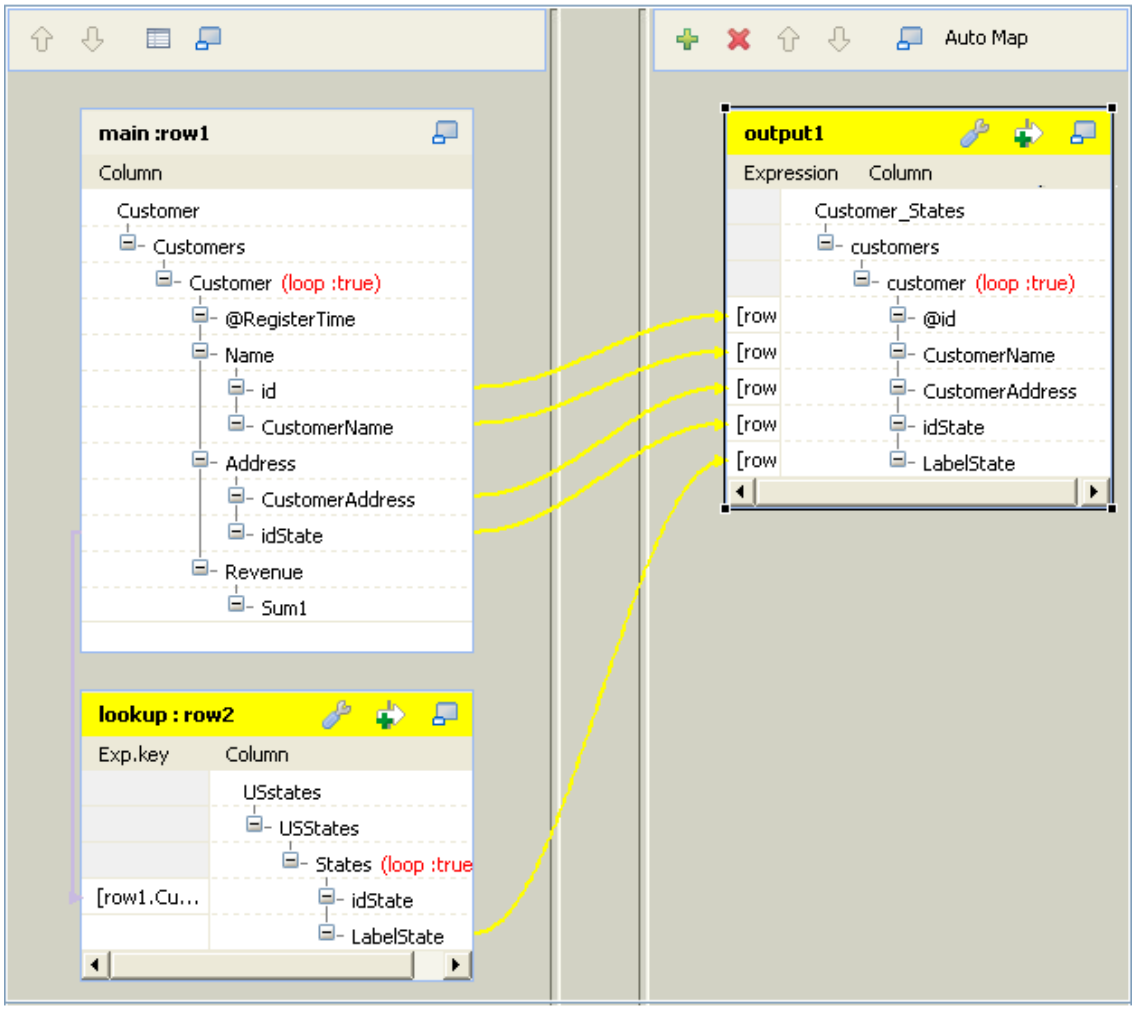
For example, you need to select the customer data where the state id is *9*.

To replicate this scenario, proceed as follows:

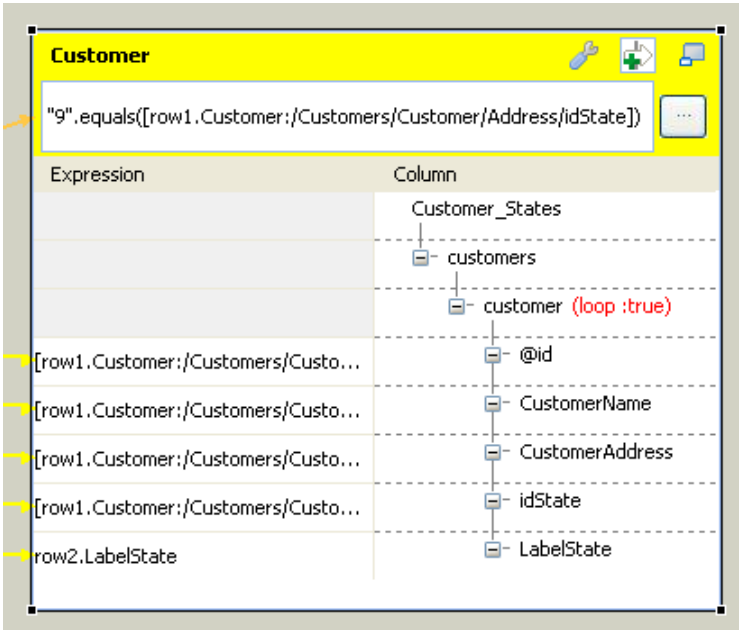
1. In your Studio, open the Job used in the previous scenario to display it in the Design workspace.



2. Double click **tXMLMap** to open its editor. In this editor, the input and output data flows have been mapped since the replication of the previous scenario.



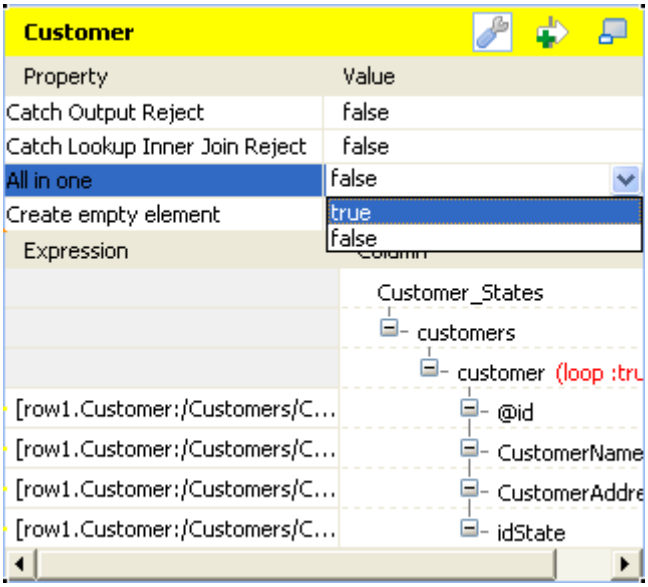
3. On the output side (right), click the button to open the filter area.



4. In this filter area, drop the *idState* node from the tree view of the input data flow. The Xpath of *idState* is added automatically to this filter area.

5. Still in this area, write down the filter condition of interest in Java. In this scenario, this condition reads: `"9".equals([row1.Customer:/Customers/Customer/Address/idState])`

6. If required to generate single XML flow, click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **true**. In this example, this option is set as **true**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*.



7. Click **OK** to validate this editing and close this editor.

8. Press **F6** to run this Job.

The execution result is displayed in the **Run** view as follows:

```
[statistics] connecting to socket on port 3385
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Florida</LabelState>
</customer></customers>
[statistics] disconnected
```

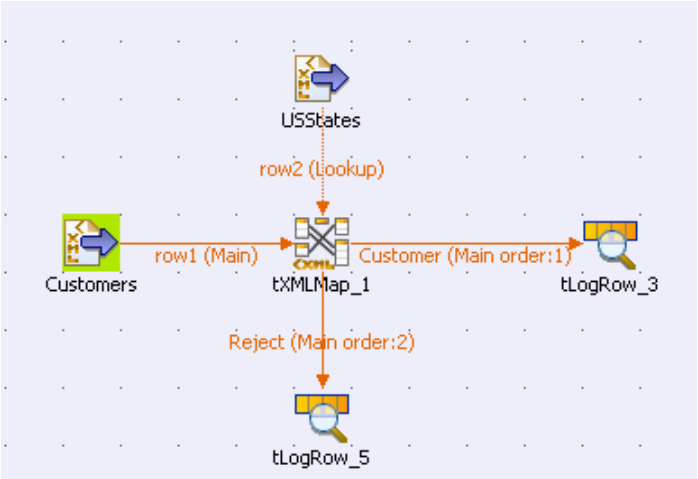
The result says that the customer *Pivot Point College* is selected as its state ID is *9*, representing the *Florida* state in this scenario.

Scenario 4: Catching the data rejected by lookup and filter

The data rejected by the lookup and filter conditions you set in **tXMLMap** can be caught and outputted by this component itself.

Based on section *Scenario 3: Mapping data using a filter*, this scenario presents how to catch the data rejected by the lookup and the filter set up in the previous sections.

In this scenario, another **tLogRow** component is added to the Job used in the previous scenario and thus the Job displays as follows:

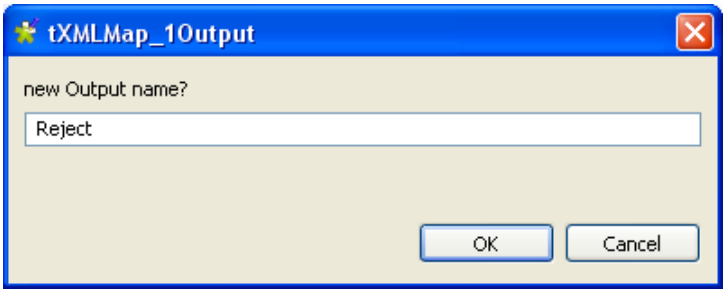


To replicate this scenario, proceed as follows:

1. In your Studio, open the Job used in the previous scenario to display it in the Design workspace.

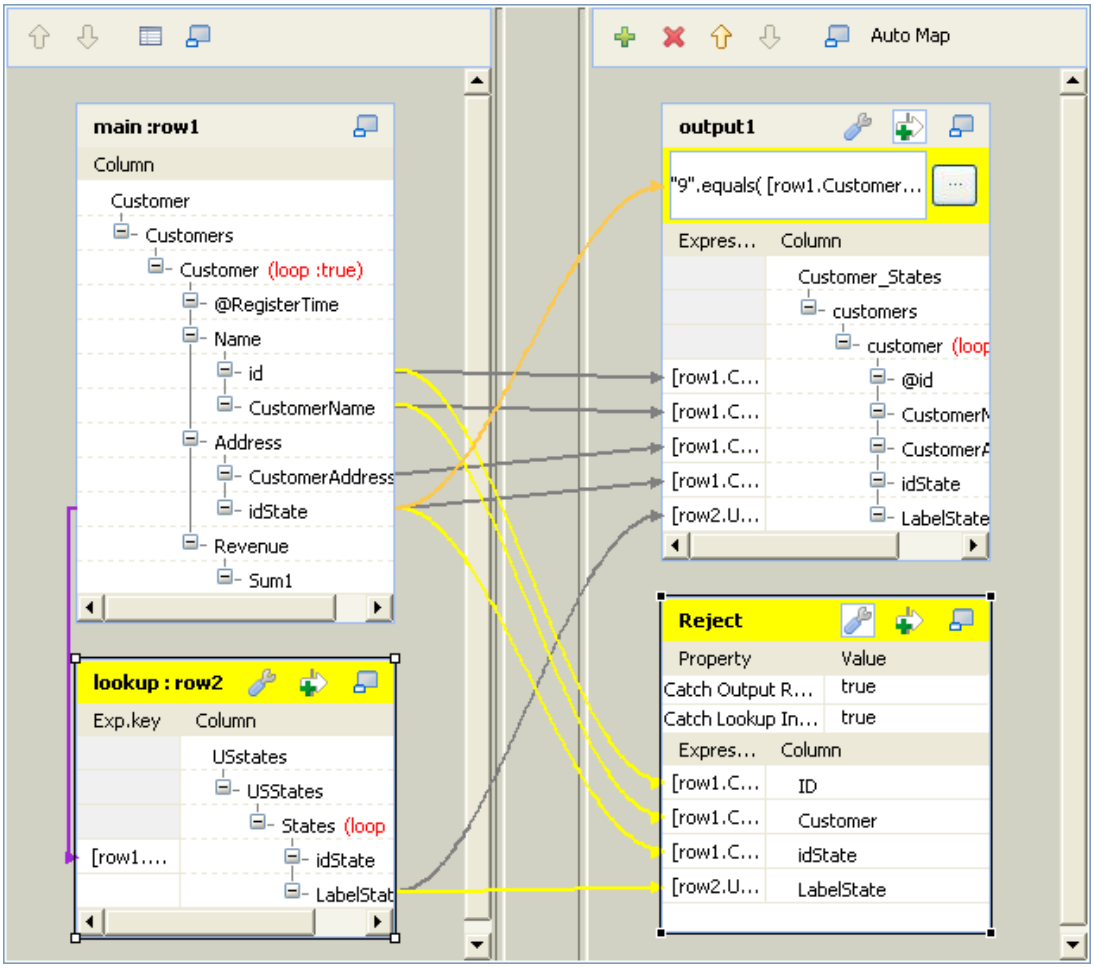
2. From the **Palette**, drop the **tLogRow** component on the workspace.

3. Right-click **tXMLMap** to open its contextual menu and select **Row > *New Output* (Main)** to connect this component to the newly added **tLogRow** component. A dialog box pops up to prompt you to name this output link. In this scenario, name it as *Reject*.

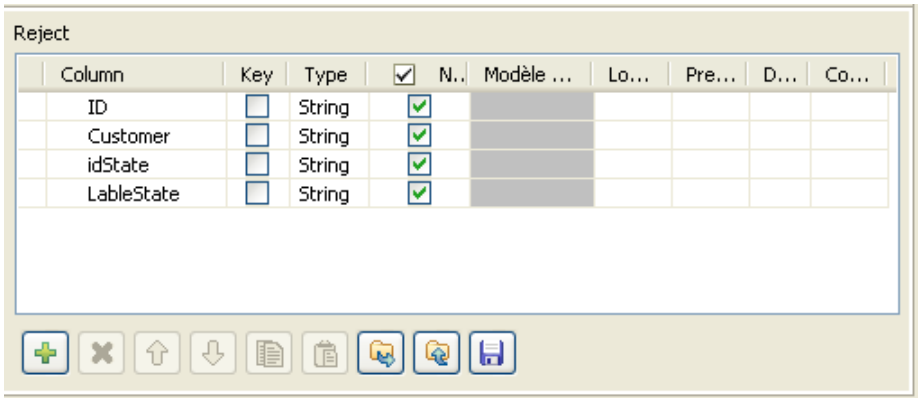


4. Click **OK** to validate this creation.

5. Double click the **tXMLMap** component to open its editor. An empty *Reject* table has been added to the output side to represent the output data flow carrying the rejected data. You need to complete this table to make this editor look like as follows:



6. Select this empty *Reject* table.
7. In the lower part of this editor, click the **Schema editor** tab to open the corresponding view.
8. On the right part of this **Schema editor** view, click the plus button to add the rows you need to use. In this scenario, click four times to add four rows to the *Reject* table.



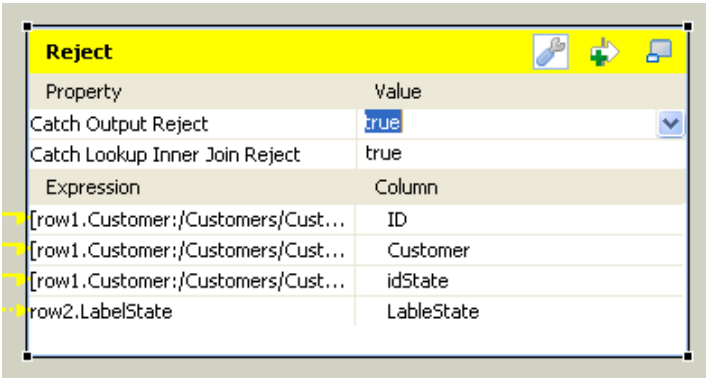
9. In the *Reject* table presented on the right part of this **Schema editor** view, rename each of the four newly added rows. They are: *ID*, *Customer*, *idState*, *LabelState*.

In this scenario, the *Reject* output flow uses flat data type. However, you can create an XML tree view for this flow using the **Document** data type. For further information about how to use this **Document** type, see section *Scenario 1: Mapping and transforming XML data*.

The *Reject* table is completed and thus you have defined the schema of the output flow used to carry the captured rejected data. Then you need to set up the condition(s) to catch the rejected data of interest.

10.On the upper part of the output side in this **Map editor**, select the *Reject* table.

11.At the top of this table, click the button to open the setting area.



12.In the **Catch Output Reject** row of the setting area, select **true** from the drop-down list. Thus **tXMLMap** outputs the data rejected by the filter set up in the previous scenario for the *Customer* output flow.

13.Do the same thing to switch the **Catch Lookup Inner Join Reject** row to the **true** option.

14.Click **OK** to validate this editing and close this editor.

15.Press **F6** to run this Job.

The captured data rejected by the filter and the lookup reads as follows in the **Run** view:

```
1|Griffith Paving and Sealcoatin|7|Connecticut
56|Glenn Oaks Office Supplies|7|Connecticut
2|Bill's Dive Shop|35|Ohio
61|DBN Bank|35|Ohio
62|BBQ Smith's Tex Mex|60|Ohio
```

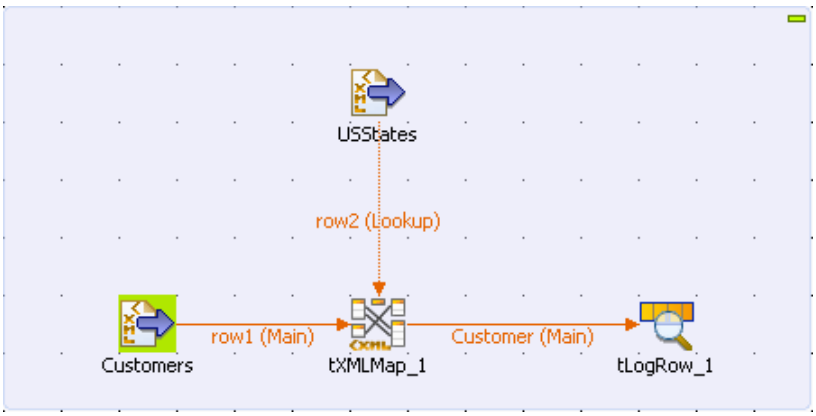
None of the State IDs of these customers is *9*. The customer *BBQ Smith's Tex Mex* is marked with the state ID *60*. This number does not exist in the *idState* column of *USState.txt* where the defined lookup was done, so the data of this customer is rejected by the lookup and the other data rejected by the filter.

The data selected by the filter you set up in the previous scenario reads as follows in XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Florida</LabelState>
</customer></customers>
```

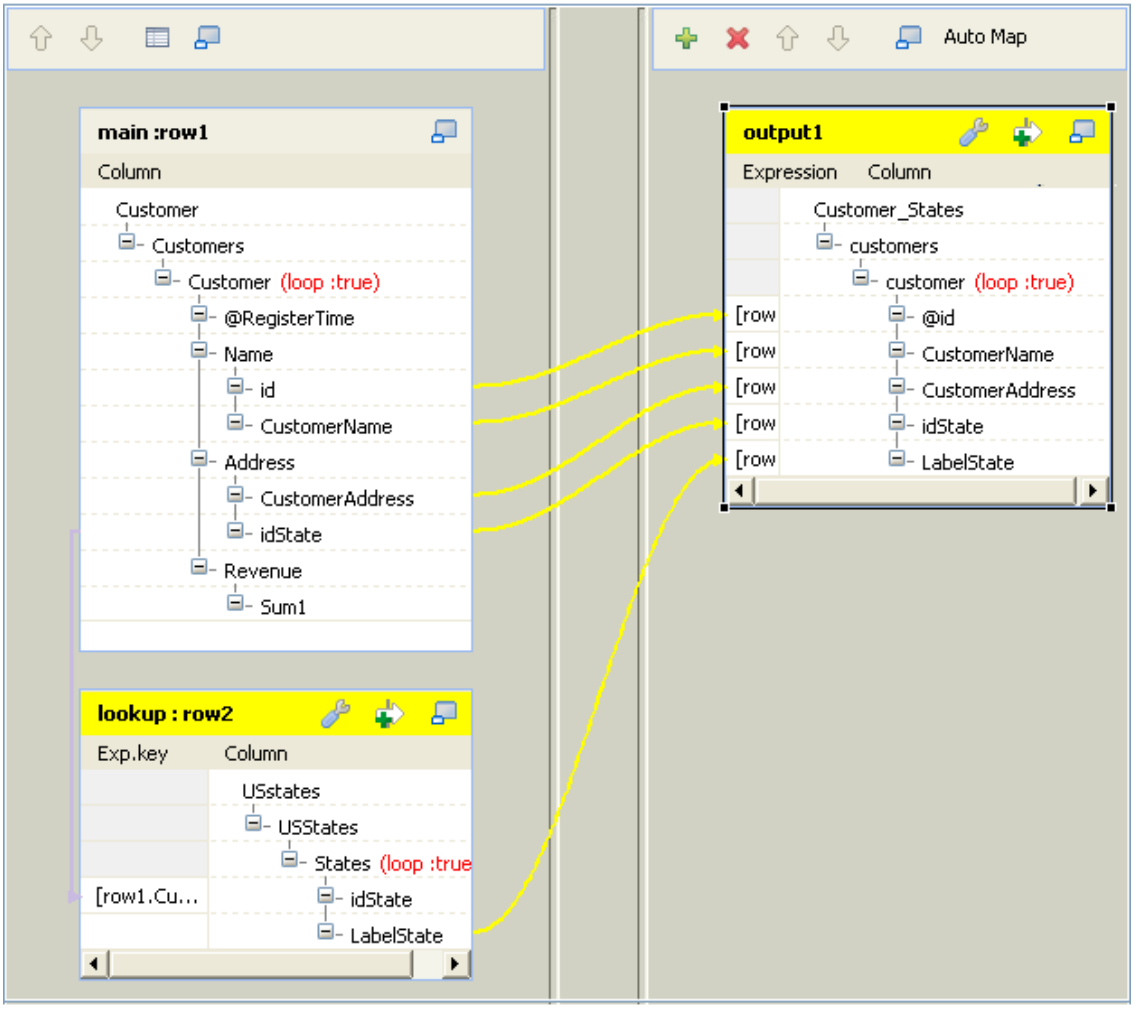
Scenario 5: Mapping data using a group element

Based on the Job used in section *Scenario 2: Launching a lookup in a second XML flow to join complementary data*, this scenario presents how to set up an element as **group element** in the **Map editor** of **tXMLMap** to group the output data.



To replicate this scenario, you can reuse the Job in section *Scenario 2: Launching a lookup in a second XML flow to join complementary data*.

In this Job, double click **tXMLMap** to open the **Map editor**.



The objective of this scenario is to group the customer id and the customer name information according to the States the customers come from. To do this, you need to adjust the XML structure with considering the following factors:

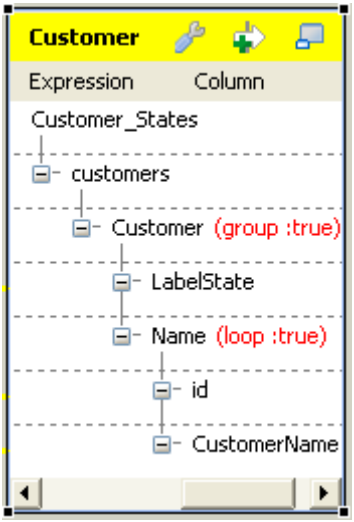
- The elements tagging the customer id and the customer name information should be located under the loop element. Thus they are the sub-elements of the loop element.
- The loop element and its sub-elements should be dependent directly on the group element.
- The element tagging the States used as grouping condition should be dependent directly on the group element.

- The group element cannot be the root element.

To put a group element into effect, the XML data to be processed should have been sorted, for example via your XML tools, around the element you need to use as the grouping condition. The figure below presents part of the sorted source data used in this scenario. The customers possessing the same State id is already put together.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<Customers>
    <Customer RegisterTime="2001-01-17 06:26:40.000">
        <Name>
            <id>1</id>
            <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
        </Name>
        <Address>
            <CustomerAddress>talend@apres91</CustomerAddress>
            <idState>7</idState>
        </Address>
        <Revenue>
            <Suml>67852</Suml>
        </Revenue>
    </Customer>
    <Customer RegisterTime="1987-02-23 17:33:20.000">
        <Name>
            <id>56</id>
            <CustomerName>Glenn Oaks Office Supplies</CustomerName>
        </Name>
        <Address>
            <CustomerAddress>1859 Green Bay Rd.</CustomerAddress>
            <idState>7</idState>
        </Address>
        <Revenue>
            <Suml>1225.</Suml>
        </Revenue>
    </Customer>
    <Customer RegisterTime="2002-06-07 09:40:00.000">
        <Name>
            <id>2</id>
            <CustomerName>Bill's Dive Shop</CustomerName>
```

Based on this analysis, the structure of the output data should read as follows:



In this figure, the *customers* node is the root, the *Customer* element is set as **group element** and the output data is grouped according to the *LabelState* element.

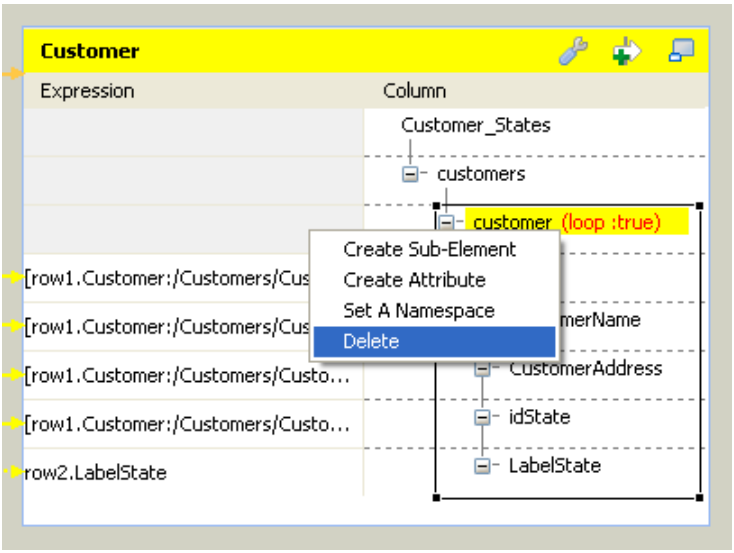
To set a group element, two restrictions must be respected:

- the root node cannot be set as **group element**;
- the **group element** must be the parent of the loop element.

Once the group element is set, the first element except the loop one is used as condition to group the output data.

To perform the adjustment according to this analysis, proceed as follows:

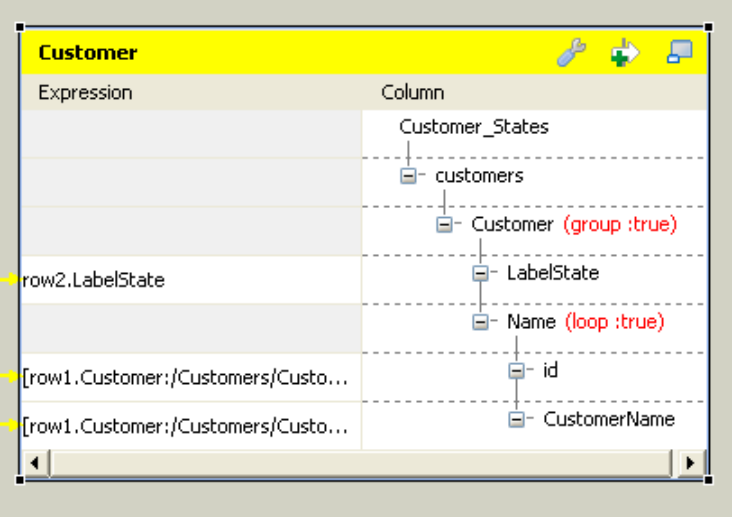
1. In the XML tree view of the output side, right-click the *customer (loop:true)* node to open the contextual menu and select **Delete**. Thus all of the elements under the root *customers* are removed. Then you can reconstruct the XML tree view to have the best structure used to group the output data of interest.



2. Again in the XML tree view of the output side, right-click the root node *customers* to open the contextual menu and select **Create sub-element**. Then a dialog box pops up.

3. Type in the name of the new sub-element. In this scenario, it is *Customer*.
4. Repeat the previous operations to create two more sub-elements under this *Customer* node. They are: *LabelState* and *Name*.
5. Do these operations again to create two more sub-elements under this newly created *Name* node. They are: *id* and *CustomerName*.
6. Right-click the *Name* node to open the contextual menu and select **As loop element** to set this element as loop.
7. Right-click the *Customer* node to open its contextual menu and select **As group element**. This means that the output data is grouped according to the *LabelState* element.
8. From the lookup data flow on the input side, click and drop the *LabelState* row to the row of the *LabelState* node in the **Expression** column on the output side. Thus the corresponding data is mapped.

9. Do the same to map the *id* element and the *CustomerName* elements between both sides. Then this modification is done.



10. If required to generate single XML flow, click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **true**. In this example, this option is set as **true**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*.

11. Click **OK** to validate this modification and close this editor.

If you close the **Map Editor** without having set the required loop elements as described earlier in this scenario, the root element will be automatically set as loop element.

12. Press **F6** to run this Job.

The execution result reads as follows in the **Run** view.

```
<?xml version="1.0" encoding="UTF-8"?>
<customers><Customer><LabelState>Connecticut</LabelState><Name><id>1
</id><CustomerName>Griffith Paving and
Sealcoatin</CustomerName></Name><Name><id>56</id><CustomerName>Glen
n Oaks Office
Supplies</CustomerName></Name></Customer><Customer><LabelState>Ohio<
/LabelState><Name><id>2</id><CustomerName>Bill's Dive
Shop</CustomerName></Name><Name><id>61</id><CustomerName>DBN
Bank</CustomerName></Name></Customer><Customer><LabelState>Florida</
LabelState><Name><id>63</id><CustomerName>Pivot Point
College</CustomerName></Name></Customer></customers>
```

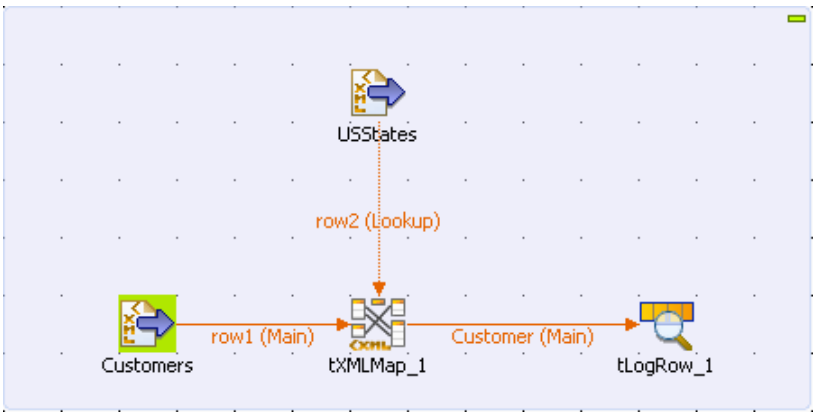
The *id* element and the *CustomerName* element contained in the loop are grouped according to the *LabelState* element. The group element *Customer* tags the start and the end of each group.

tXMLMap provides **group element** and **aggregate element** to classify data in the XML tree structure. When handling one row of data (one complete XML flow), the behavioral difference between them is:

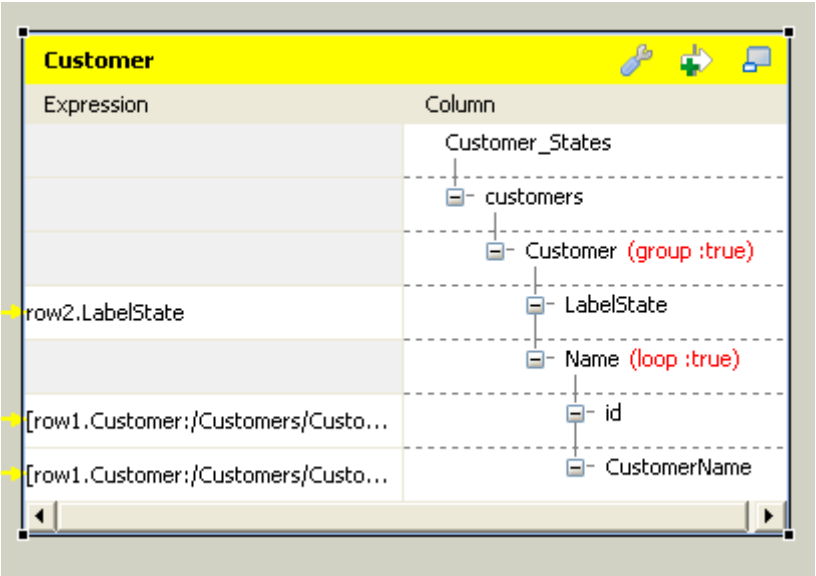
- The **group element** processes the data always within one single flow.
- The **aggregate element** splits this flow into separate and complete XML flows.

Scenario 6: classing the output data with aggregate element

Based on the Job used in section *Scenario 5: Mapping data using a group element*, this scenario presents how to set up an element as **aggregate element** in the **Map editor** of **tXMLMap** in order to class the output data into separate XML flows.



On the Design workspace, double-click the **tXMLMap** component to open its **Map editor**. There the output side reads as follows:



The objective of this scenario is to class the customer information using aggregate element in accordance with the States they come from and then to send these classes separately in different XML flows to the component that follows.

To put an aggregate element into effect, the XML data to be processed should have been sorted, for example via your XML tools, around the element you need to use as the aggregating condition. The figure below presents part of the sorted source data used in this scenario. The customers possessing the same State id is already put together.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<Customers>
    <Customer RegisterTime="2001-01-17 06:26:40.000">
        <Name>
            <id>1</id>
            <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
        </Name>
        <Address>
            <CustomerAddress>talend@apres91</CustomerAddress>
            <idState>7</idState>
        </Address>
        <Revenue>
            <Suml>67852</Suml>
        </Revenue>
    </Customer>
    <Customer RegisterTime="1987-02-23 17:33:20.000">
        <Name>
            <id>56</id>
            <CustomerName>Glenn Oaks Office Supplies</CustomerName>
        </Name>
        <Address>
            <CustomerAddress>1859 Green Bay Rd.</CustomerAddress>
            <idState>7</idState>
        </Address>
        <Revenue>
            <Suml>1225.</Suml>
        </Revenue>
    </Customer>
    <Customer RegisterTime="2002-06-07 09:40:00.000">
        <Name>
            <id>2</id>
            <CustomerName>Bill's Dive Shop</CustomerName>
```

To do this, adjust the output XML tree as follows:

1. Right-click the **Customer** element to open its contextual menu and from this menu, select **Remove group element**.

2. Click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **false**.

3. Right-click the **LabelState** element to open its context menu and from this menu, select **As aggregate element**. This element tags the State information of each customer and the customer information will be classed under the State information.

   To make the aggregate element available, ensure that the **All in one** feature is set as **false**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*

4. Click **OK** to validate these changes and close the **Map editor**.

5. Press **F6** to run this Job.

Once done, the **Run** view is opened automatically, where you can check the execution result.

```
Démarrage du job tXMLMap a 15:51 04/04/2012.

[statistics] connecting to socket on port 3364
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<root><Customers><LabelState>Connecticut</LabelState><Name><id>1</id><CustomerNa
me>Griffith Paving and
Sealcoatin</CustomerName></Name><Name><id>56</id><CustomerName>Glenn Oaks Office
Supplies</CustomerName></Name></Customers></root>
<?xml version="1.0" encoding="UTF-8"?>
<root><Customers><LabelState>Ohio</LabelState><Name><id>2</id><CustomerName>Bill
s Dive Shop</CustomerName></Name><Name><id>61</id><CustomerName>DBN
Bank</CustomerName></Name></Customers></root>
<?xml version="1.0" encoding="UTF-8"?>
<root><Customers><LabelState>Florida</LabelState><Name><id>63</id><CustomerName>P
ivot Point College</CustomerName></Name></Customers></root>
[statistics] disconnected
Job tXMLMap terminé à 15:51 04/04/2012. [Code sortie=0]
```

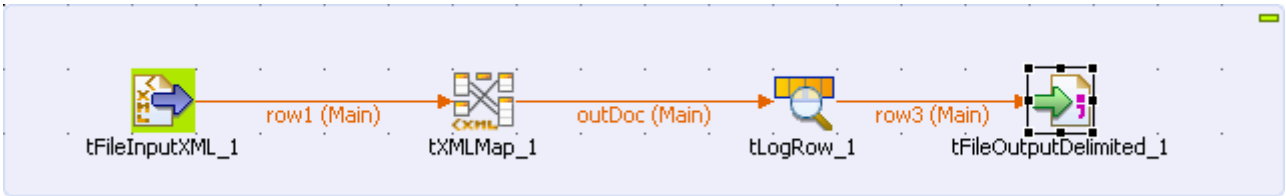
tXMLMap outputs three separate XML flows, each of which carries the information of one State and the customers from that State.

tXMLMap provides **group element** and **aggregate element** to classify data in the XML tree structure. When handling one row of data (one complete XML flow), the behavioral difference between them is:

- The **group element** processes the data always within one single flow.
- The **aggregate element** splits this flow into separate and complete XML flows.

Scenario 7: Restructuring products data using multiple loop elements

This scenario uses a four-component Job to restructure the products data given by a document flow using multiple loop elements.



The components used are:

- **tFileInputXML**: it reads the source product data and pass them to the tXMLMap component.
- **tXMLMap**: it transforms the input flows with the expected structure streamlined.
- **tLogRow**: it presents the execution result in the console.
- **tFileOutputDelimited**: it generates the output flow into an XML file.

The input flow reads as follows:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<products category="1" name="laptop">

    <!-- Summary -->
    <summary>
        <company>DELL, HP</company>
        <sales unit="Dollars">12345678910.12345</sales>
        <model>business</model>
    </summary>

    <!-- Loop1 manufacture -->
    <manufacture id="manu_1" date="2012-10-30">
        <name>DELL</name>
    </manufacture>
    <manufacture id="manu_2" date="2012-10-28">
        <name>HP</name>
    </manufacture>

    <!-- Loop2 types -->
    <types model="business1">
        <type>DELL123</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>

    <types model="business2">

    <types model="business3">

    <types model="business4">

    <!-- Loop3 sale -->
    <sales>
        <sale unit="Dollars" type="DELL123">
            <quater>1</quater>
            <income>12345</income>
        </sale>

        <sale unit="Dollars" type="DELL456">

        <sale unit="Dollars" type="HP123">

        <sale unit="Dollars" type="HP456">
    </sales>
</products>
```

The objective of this restructuring is to streamline the presentation of the products information to serve the manufacturing operations.

The output flow is expected to read as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<manufactures category="1" name="laptop">
    <sales unit="Dollars">
        <sale sales_type="DELL123">12345</sale>
        <sale sales_type="DELL456">6789</sale>
        <sale sales_type="HP123">12345.123</sale>
        <sale sales_type="HP456">6789.678</sale>
    </sales>
    <manufacture id="manu_1" date="2012-10-30" name="DELL"/>
    <manufacture id="manu_2" date="2012-10-28" name="HP"/>
    <types>
        <type>DELL123</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>DELL123</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>DELL456</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>DELL456</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>HP123</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>HP123</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>HP456</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>HP456</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
</manufactures>
```

In the output flow, the root element is changed to *manufactures*, the sales information is selected and consolidated into the *sale* element and the *manufacture* element is reduced to one single level.

To replicate this scenario, proceed as follows:

Dropping and linking the components

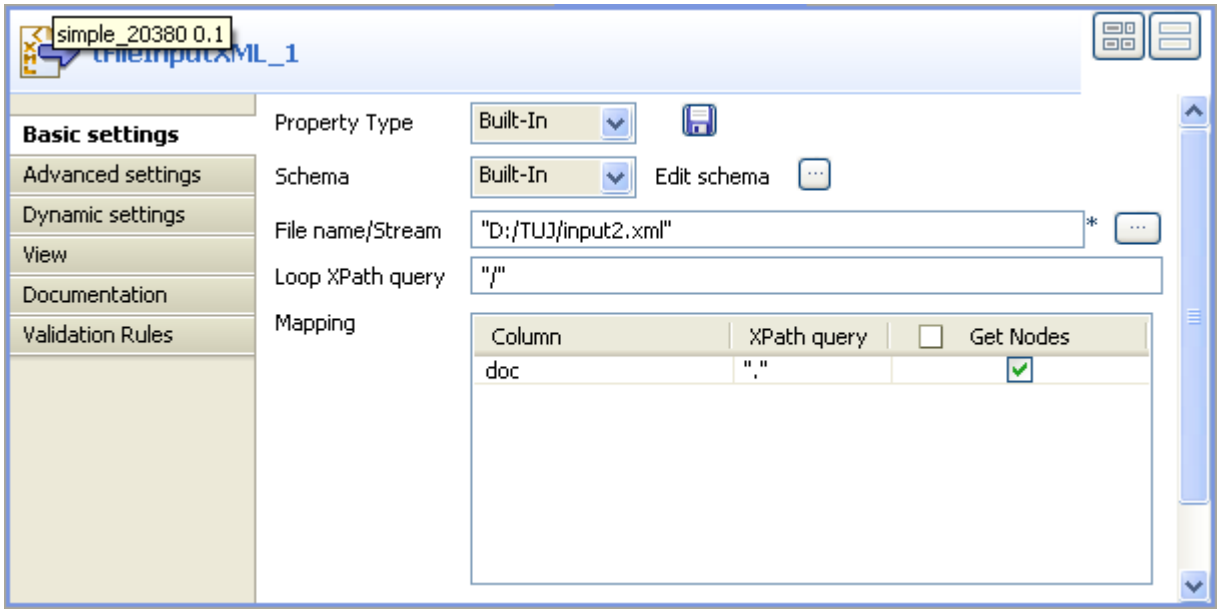
To do this, perform the following operations:

1. On the workspace, drop **tFileInputXML**, **tXMLMap**, **tLogRow** and **tFileOutputDelimited** from the **Palette**.
2. Right-click **tFileInputXML** to open its contextual menu and select the **Row** > **Main** link from this menu to connect this component to the **tXMLMap** component.
3. Repeat this operation to connect **tXMLMap** to **tLogRow** using **Row** > ***New output* (Main)** link. A dialog box pops up to prompt you to name this output link. In this scenario, name it as *outDoc*.
4. Do the same to connect **tLogRow** to **tFileOutputDelimited** using the **Row** > **Main** link.

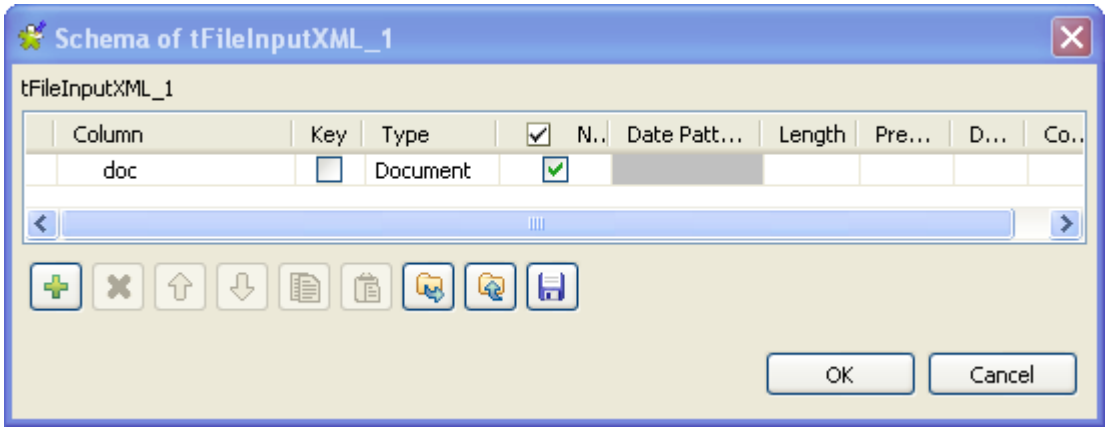
Configuring the input flow

To do this, do the following:

1. Double-click **tFileInputXML** to open its **Component** view.



2. Click the **[...]** button next to **Edit schema** to open the schema editor.



3. Click the **[+]** button to add one row to the editor and rename it as *doc*.
4. In the **Type** column, select **Document** from the drop-down list as the type of the input flow.
5. In the **File name / Stream** field, browse to, or type in the path to the XML source that provides the customer data.

6. In the **Loop XPath query** field, type in "/" to replace the default one. This means the source data is queried from the root.

7. In the **XPath query** column of the **Mapping** table, type in the XPath. In this scenario, type in ".", meaning that all of the data from source are queried.

8. In the **Get Nodes** column of the **Mapping** table, select the check box.

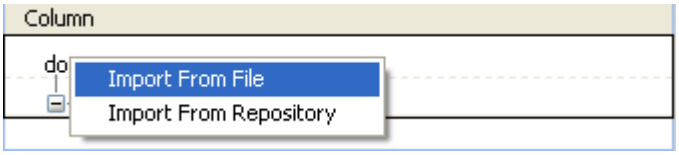
Configuring tXMLMap with multiple loops

To do this, proceed as follows:

1. Double-click the **tXMLMap** component to open the **Map Editor**.

   Note that the input area is already filled with the default basic XML structure and that the top table is the main input table.

2. In the left table, right-click *doc* to open the contextual menu.

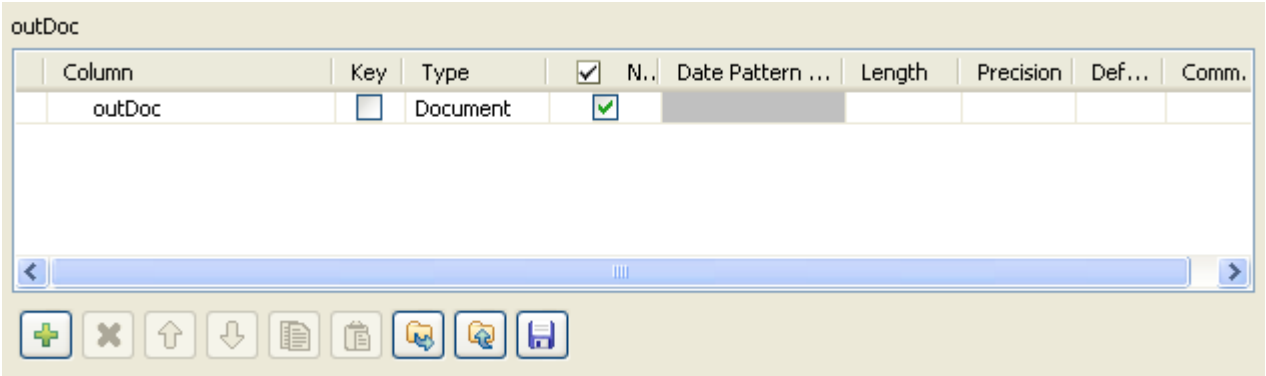


3. From this contextual menu, select **Import From File** and in the pop-up dialog box, browse to the corresponding source file in order to import therefrom the XML structure used by the data to be received by **tXMLMap**. In this scenario, the source file is *input2.xml*, which provides the data read and loaded by **tFileInputXML**.

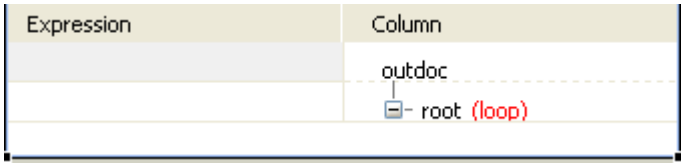
4. In the imported XML tree, right-click the *manufacture* node and select **As loop element** to set it as the loop element. Then do the same to set the *types* node and the *sale* node as loop element, respectively.

5. On the lower part of this map editor, click the **schema editor** tab to display the corresponding view.

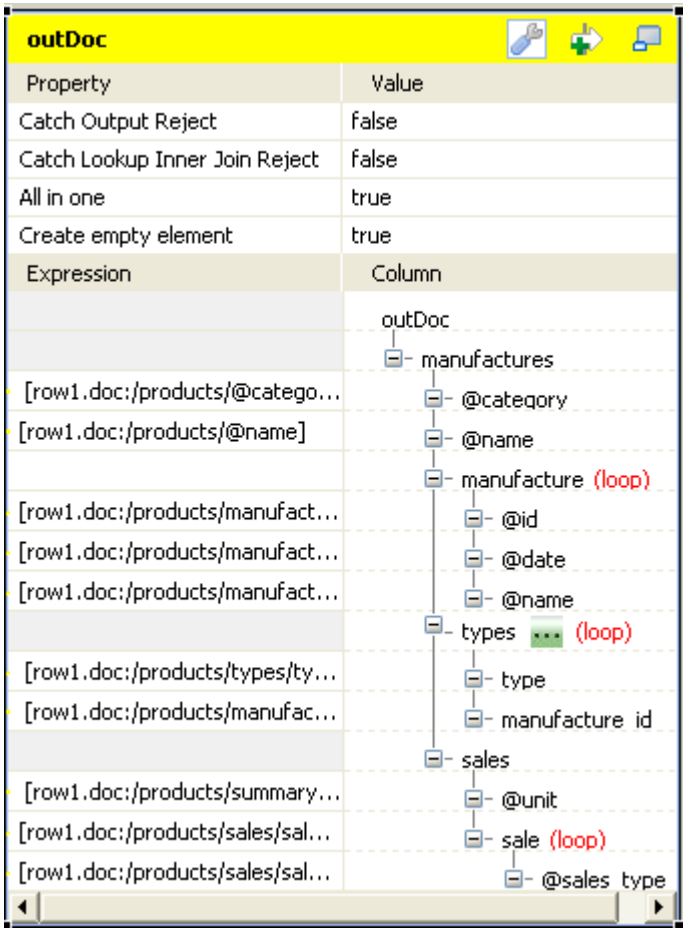
6. On the right side of this view, click the **[+]** button to add one row to the *outDoc* table and rename this row as *outDoc*.



7. In the **Type** column of this *outDoc* row, select **Document** as the data type. The corresponding XML root is added automatically to the top table on the right side which represents the output flow.



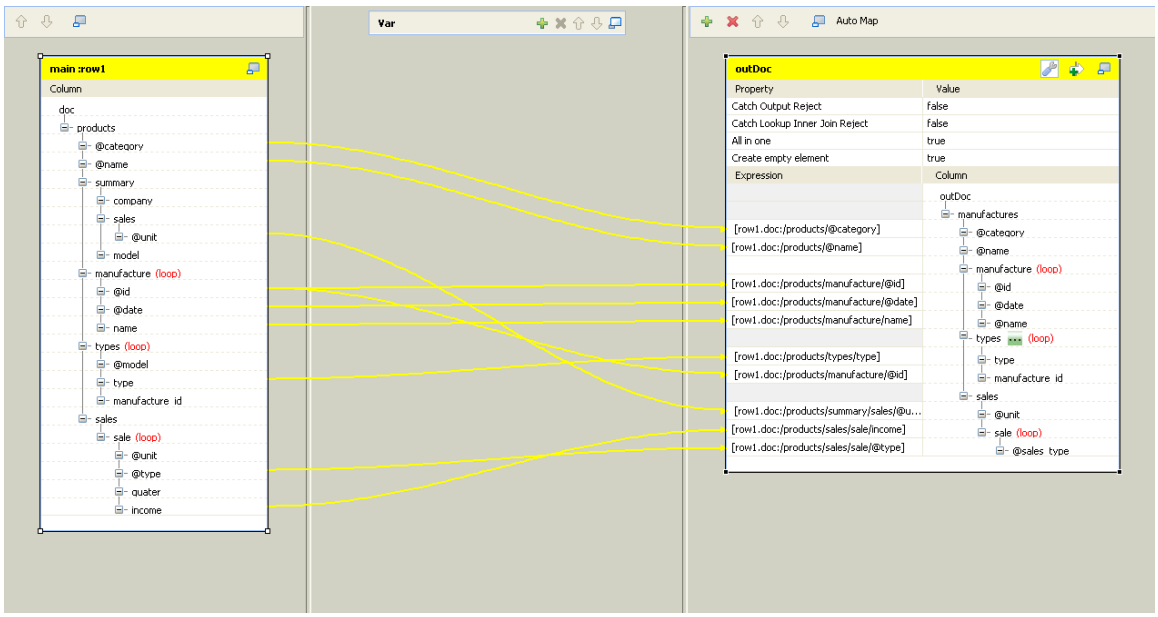
8. On the right side in the top table labelled *outDoc*, import the XML data structure that you need to use from the corresponding XML source file. In this scenario, it is *ref.xml*. This file provides the expected XML structure mentioned earlier.



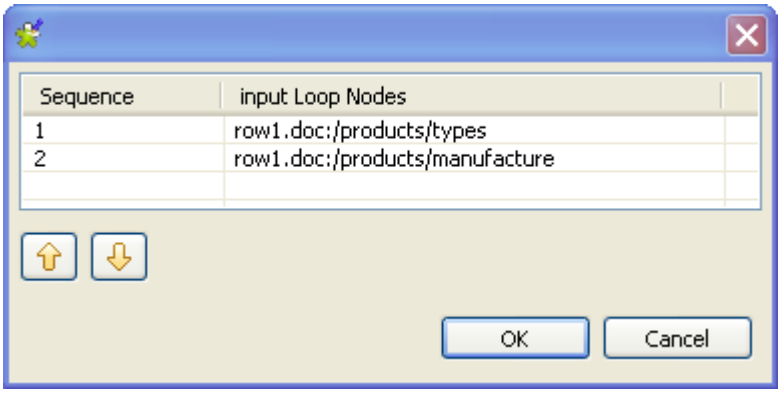
9. Right-click the *manufacture* node and select **As loop element** from the contextual menu. Then do the same to set the *types* node and the *sale* node as loop element, respectively.

   Then you can begin to map the input flow to the output flow.

10. In the top table on the input side (left) of the map editor, click the *@category* node and drop it to the **Expression** column in the row corresponding to the output row you need to map. In this scenario, it is the *@category* node.



11. Do the same to map:
    - *@name* to *@name*
    - *@unit* under the *summary* node to *@unit*
    - *@id* to *@id* and to *manufacture id*, respectively
    - *@date* to *@date*
    - *name* to *@name*
    - *type* to *type*
    - *@type* to *@sales_type*
    - *income* to *sale (loop)*
12. If required to generate single XML flow, click the wrench icon on top of the output side to open the setting panel and set the **All in one** feature as **true**. In this example, this option is set as **true**. For further information about the **All in one** feature, see *Talend Open Studio for Big Data User Guide*.
13. Click the **[...]** button next to the **types** loop element to open the loop sequence table. In this table, ensure that the *types* input loop is the primary loop, meaning that its sequence number is *1*. This way, the relative part of the output flow will be sorted with regards to the values of the *type* element.



When a loop element receives mappings from more than one loop element of the input flow, a **[...]** button appears next to this receiving loop element and allows you to set the sequence of the input loops. For example, in this scenario the

types loop element of the output flow is mapped with *@id* and *type* which belong to the *manufacture* loop element and the *types* loop element, respectively, so the **[...]** button appears beside this *types* loop element.

If the receiving flow is flat data, once it receives mappings from more than one loop element, this **[...]** button appears as well, on the head of the table representing the flat data flow, though.

14. Click **OK** to validate the mappings and close the **Map Editor**.

If you close the **Map Editor** without having set the required loop elements as described earlier in this scenario, the root element will be automatically set as loop element.

Configuring the output flow

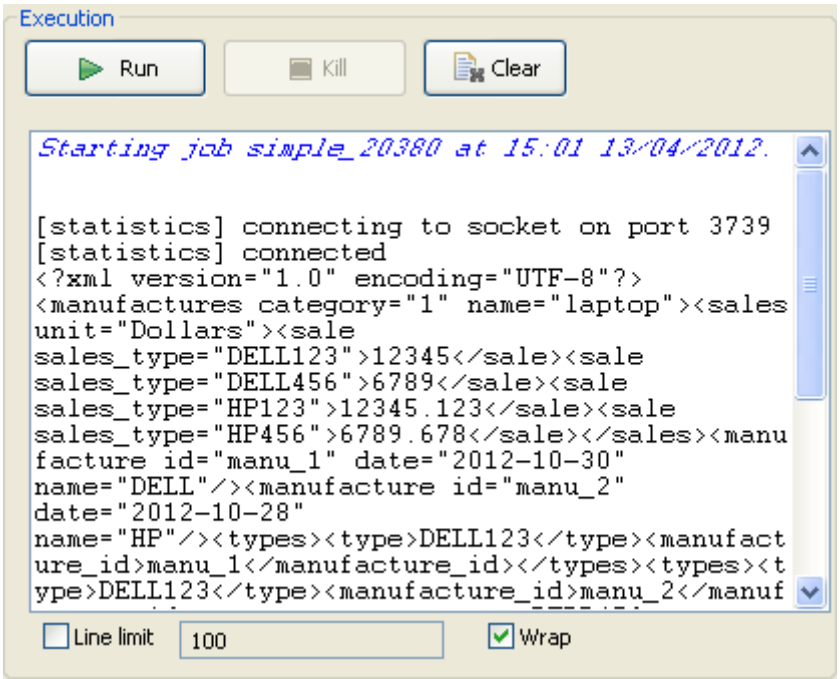
To do this, proceed as follows:

1. Double-click **tLogRow** to open its **Component** view.
2. If this component does not have the same schema of the preceding component, a warning icon appears. In this case, click the **Sync columns** button to retrieve the schema from the preceding one and once done, the warning icon disappears.
3. Click **OK** to validate these changes and accept the propagation prompted by the pop-up dialog box.
4. Double-click **tFileOutputDelimited** to open its **Component** view.
5. In the **File Name** field, browse to, or enter the path to the file you need to generate the output flow in.

Executing the Job

To execute this Job, press **F6**.

Once done, the **Run** view is opened automatically, where you can check the execution result.



Open the file generated, and you will see the expected products data restructured for manufacturing.

System components

This chapter details the main components that you can find in the **System** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The System family groups together components that help you to interact with the operating system.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tRunJob

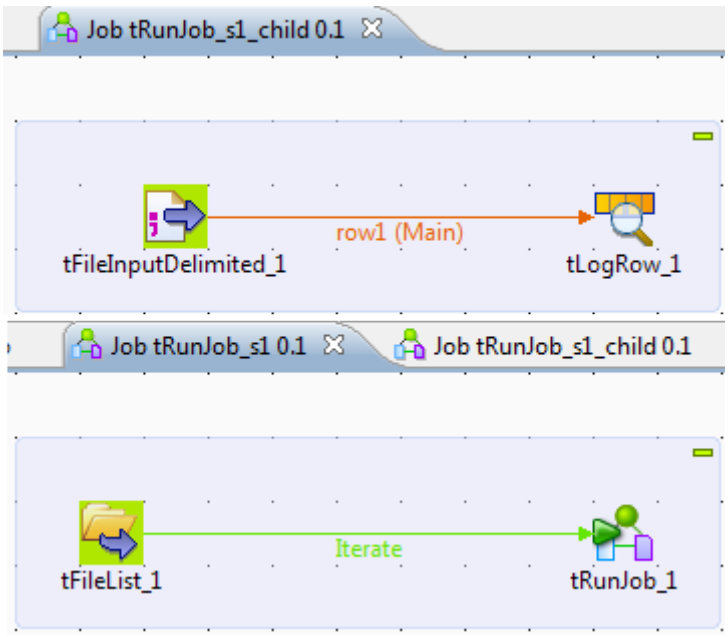
tRunJob Properties

Component family	System	
Function	**tRunJob** executes the Job called in the component's properties, in the frame of the context defined.	
Purpose	**tRunJob** helps mastering complex Job systems which need to execute one Job after another.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. . Click **Edit Schema** to make changes to the schema. If you are using *Talend Open Studio for Big Data*, only the **Built-in** mode is available.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use dynamic job	Select this check box to allow multiple Jobs to be called and processed. When this option is enabled, only the latest version of the Jobs can be called and processed. An independent process will be used to run the subjob. The **Context** and the **Use an independent process to run subjob** options disappear. *The **Use dynamic job** option is not compatible with the Jobserver cache. Therefore, the execution may fail if you run a Job that contains **tRunjob** with this check box selected in Talend Administration Center.* *This option is incompatible with the **Use or register a shared DB Connection** option of database connection components. When **tRunJob** works together with a database connection component, enabling both options will cause your Job to fail.*
	Context job	This field is visible only when the **Use dynamic job** option is selected. Enter the name of the Job that you want to call from the list of Jobs selected.
	CopyChild Job Schema	Click to fetch the child Job schema.
	Job	Select the Job to be called in and processed. Make sure you already executed once the Job called, beforehand, in order to ensure a smooth run through **tRunJob**.
	Context	If you defined contexts and variables for the Job to be run by the **tRunJob**, select the applicable context entry on the list.
	Use an independent process to run subjob	Select this check box to use an independent process to run the subjob. This helps in solving issues related to memory limits. *This option is not compatible with the Jobserver cache. Therefore, the execution may fail if you run a Job that contains **tRunjob** with this check box selected in Talend Administration Center.* *This option is incompatible with the **Use or register a shared DB Connection** option of database connection components. When **tRunJob** works together with a database connection component, enabling both options will cause your Job to fail.*

	Die on child error	Clear this check box to execute the parent Job even though there is an error when executing the child Job.
	Transmit whole context	Select this check box to get all the context variables from the parent Job. Deselect it to get all the context variables from the child Job.
	Context Param	You can change the selected context parameters. Click the plus button to add the parameters as defined in the Context of the child Job. For more information on context parameters, see *Talend Open Studio for Big Data User Guide*.
Advanced settings	*Print Parameters*	Select this check box to display the internal and external parameters in the **Console**.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component can be used as a standalone Job or can help clarifying complex Job by avoiding having too many sub-jobs all together in one Job.	
Global Variables		**Child return code**: Indicates the Java return code of the child Job. This is available as an **After** variable. Returns an integer: - if no errors > the code value is 0. - if errors > an exception message shows. **Child exception stack trace**: Returns a Java stack trace from a child Job. This is available as an **After** variable. Returns a string. For further information about variables, see *Talend Open Studio for Big Data User Guide*.
		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Limitation	n/a	

Scenario: Executing a child Job

This scenario describes a two-component Job that calls another Job, which is the child Job, to display the content of files specified in the parent Job on the **Run** log console.



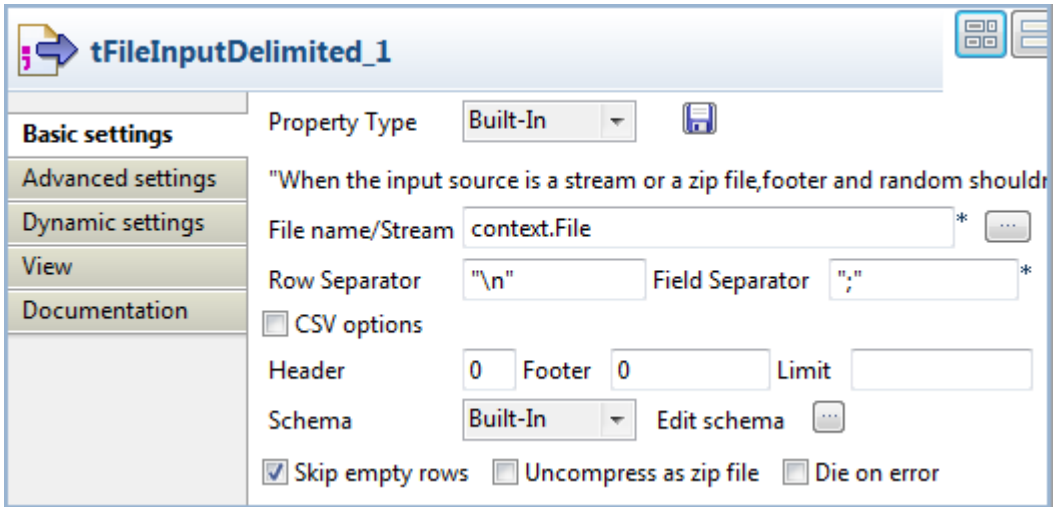
Creating the child Job

Dropping and linking components

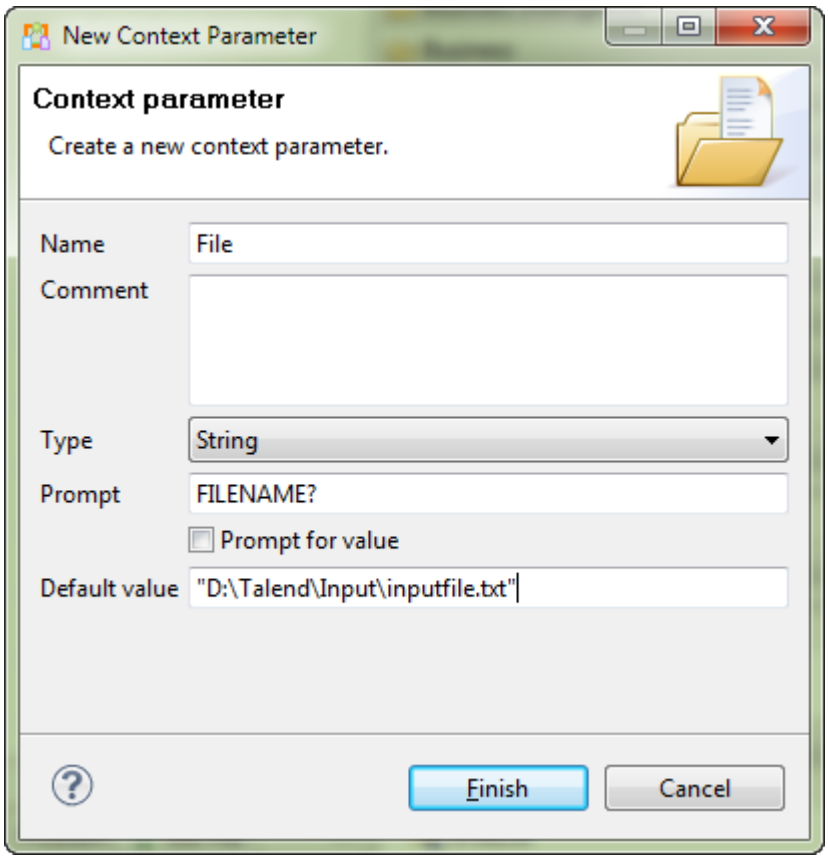
1. Drop a **tFileInputDelimited** and a **tLogRow** from the **Palette** to the design workspace.
2. Connect the two components together using a **Row** > **Main** link.

Configuring the components

1. Double-click **tFileInputDelimited** to open its **Basic settings** view and define its properties.



2. Click in the **File Name** field and then press **F5** to open the **[New Context Parameter]** dialog box and configure the context variable.



3. In the **Name** field, enter a name for this new context variable, *File* in this example.
4. In the **Default value** field, enter the full path to the default input file.
5. Click **Finish** to validate the context parameter setup and fill the **File Name** field with the context variable.

   You can also create or edit a context parameter in the **Context** tab view beneath the design workspace. For more information, see *Talend Open Studio for Big Data User Guide*.

6. Click the **[...]** button next to **Edit schema** to open the **[Schema]** dialog box where you can configure the schema manually.
7. In the dialog box, click the **[+]** button to add columns and name them according to the input file structure.

   In this example, this component will actually read files defined in the parent Job, and these files contain up to five columns. Therefore, add five string type columns and name them *col_1*, *col_2*, *col_3*, *col_4*, and *col_5* respectively, and then click **OK** to validate the schema configuration and close the **[Schema]** dialog box.
8. Double-click **tLogRow** to display its **Basic settings** view and define its properties.
9. Select the **Table** option to view displayed content in table cells.

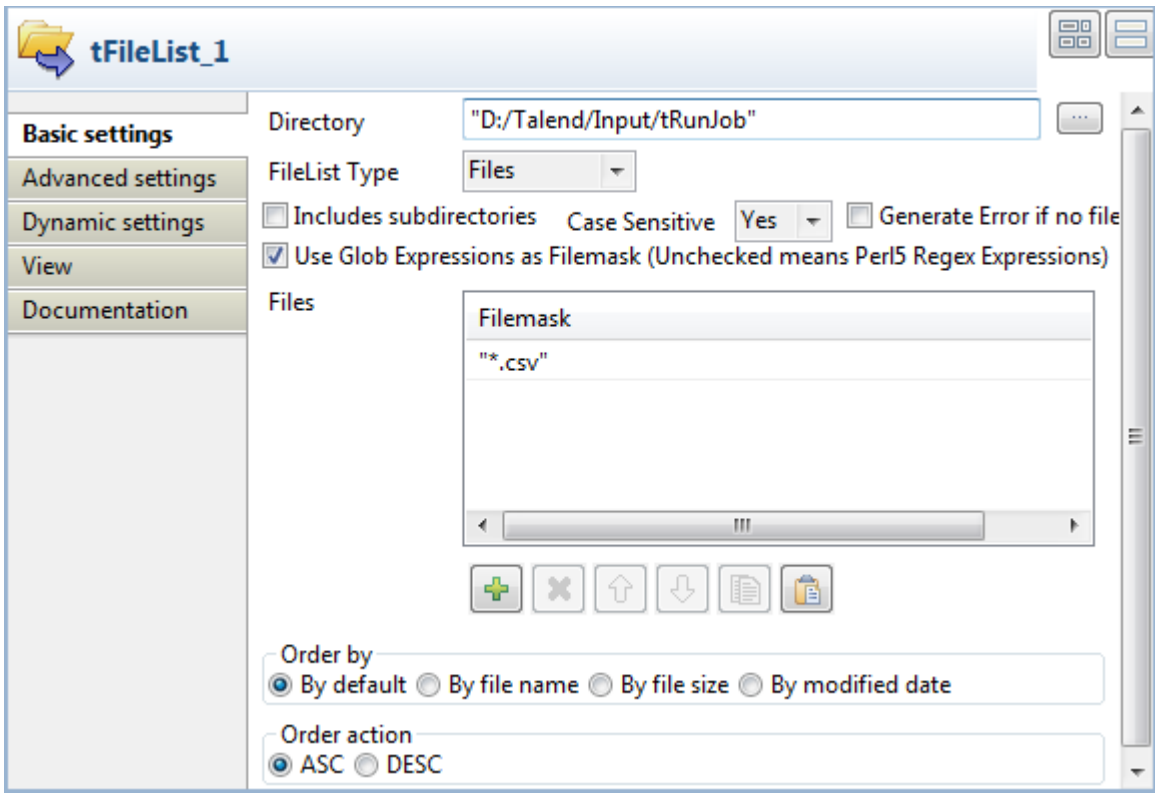
Creating the parent Job

Dropping and linking components

1. Drop a **tFileList** and a **tRunJob** from the **Palette** to the design workspace.
2. Connect the two components together using an **Iterate** link.

Configuring the components

1. Double-click **tFileList** to open its **Basic settings** view and define its properties.



2. In the **Directory** field, specify the path to the directory that holds the files to be processed, or click the **[...]** button next to the field to browse to the directory.

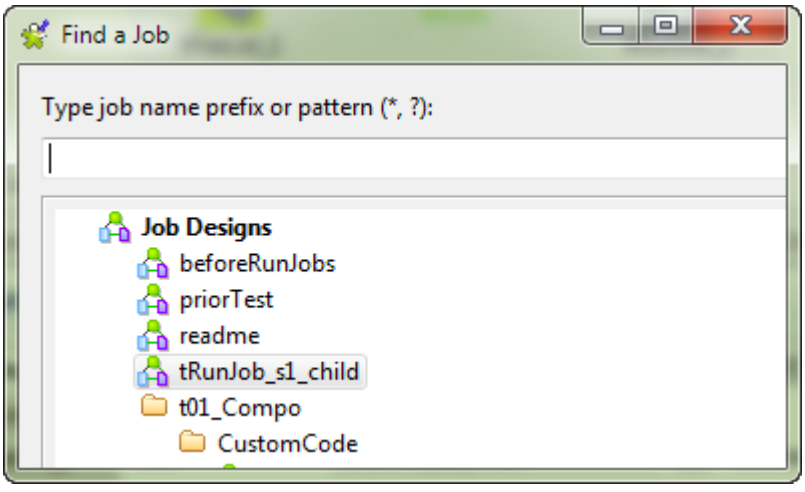
   In this example, the directory is called *tRunJob* and it holds three delimited files with up to five columns.

3. In the **FileList Type** list, select **Files**.

4. Check that the **Use Glob Expressions as Filemask** check box is selected, and then click the **[+]** button to add a line in the **Files** area and define a filter to match files. In this example, enter `"*.csv"` to retrieve all delimited files.

5. Double-click **tRunJob** to display its **Basic settings** view and define its properties.

6. Click the **[...]** button next to the **Job** field to open the **[Find a Job]** dialog box.



7. Select the child Job you want to execute and click **OK** to close the dialog box. The name of the selected Job appears in the **Job** field.

8. In the **Context Param** area, click the plus button to add a line and define the context parameter. The only context parameter defined in the child Job, named *File*, appears in the **Parameter** cell.

9. Click in the **Values** cell, press **Ctrl+Space** on your keyboard to access the list of context variables, and select *tFileList-1.CURRENT_FILEPATH*.

   The corresponding context variable appears in the **Values** cell: `((String)globalMap.get("tFileList-1.CURRENT_FILEPATH"))`.

   For more information on context variables, see *Talend Open Studio for Big Data User Guide*.

Executing the parent Job

1. Press **Ctrl+S** to save your Job.

2. Press **F6** to execute the Job.

```
.--------------------+-----+-----+-----+-----.
|                  tLogRow_1                  |
|====================+=====+=====+=====+=====|
|col_1               |col_2|col_3|col_4|col_5|
|====================+=====+=====+=====+=====|
|email               |     |     |     |     |
|mmoore@hotmail.com  |     |     |     |     |
|ataylor@gmail.com4  |     |     |     |     |
|canderson@foxmail.com|     |     |     |     |
'--------------------+-----+-----+-----+-----'

.-----+----------+---------+---------+------.
|                  tLogRow_1                  |
|=====+==========+=========+=========+======|
|col_1|col_2     |col_3    |col_4    |col_5 |
|=====+==========+=========+=========+======|
|id   |firstName |lastName |company  |city  |
|1    |Michael   |Jackson  |IBM      |Roma  |
|2    |Elisa     |Black    |Microsoft|London|
```

   The parent Job calls the child Job, which reads the files defined in the parent Job, and the content of the files is displayed on the **Run** console.

Related topic: section *tLoop*, and section *Scenario 1: Buffering data (Java)* of the **tBufferOutput** component.

tSetEnv

tSetEnv Properties

Component family	System	
Function	**tSetEnv** adds variables temporarily to system environment during the execution of a Job.	
Purpose	**tSetEnv** allows to create variables and execute a Job script through communicating the information about the newly created variables between subjobs. After job execution, the newly created variables are deleted.	
Basic settings	*Parameters*	**Click the plus button to add the variables needed for the job.** **name**: Enter the syntax for the new variable. **value**: Enter a parameter value according to the context. **append**: Select this check box to add the new variable at the end.
Usage	**tSetEnv** can be used as a start or an intermediate component.	
Limitation	n/a	

Scenario: Modifying a variable during a Job execution

The following scenario is made of two Jobs parent and child. With the **tSetEnv** component, you can transfer and modify in a child Job a value created in a parent Job. As part of this Job, the **tMsgBox** components allow you to display, for information purposes only, that a variable is properly set, via an info-box.

To modify the value of the parent Job by using a variable set in the **tSetEnv** component, do as described in the following sections:

Drop and link components

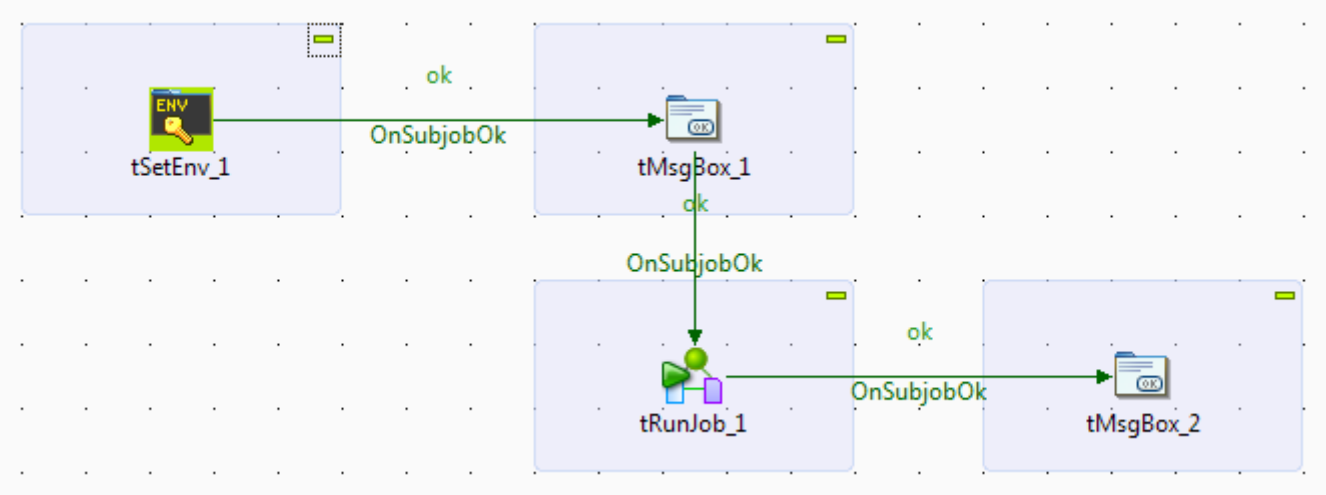
1. Create a first Job named *parentJob*: right-click on the **Job Design** node of the **Repository**, and choose **Create a Job**.

2. From the **Palette**, drop a **tSetEnv** component, two **tMsgBox** components, and one **tRunJob** component onto the design workspace.

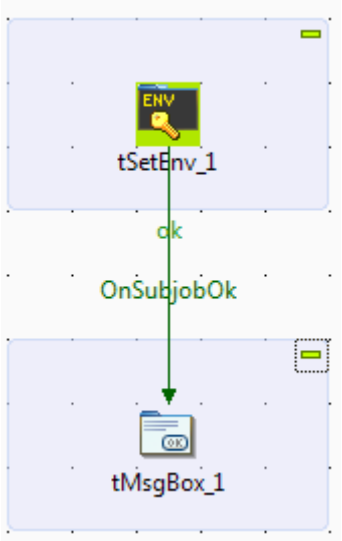
3. Connect the **tSetEnv** component to a first **tMsgBox** component with a **OnSubjobOk** link : right-click on the start component, select *Trigger*, then **OnSubjobOk**. Then click on the end component you want to connect.

4. Connect the first **tMsgBox** component to the **tRunJob** with a **OnSubjobOk** link.

5. Then connect the **tRunJob** component to the second **tMsgbox** with a **OnSubjobOk** link.



6. Now create a child Job named *ChildJob*.
7. From the **Palette**, drop a **tSetEnv** component onto the design workspace.
8. Connect the **tSetEnv** component to the **tMsgBox** with a **OnSubjobOk** link : right click on the start component, select *Trigger*, then **OnSubjobOk**. Then click on the end component you want to connect.



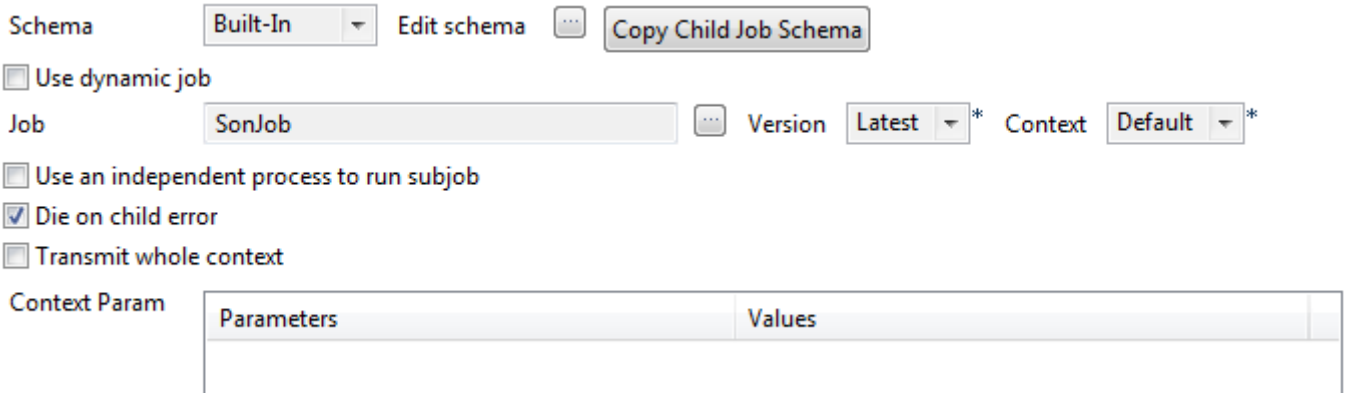
Set the components

In this example, the value set in the parent Job is transferred to the child Job. There, it is modified and adopts the value of the child Job, and then transferred to the parent Job again.

1. In *parentJob*, select the **tSetEnv** component and click the **Component** tab. Add a variable row by clicking the **[+]** button to set the initial value of the variable. Type *Variable_1* in the **Name** field, and *Parent job value* in the **Value** field.
2. Select the first **tMsgBox** component, and click the **Component** tab. In the **Message** field, type the message displayed in the info-box which confirms that your variable has properly been taken into account. For example: `"Parent:"+System.getProperty("Variable_1")` displays the variable set in the **tSetEnv** component (here *Parent job value*).

3. Select the second **tMsgBox** component, and click the **Component** tab. In the **Message** field, type the `"Parent:"+System.getProperty("Variable_1")` line again. It makes the variable set in the child Job appear.

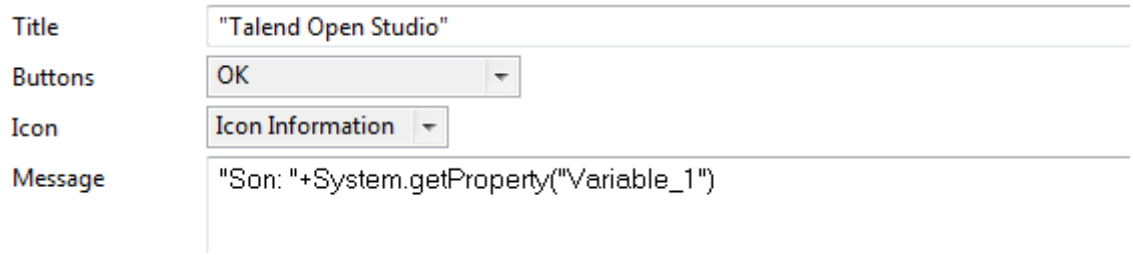
4. Select the **tRunJob** component and click the **Component** tab. In the **Job** field, type the name of your child Job, here *ChildJob*. This will run the child Job when you run the parent Job.



5. Now double-click the **tRunJob** component to open the child Job *ChildJob*.

6. Select the **tSetEnv** component, and click the **Component** tab. Add a variable row by clicking the **[+]** button to set the initial value of the variable. Type *Variable_1* in the **Name** field, and *Child job value* in the **Value** field.

7. Select the **tMsgBox** component and click the **Component** tab. In the **Message** field, type the message displayed in the info-box which confirms that your variable has properly been taken into account. For example: `"Parent:"+System.getProperty("Variable_1")` displays the variable set in the **tSetEnv** component (here *Child job value*).



8. Save your Job, go back to *parentJob*, then run the Job by pressing **F6**.

Run the Job

Three info-boxes are displayed one after the other:

- *Parent: Parent job value*: parent Job's value is *Parent job value*.
- *Child: Child job value*: Child Job's value is *Child job value*.
- *Parent: Parent job value*: parent Job's value was modified by the variable set in the **tSetEnv** of the child Job, then transferred again to the parent Job. parent Job's value is now the one set in the child Job.

tSSH

tSSH Properties

Component family	System	
Function	Returns data from a remote computer, based on the secure shell command defined.	
Purpose	Allows to establish a communication with distant server and return securely sensible information.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the preceding component in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* **User Guide**
	Host	IP address
	Port	Listening port number
	User	User authentication information
Authentication method	*Public Key/Key Passphrase/ Private Key*	Select the relevant option. In case of **Public Key**, type in the passphrase, if required, in the **Key Passphrase** field and then in the **Private key** field, type in the private key or click the three dot button next to the **Private key** field to browse to it.
Authentication method	*Password/Password*	Select the relevant option. In case of **Password**,type in the required password in the **Password** field.
Authentication method	*Keyboard Interactive/ Password*	Select the relevant option. In case of **Keyboard Interactive**, type in the required password in the **Password** field.
	Pseudo terminal	Select this check box to call the interactive shell that performs the terminal operations.
	Command separator	Type in the command separator required. Once the **Pseudo terminal** check box is selected, this field becomes unavailable.
	Commands	Type in the command for the relevant information to be returned from the remote computer. When you select the **Pseudo terminal** check box, this table becomes a terminal emulator and each row in this table is a single command.
	Use timeout/timeout in seconds	Define the timeout time period. A timeout message will be generated if the actual response time exceeds this expected processing time.
	Standard Output	Select the destination to which the standard output is returned. The output may be returned to:

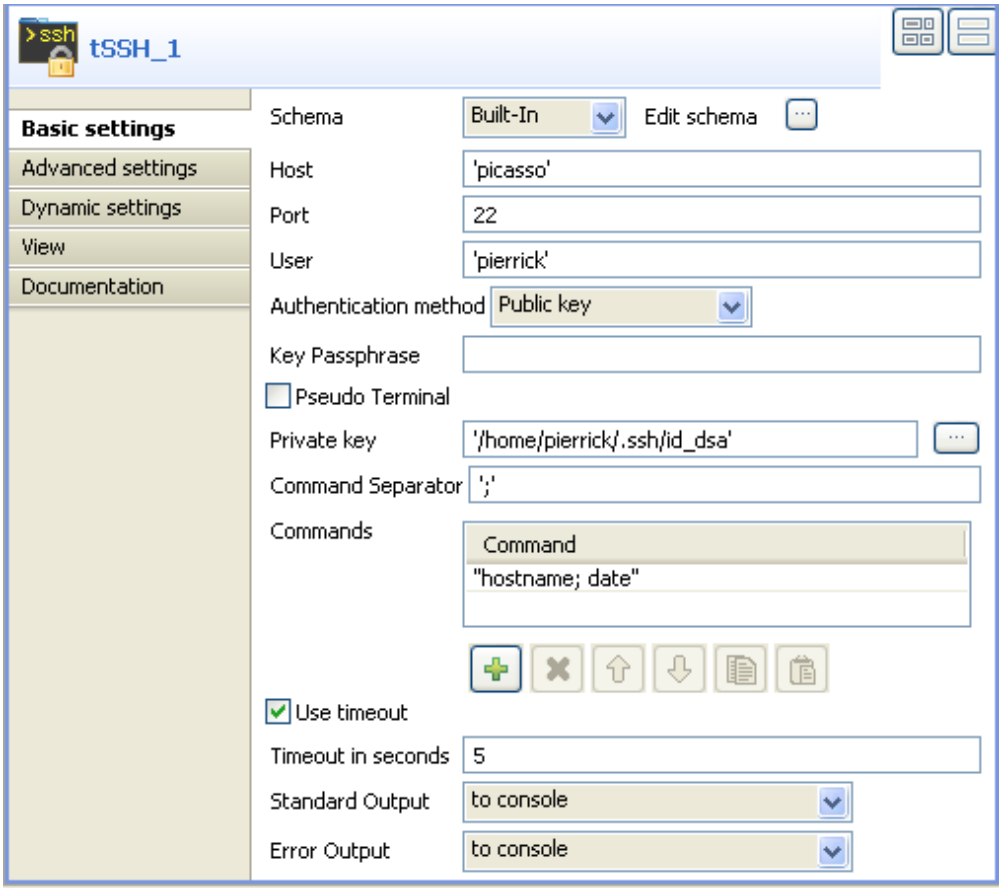
		- **to console**: the output is displayed in the console of the **Run** view. - **to global variable**: the output is indicated by the corresponding global variable. - **both to console and global variable**: the output is indicated both of the two means. - **normal**: the output is a standard ssh output.
	Error Output	Select the destination to which the error output is returned. The output may be returned to: - **to console**: the output is displayed in the console of the **Run** view. - **to global variable**: the output is indicated by the corresponding global variable. - **both to console and global variable**: the output is indicated both of the two means. - **normal**: the output is a standard ssh output.
Usage	This component can be used as standalone component.	
Global variables		**Standard Output**: Indicates the standard execution output of the remote command. It is available as an **After** variable. Returns a String. **Error output**: Indicates the error execution output of the remote command. It is available as an **After** variable. Returns a String. **Exit value**: Indicates the exit status of the remote command. It is available as an **After** variable. Returns an Integer. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Main **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. Incoming links (from one component to another): **Row**: Main; Iterate **Trigger**: Run if; On Component Ok; On Component Error; On Subjob Ok; On Subjob Error. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	The component use is optimized for Unix-like systems.	

Scenario: Remote system information display via SSH

The following use case describes a basic Job that uses SSH command to display the hostname of the distant server being connected to, and the current date on this remote system.

The **tSSH** component is sufficient for this Job. Drop it from the **Palette** to the design workspace.

Double-click on the **tSSH** component and select the **Basic settings** view tab.



- Type in the name of the **Host** to be accessed through SSH as well as the **Port** number.
- Fill in the **User** identification name on the remote machine.
- Select the **Authentication method** on the list. For this use case, the authentication method used is the public key.
- Thus fill in the corresponding **Private key**.
- On the **Command** field, type in the following command. For this use case, type in `hostname; date` between double quotes.
- Select the **Use timeout** check box and set the time before falling in error to 5 seconds.

```
Starting job uniteElisa at 16:26 26/09/2007.
picasso
Wed Sep 26 14:24:15 CEST 2007
Job uniteElisa ended at 16:26 26/09/2007. [exit code=0]
```

The remote machine returns the host name and the current date and time as defined on its system.

tSystem

tSystem Properties

Component family	System	
Function	**tSystem** executes one or more system commands.	
Purpose	**tSystem** can call other processing commands, already up and running in a larger Job.	
Basic settings	*Use home directory*	Select this check box to change the name and path of a dedicated directory.
	Use Single Command	When the required command is very simple, to the degree that, for example, only one parameter is used and without space, select this option to activate its **Command** field. In this field, enter the simple system command. Note that the syntax is not checked. *In Windows, the MS-DOS commands do not allow you to pass directly from the current folder to the folder containing the file to be launched. To launch a file, you must therefore use an initial command to change the current folder, then a second one to launch the file*

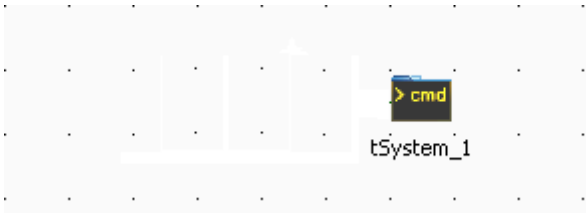
	Use Array Command	Select this option to activate its **Command** field. In this field, enter the system command in array, one parameter per line.
		
		For example, enter the following command with consecutive spaces in array for Linux:
		
		`"cp"`
		`"/temp/source.txt"`
		`"/temp/copy  to/"`

	Standard Output and *Error Output*	Select the type of output for the processed data to be transferred to.
		to console: data is passed on to be viewed in the **Run** view.
		to global variable: data is passed on to an output variable linked to the **tSystem** component.
		to console and to global variable: data is passed on to the **Run** view and to an output variable linked to the **tSystem** component.
		normal: data is passed on to the component that comes next.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the preceding component in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* **User Guide**
	Environment variables	Click the **[+]** button to add as many global variables as needed. **name**: Enter the syntax of the new variable.

		value: Enter a value for this variable according to the context.
Usage	This component can typically used for companies which already implemented other applications that they want to integrate into their processing flow through Talend.	
Global Variables		**Standard Output**: Returns the standard output from a process. This is available as an **After** variable Returns a string. **Error Output**: Returns the erroneous output from a process. This is available as an **After** variable. Returns a string. **Exit Value**: Returns an exit code. This is available as an **After** variable. Returns an integer: - if there are no errors > the exit code is 0. - if there are errors > the exit code is 1. For further information about variables, see *Talend Open Studio for Big Data* User Guide.
Connections		Outgoing links (from one component to another): **Row**: Main. **Trigger**: On Subjob Ok; On Subjob Error; Run if. Incoming links (from one component to another): **Row**: Main; Reject; Iterate. **Trigger**: On Subjob Ok; On Subjob Error; Run if; On Component Ok; On Component Error; Synchronize; Parallelize. For further information regarding connections, see *Talend Open Studio for Big Data* User Guide.
Limitation	n/a	

Scenario: Echo ‘Hello World!’

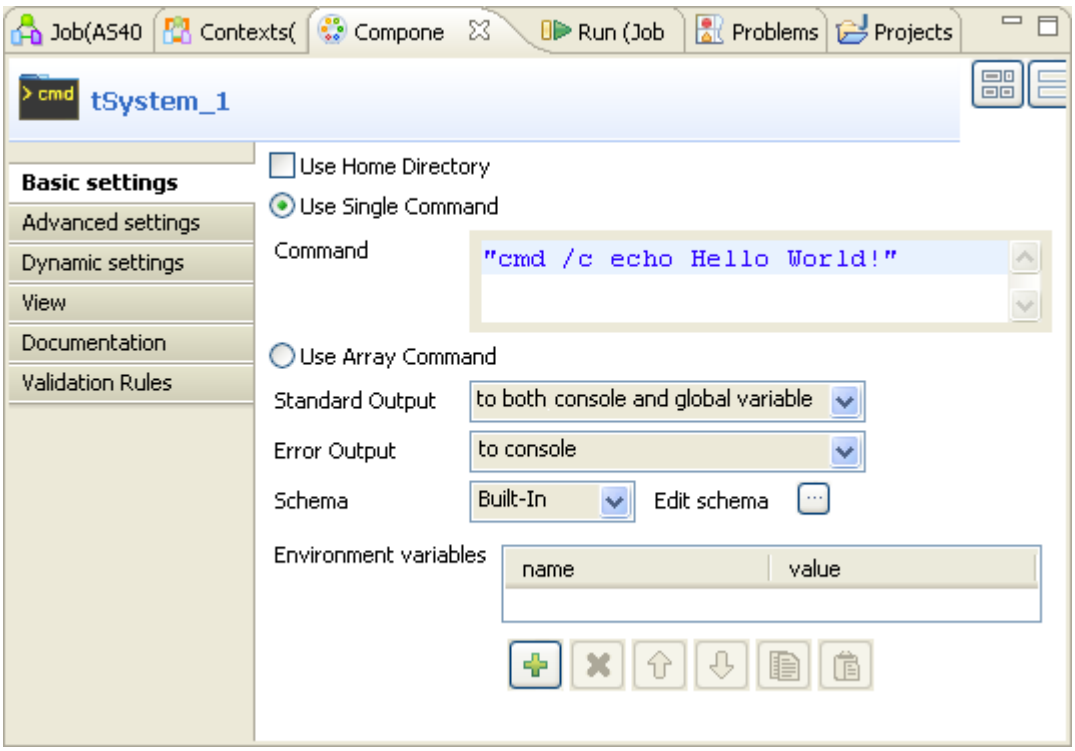
This scenario is one single component **tSystem** to execute a system command and shows the results in the **Run** view “console”.



To replicate this scenario, proceed as follows:

1. Drop a **tSystem** component from the **Palette** to the design workspace.

2. Double-click **tSystem** to open its **Component** view.



3. Select the **Use Single Command** option to activate its **Command** field and type in `"cmd /c echo Hello World!"`.
4. In the **Standard Output** drop-down list, select **to both console and global variable**.
5. Press **F6** to run this Job.

```
Starting job tSystem_scenario at 17:26 08/10/2009.

[statistics] connecting to socket on port 3961
[statistics] connected
Hello World!
[statistics] disconnected
Job tSystem_scenario ended at 17:26 08/10/2009. [exit code=0]
```

The Job executes an echo command and shows the output in the Console of the **Run** view.

Talend MDM components

This chapter details the main components that you can find in the **Talend MDM** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The **Talend MDM** family groups together connectors that read and write master data in the MDM Hub.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tMDMBulkLoad

tMDMBulkLoad properties

Component family	Talend MDM	
Function	**tMDMBulkLoad** writes XML structured master data into the MDM hub in bulk mode.	
Purpose	This component uses bulk mode to write data so that big batches of data or data of high complexity can be fast uploaded onto the MDM server.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in. Click **Sync columns** to collect the schema from the previous component.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and Job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	XML field	Select the name of the column in which you want to write the XML data.
	URL	Type in the URL required to access the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the Version of master data you want to connect to, for which you have the required user rights. Leave this field empty if you want to display the default Version of master data.
	Data model	Type in the name of the data model against which the data to be written is validated.
	Data Container	Type in the name of the data container where you want to write the master data.
	Entity	Type in the name of the entity that holds the data record(s) you want to write.
	Validate	Select this checkbox to validate the data you want to write onto the MDM server against validation rules defined for the current data model. For more information on how to set the validation rules, see *Talend Open Studio for MDM* Administrator Guide. *If you need faster loading performance, do not select this checkbox.*
	Generate ID	Select this check box to generate an ID number for all of the data written. *If you need faster loading performance, do not select this checkbox.*
	Commit size	Type in the row count of each batch to be written onto the MDM server.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Connections		Outgoing links (from one component to another):

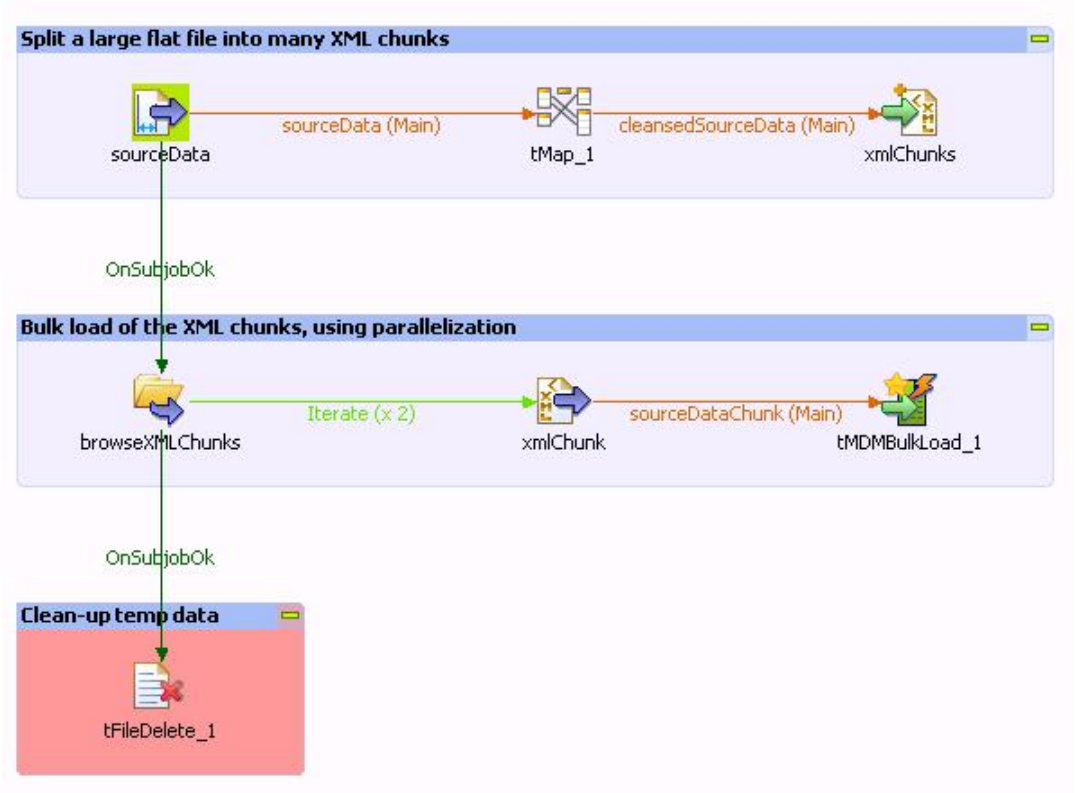
		Row: Main, **Trigger**: Run if; On Component Ok; On Component Error, On Subjob Ok, On Subjob Error. Incoming links (from one component to another): **Row**: Main **Trigger**: Run if, On Component Ok, On Component Error, On Subjob Ok, On Subjob Error For further information regarding connections, see *Talend Open Studio for Big Data* **User Guide**.
Usage	This component needs always an incoming link to offer XML structured data. If your data offered is not yet in the XML structure, you need use components like **tWriteXMLField** to transform this data into the XML structure. For further information about **tWriteXMLField**, see section *tWriteXMLField*.	

Enhancing the MDM bulk data load

The information below concerns only MDM used with eXist.

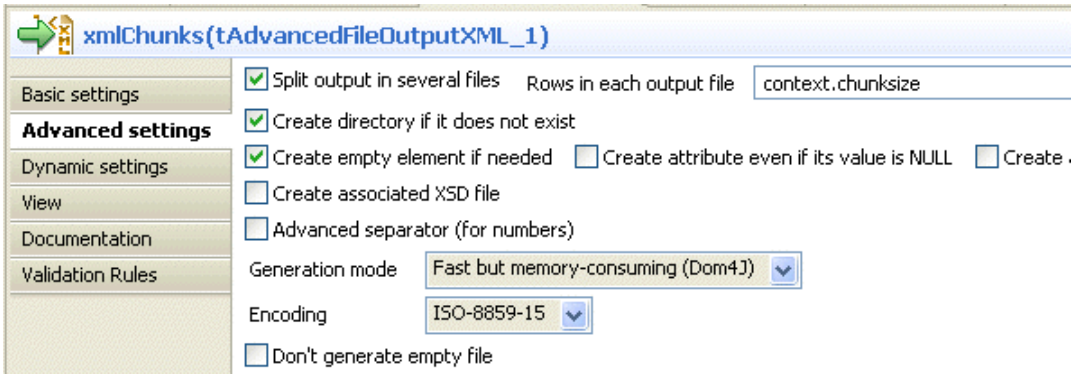
As XML parsing is a CPU and memory consuming process, it is not really compatible with large datasets. The following Scenario: section *Scenario: Loading records into a business entity*, which shows how to use the **tMDMBulkLoad** component, has some limitations because it cannot work with large dataset, for the time being at least.

An alternative scenario in which you process the dataset file per bulk load iterations can be designed as the following:



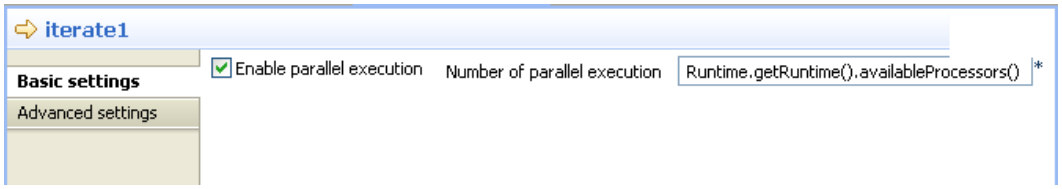
In such a scenario, the **tMDMBulkLoad** component waits for XML data as an input. You must manually format this incoming data to match the entity schema defined in the MDM Studio. Most of the time, the data you want to import is in a flat “format”, and you have to transform it into XML.

As XML parsing is memory consuming, you can workaround this problem by splitting your source file into several files using the **tAdvancedFileOutputXML** component. To do this, you select the **Split output in several files** option in the **Advaced settings** view of the component and then set the rows in each output file through a context variable (*context.chunkSize*), for example.



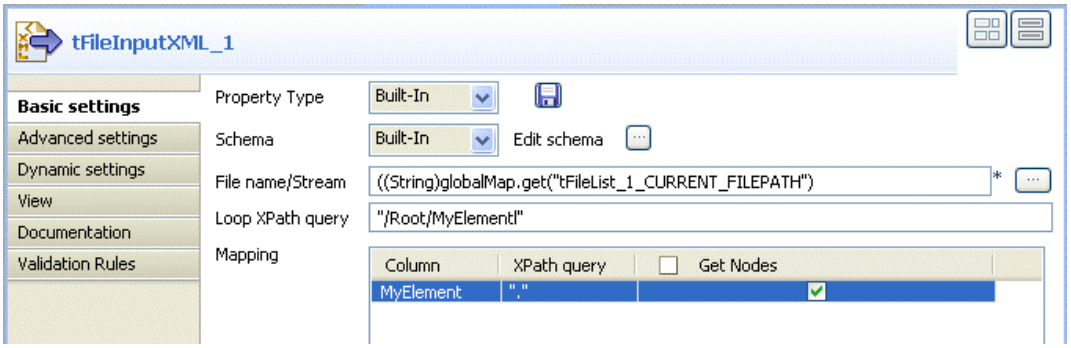
The XML schema you must define in the XML editor of this component should be an exact match of the business entity defined in the MDM Studio. The XML schema in the editor must represent a single `<root>` element which contains all the other elements, so that you can loop on each of the element. The path of the file should be defined in a temporary folder.

Use a **tFileList** component to read all the XML files that have just been created. This component enables you to parallelize the process. Connect it to a **tFileInputXML** component using the **Iterate** link.

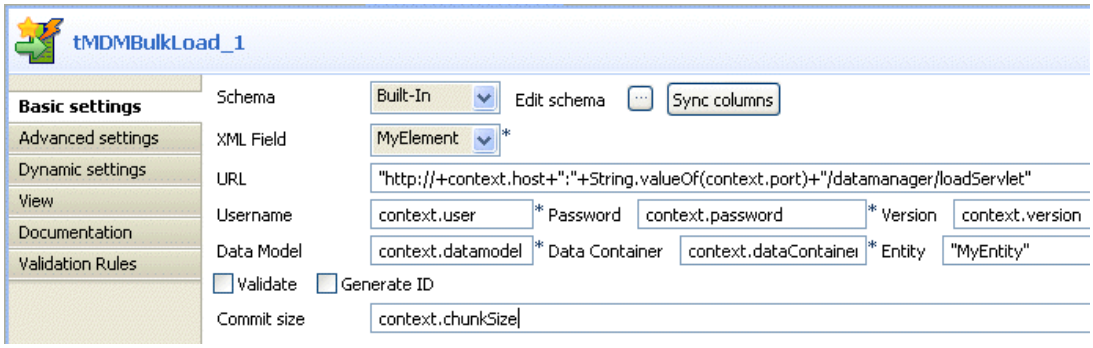


For the **Iterate** link, it is recommended that you set as many threads as the number of the physical cores of the computer. You can achieve that using `Runtime.getRuntime().availableProcessors()`

The **tFileInputXML** component will read the data from the XML files you have created, by defining a loop on the elements, and getting all the nodes that are already formatted as XML. You must then select the **Get Nodes** check box.



Finally, you must setup the **tMDMBulkLoad** component as the following:



Ensure that you set the commit size to the same value you defined in the **tAdvancedfileOutputXML**, the *context.chunkSize* context variable.

The **tFiledelete** component in such a scenario will delete all the temporary data at the end of the Job.

Scenario: Loading records into a business entity

This scenario describes a Job that loads records into the *ProductFamily* business entity defined by a specific data model in the MDM hub.

Prerequisites of this Job:

- The *Product* data container: this data container is used to separate the product master data domain from the other master data domains.
- The *Product* data model: this data model is used to define the attributes, validation rules, user access rights and relationships of the entities of interest. Thus it defines the attributes of the *ProductFamily* business entity.
- The *ProductFamily* business entity: this business entity contains *Id*, *Name*, both defined by the *Product* data model.

For further information about how to create a data container, a data model, and a business entity along with its attributes, see *Talend Open Studio for MDM* Administrator Guide.

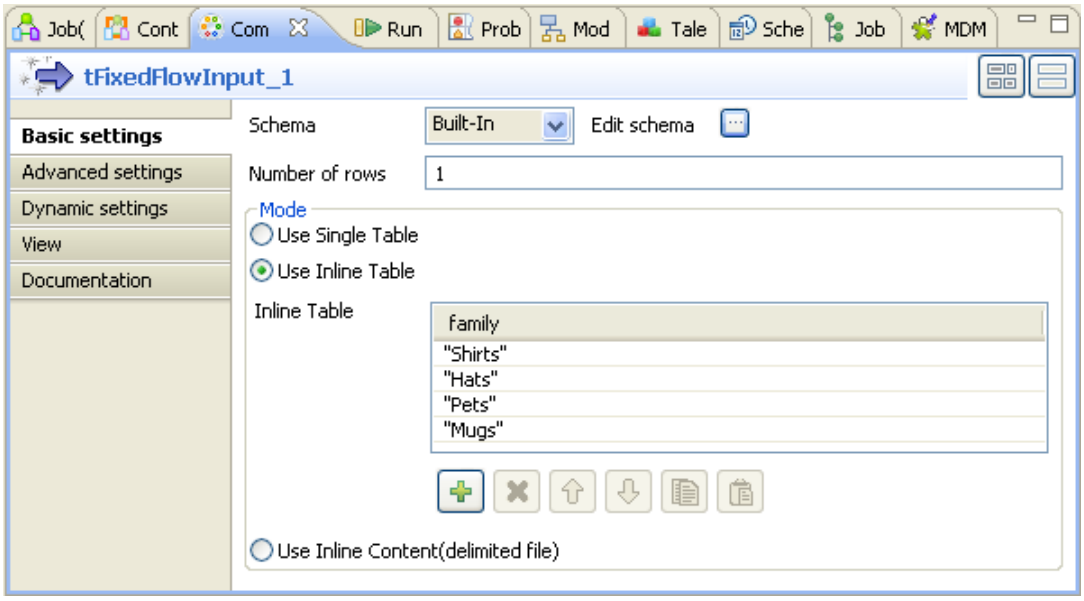
The Job in this scenario uses three components.

- **tFixedFlowInput**: this component generates the records to be loaded into the *ProductFamily* business entity. In the real case, your records to be loaded are often voluminous and stored in a specific file, while in order to simplify the replication of this scenario, this Job uses **tFixedFlowInput** to generate four sample records.
- **tWriteXMLField**: this component transforms the incoming data into XML structure.
- **tMDMBulkLoad**: this component writes the incoming data into the *ProductFamily* business entity in bulk mode, generating ID value for each of the record data.

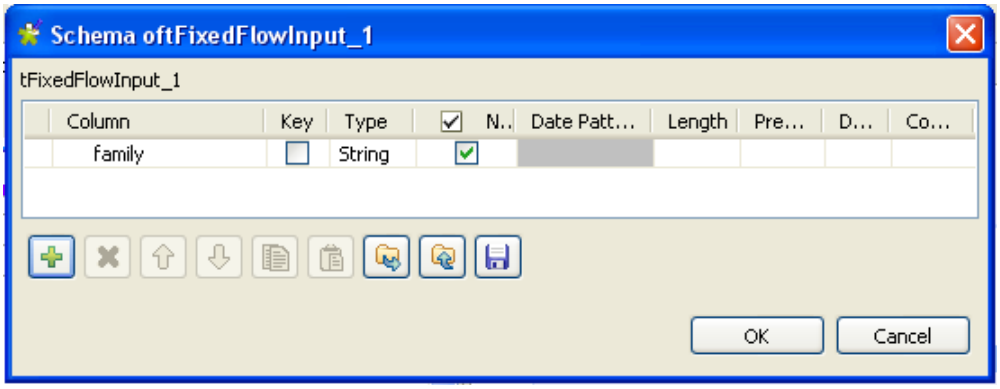
*For the time being, **tWriteXMLField** has some limitations when used with very large datasets. Another scenario is possible to enhance the MDM bulk data load. For further information, see section Enhancing the MDM bulk data load.*

To replicate this scenario, proceed as follows:

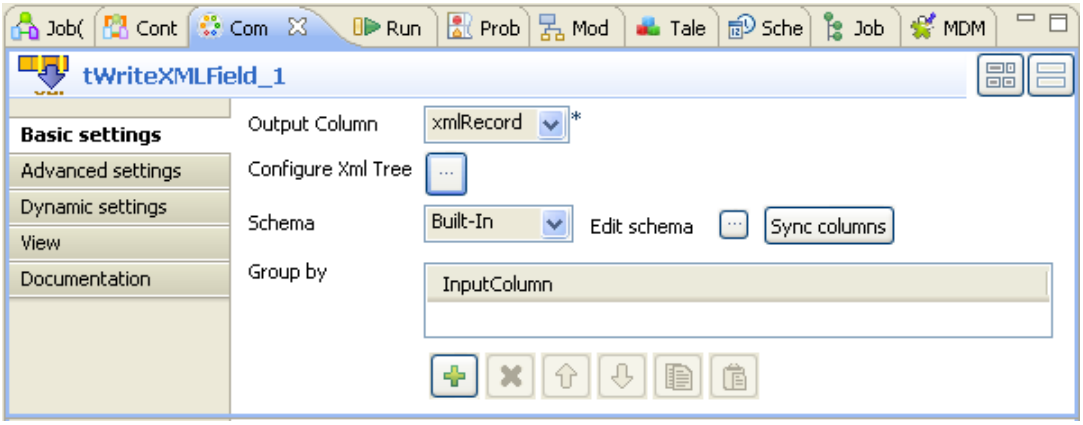
- Drop **tFixedFlowInput**, **tWriteXMLField** and **tMDMBulkLoad** onto the design workspace.
- Right click **tFixedFlowInput** to open its contextual menu.
- Select **Row > Main** to connect **tFixedFlowInput** to the following component using **Main** link.
- Do the same to link the other components.
- Double click **tFixedFlowInput** to open its **Basic settings** view.



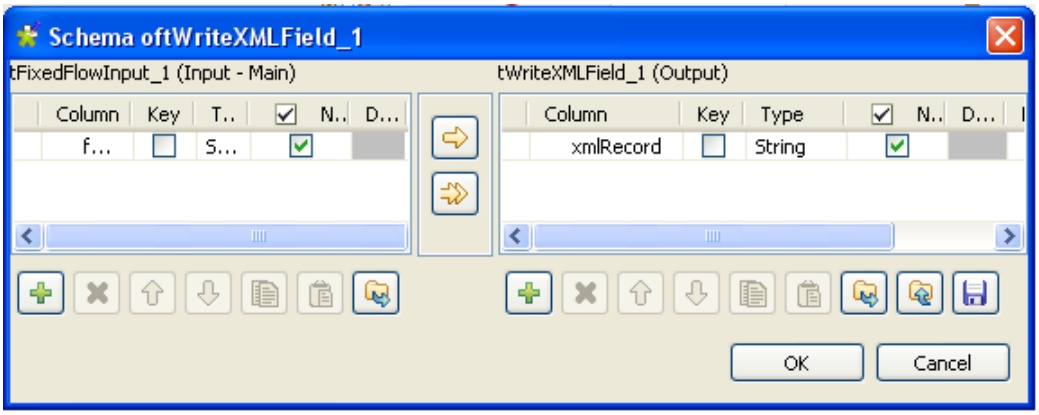
- Click the three-dot button next to **Edit schema** to open the schema editor.



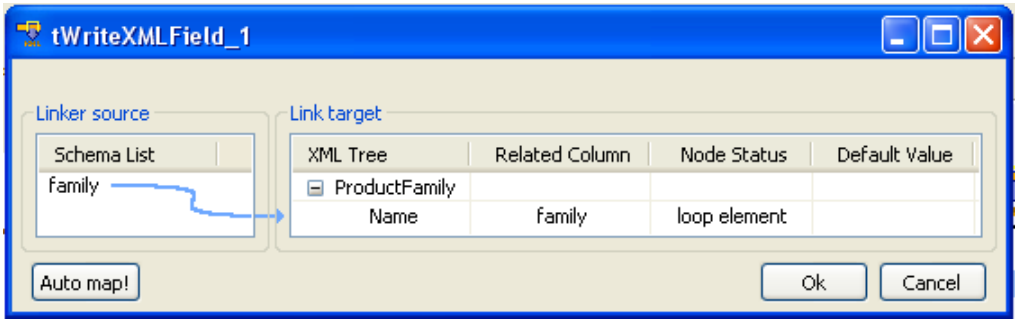
- In the schema editor, click the plus button to add one row.
- In the schema editor, click the new row and type in the new name: *family*.
- Click **OK**.
- In the **Mode** area of the **Basic settings** view, select the **Use inline table** option.
- Under the inline table, click the plus button four times to add four rows in the table.
- In the inline table, click each of the added rows and type in their names between the quotation marks: *Shirts*, *Hats*, *Pets*, *Mugs*.
- Double click **tWriteXMLField** to open its **Basic settings** view.



- Click the three-dot button next to the **Edit schema** field to open the schema editor where you can add a row by clicking the plus button.



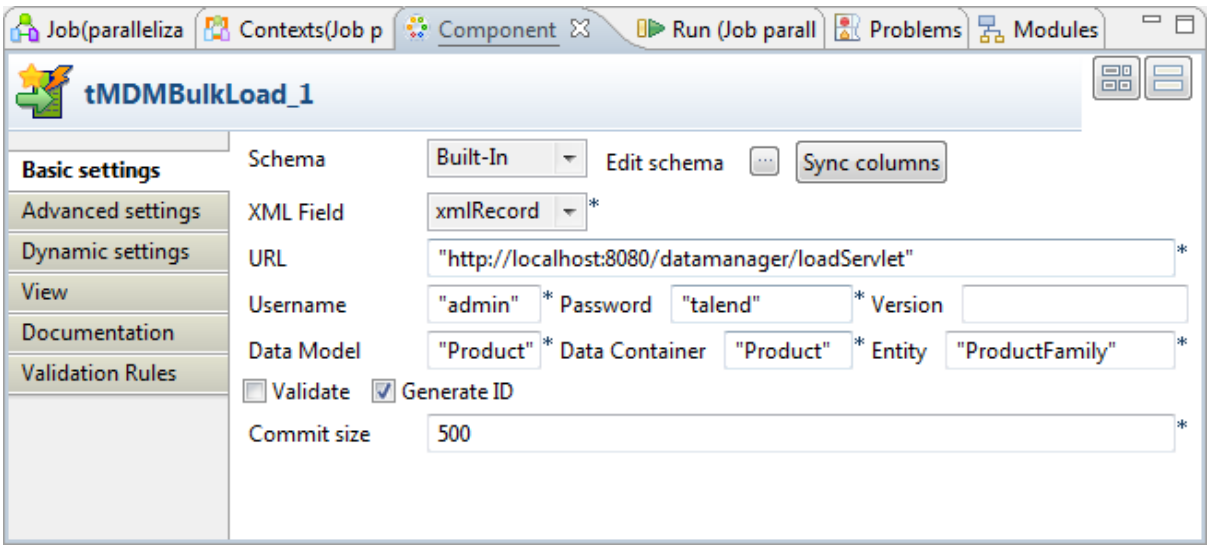
- Click the newly added row to the right view of the schema editor and type in the name of the output column where you want to write the XML content. In this example, type in *xmlRecord*.
- Click **OK** to validate this output schema and close the schema editor.
- In the popped up dialog box, click **OK** to propagate this schema to the following component.
- On the **Basic settings** view, click the three-dot button next to **Configure Xml Tree** to open the interface that helps to create the XML structure.



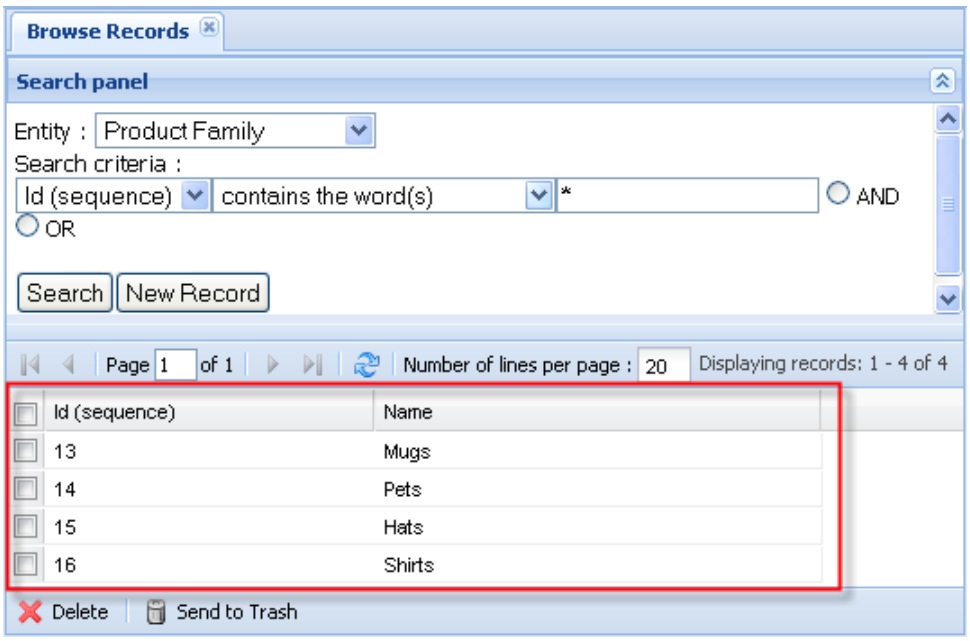
- In the **Link Target** area, click *rootTag* and rename it as *ProductFamily*, which is the name of the business entity used in this scenario.
- In the **Linker source** area, drop *family* to *ProductFamily* in the **Link target** area.

  A dialog box displays asking what type of operation you want to do.

- Select **Create as sub-element of target node** to create a sub-element of the *ProductFamily* node. Then the *family* element appears under the *ProductFamily* node.
- In the **Link target** area, click the **family** node and rename it as **Name**, which is one of the attributes of the **ProductFamily** business entity.
- Right-click the *Name* node and select from the contextual menu **Set As Loop Element**.
- Click **OK** to validate the XML structure you defined.
- Double-click **tMDMBulkLoad** to open its **Basic settings** view.



- In **XML Field**, click this field and select *xmlRecord* from the drop-down list.
- In the **URL** field, enter the bulk loader URL, between quotes: for example, *http://localhost:8080/datamanager/loadServlet*.
- In the **Username** and **Password** fields, enter your login and password to connect to the MDM server.
- In the **Data Model** and the **Data Container** fields, enter the names corresponding to the data model and the data container you need to use. Both are *Product* for this scenario.
- In the **Entity** field, enter the name of the business entity which the records are to be loaded in. In this example, type in *ProductFamily*.
- Select the **Generate ID** check box in order to generate ID values for the records to be loaded.
- In the **Commit size** field, type in the batch size to be written into the MDM hub in bulk mode.
- Press **F6** to run the Job.
- Log into your *Talend MDM Web User Interface* to check the newly added records for the *ProductFamily* business entity.


Browse Records
Search panel
Entity : Product Family
Search criteria :
Id (sequence) contains the word(s) *
AND
OR
Search
New Record
Page 1 of 1
Number of lines per page : 20
Displaying records: 1 - 4 of 4
Id (sequence)
Name
13
Mugs
14
Pets
15
Hats
16
Shirts
Delete
Send to Trash

tMDMClose

tMDMClose properties

Component family	Talend MDM	
Function	**tMDMClose** closes an opened MDM server connection.	
Purpose	This component is used to terminate an opened MDM server connection after the execution of the proceeding subjob.	
Basic settings	*Component List*	Select the **tMDMConnection** component from the list if more than one connection is planned for the current Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component is to be used along with the **tMDMConnection** component.	

Related scenario

For a related use case, see section *Scenario: Deleting master data from an MDM Hub*.

tMDMConnection

tMDMConnection properties

Component family	Talend MDM	
Function	**tMDMConnection** opens an MDM server connection for convenient reuse in the current transaction.	
Purpose	This component is used to open a connection to an MDM server for convenient reuse in the subsequent subjob or subjobs.	
Basic settings	*URL*	Type in the URL required to access the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the Version of master data you want to connect to. Leave this field empty if you want to display the default Version of master data.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	This component is to be used along with the **tMDMSP**, **tMDMViewSearch**, **tMDMInput**, **tMDMDelete**, **tMDMRouteRecord**, **tMDMOutput**, and **tMDMClose** components.	

Related scenario

For a related use case, see section *Scenario: Deleting master data from an MDM Hub*.

tMDMDelete

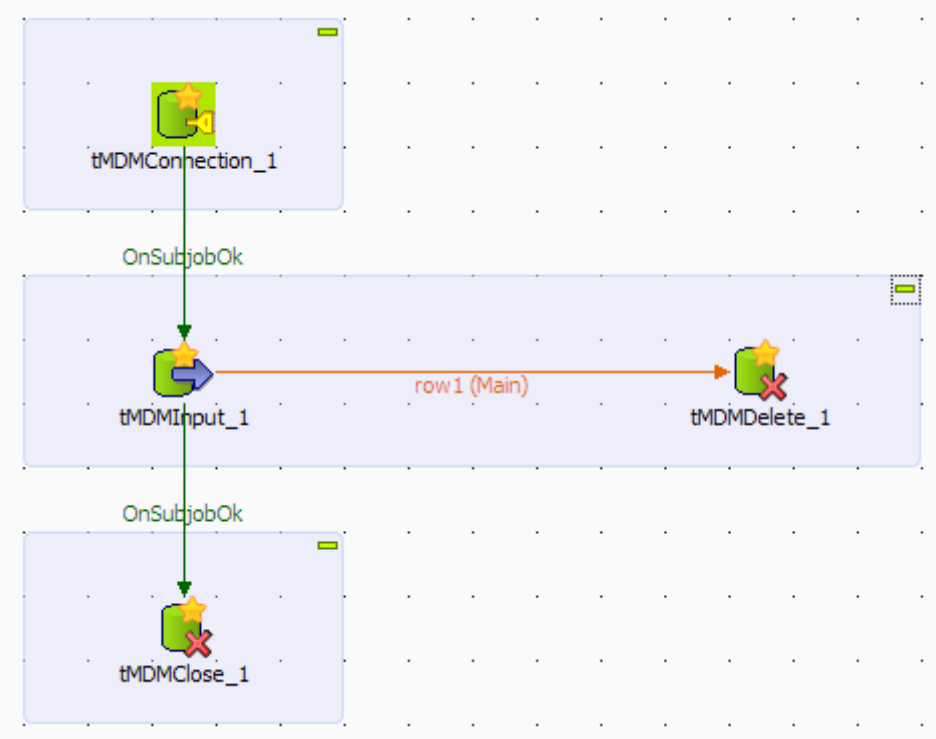
tMDMDelete properties

Component family	Talend MDM	
Function	**tMDMDelete** deletes data records from specific entities in the MDM Hub.	
Purpose	This component deletes master data in an MDM hub.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in. Click **Sync columns** to collect the schema from the previous component.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Use an existing connection	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL required to access the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the Version of master data you want to connect to, for which you have the required user rights. Leave this field empty if you want to display the default Version of master data.
	Entity	Type in the name of the entity that holds the data record(s) you want to delete.
	Data Container	Type in the name of the data container that holds the data record(s) you want to delete.
	Use multiple conditions	Select this check box to filter the master data to be deleted, using certain conditions. **Xpath**: Enter between quotes the path and the XML node to which you want to apply the condition. **Function**: Select the condition to be used from the list. **Value**: Enter between inverted commas the value you want to use. **Predicate**: Select a predicate if you use more than one condition.
	Keys	Specify the field(s) (in sequence order) composing the key when the entity have a multiple key.
	Logical delete	Select this check box to send the master data to the Recycle bin and fill in the **Recycle bin path**. Once in the Recycle bin, the master data can be definitely deleted or restored. If you leave this check box clear, the master data will be permanently deleted.

	Die on error	Select this check box to skip the row in error and complete the process for error-free rows. If needed, you can retrieve the rows in error via a **Row** > **Rejects** link.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to write a file and separate the fields using a specific separator.	

Scenario: Deleting master data from an MDM Hub

This scenario describes a four-component Job that deletes the specified data record from the MDM Hub.



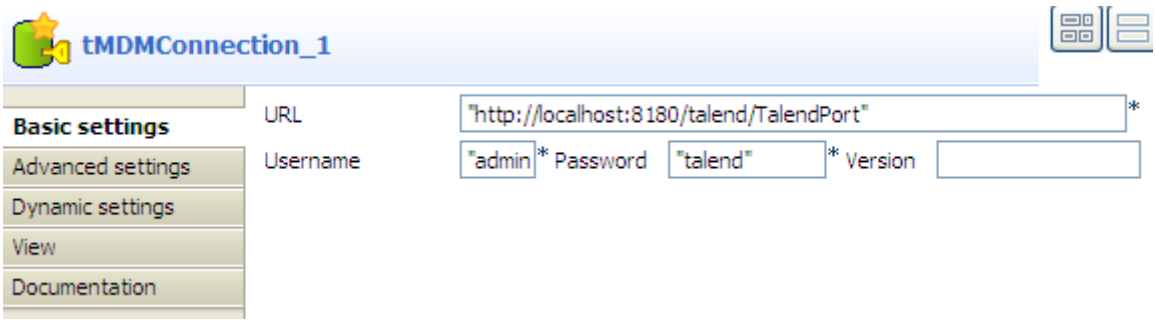
Dropping and linking the components

1. Drop **tMDMConnection**, **tMDMInput**, **tMDMDelete**, and **tMDMClose** of the **Talend MDM** family from the **Palette** onto the design workspace.
2. Connect **tMDMInput** to **tMDMDelete** using a **Row** > **Main** link.
3. Connect **tMDMConnection** to **tMDMInput**, and **tMDMInput** to **tMDMClose** using **Trigger** > **OnSubjobOK** links.

Configuring the MDM server connection

In this scenario, a **tMDMConnection** component is used to open an MDM server connection for convenient reuse in the subsequent subjob that performs the data record deletion task.

1. Double-click **tMDMConnection** to display its **Basic settings** view and define the component properties.



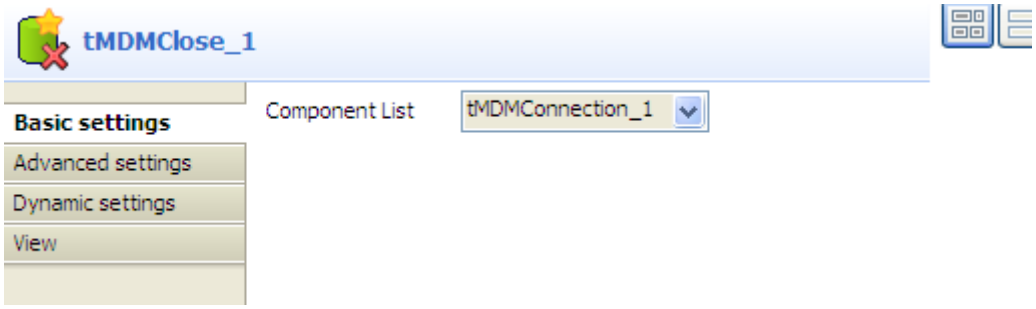
2. In the **URL** field, enter the MDM server URL, between quotation marks: for example, *"http://localhost:8180/talend/TalendPort"*.

3. In the **Username** and **Password** fields, enter your login user name and password to connect to the MDM server.

4. In the **Version** field, enter the name of the master data Version you want to access, between quotation marks. Leave this field empty to access the default master data Version.

5. Double-click **tMDMClose** to display its **Basic settings** view and define the component properties.

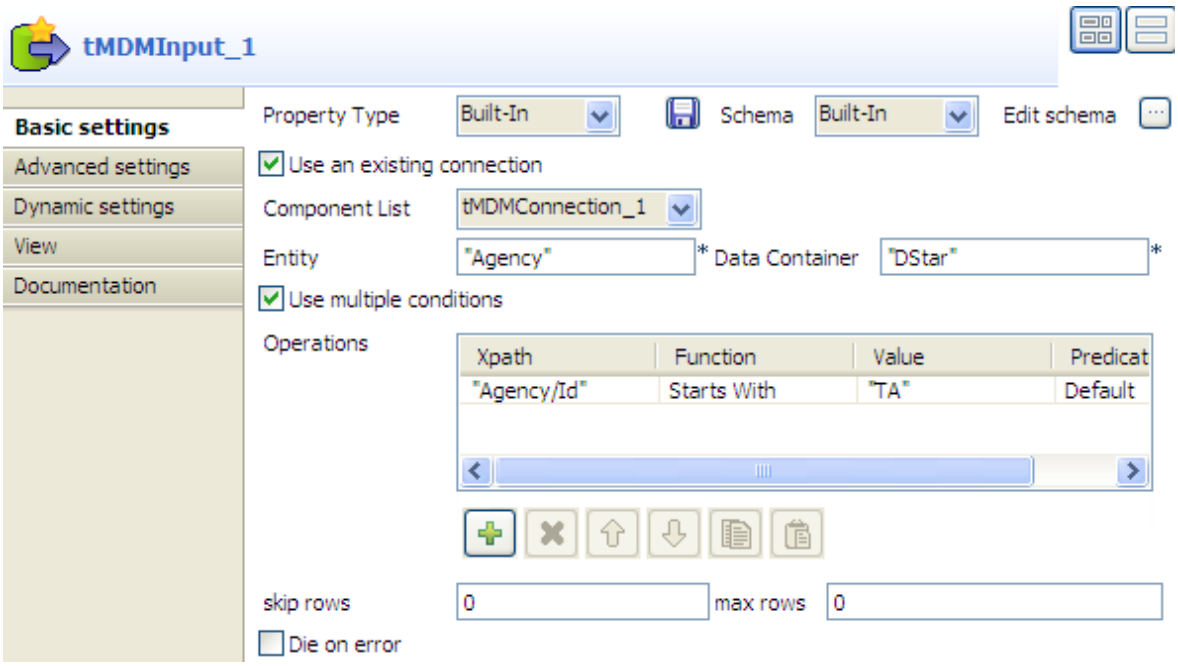
   This component closes the opened MDM server connection after the successful execution of the proceeding subjob.



6. From the **Component List** list, select the component for the server connection you want to close if you have configured more than one MDM server connection. In this use case, there is only one MDM server connection opened, so simply use the default setting.

Configuring data retrieval

1. Double-click **tMDMInput** to display its **Basic settings** view and define the component properties.

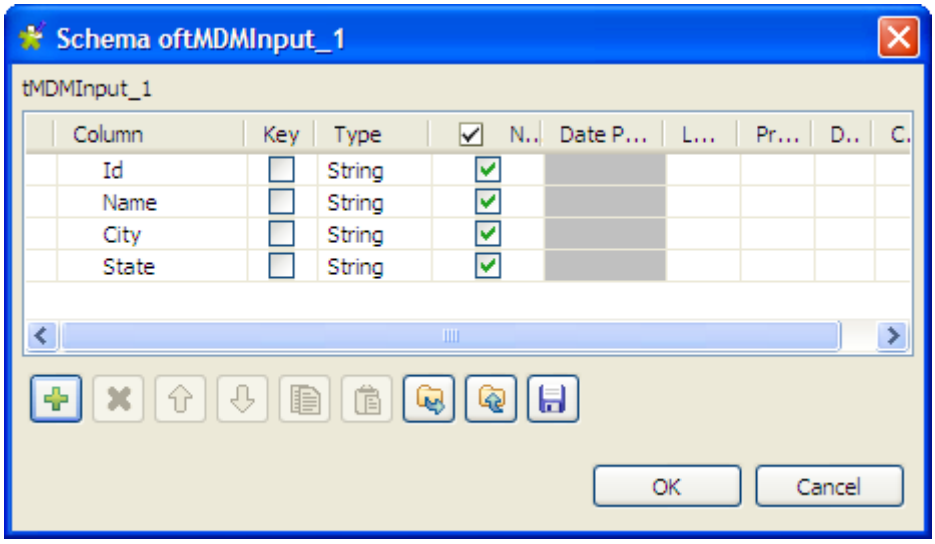


2. From the **Property Type** list, select **Built-in** to complete the fields manually.

   If you have stored the MDM connection information in the repository metadata, select **Repository** from the list and the fields will be completed automatically.

3. From the **Schema** list, select **Built-in** and click **[...]** next to **Edit schema** to open a dialog box.

   Here you can define the structure of the master data you want to read in the MDM hub.



4. The master data is collected in a four column schema of the type **String**: *Id*, *Name*, *City* and *State*. Click **OK** to close the dialog box and proceed to the next step.

5. Select the **Use an existing connection** check box, and from the **Component List** list that appears, select the component you have configured to open your MDM server connection.

   In this scenario, only one MDM server connection exists, so simply use the default selection.

6. In the **Entity** field, enter the name of the business entity that holds the data record(s) you want to read, between quotation marks. Here, we want to access the *Agency* entity.

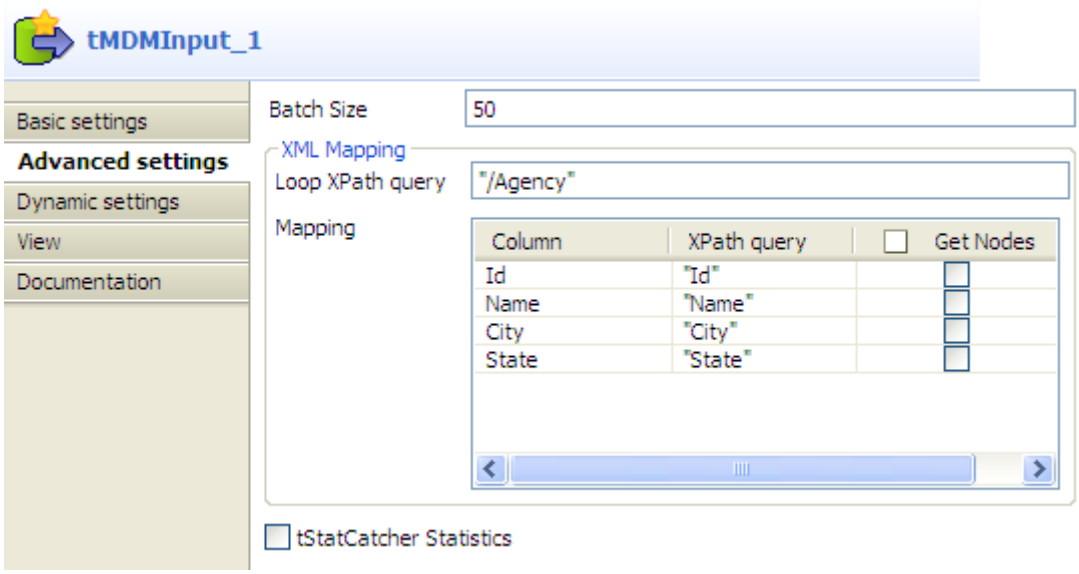
7. In the **Data Container** field, enter the name of the data container that holds the master data you want to read, between quotation marks. In this example, we use the *DStar* container.

   The **Use multiple conditions** check box is selected by default.

8. In the **Operations** table, define the conditions to filter the master data you want to delete as follows:

   - Click the plus button to add a new line.
   - In the **Xpath** column, enter the Xpath and the tag of the XML node on which you want to apply the filter, between quotation marks. In this example, we work with the **Agency** entity, so enter *"Agency/Id"*.
   - In the **Function** column, select the function you want to use. In this scenario, we use the *Starts With* function.
   - In the **Value** column, enter the value of your filter. Here, we want to filter the master data which Id starts with *TA*.

9. In the **Component** view, click **Advanced settings** to set the advanced parameters.

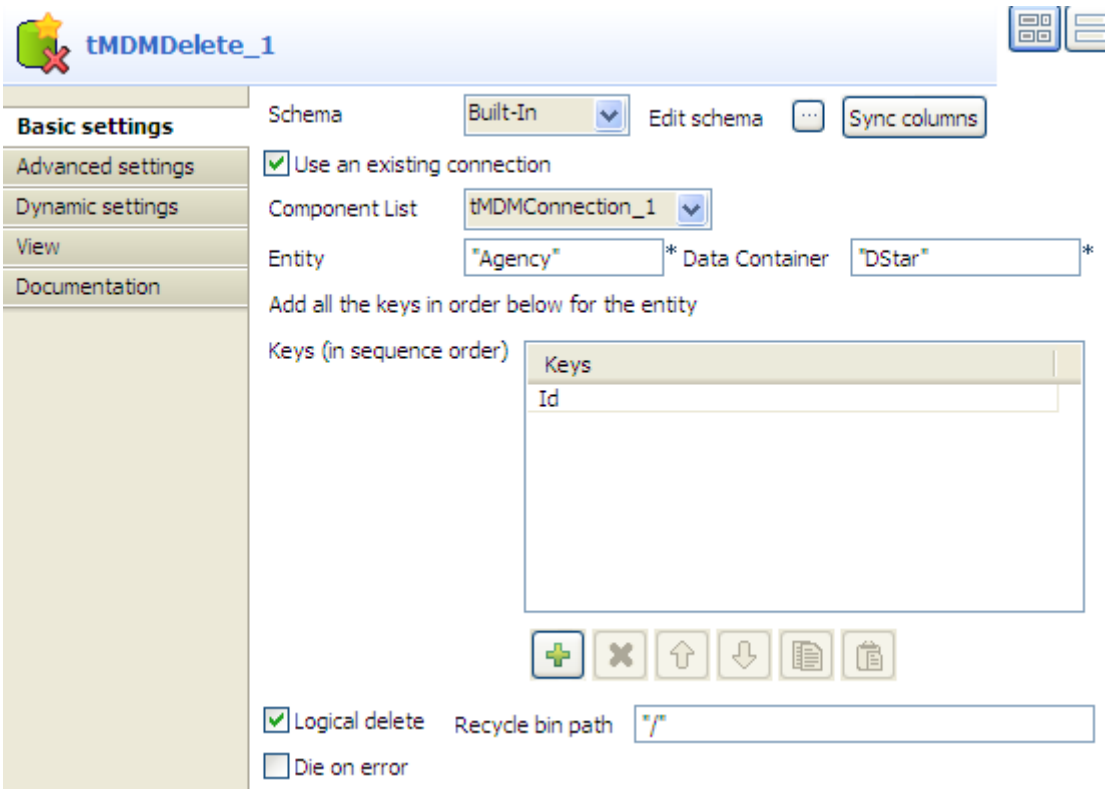


10. In the **Loop XPath query** field, enter the structure and the name of the XML node on which the loop is to be carried out, between quotation marks.

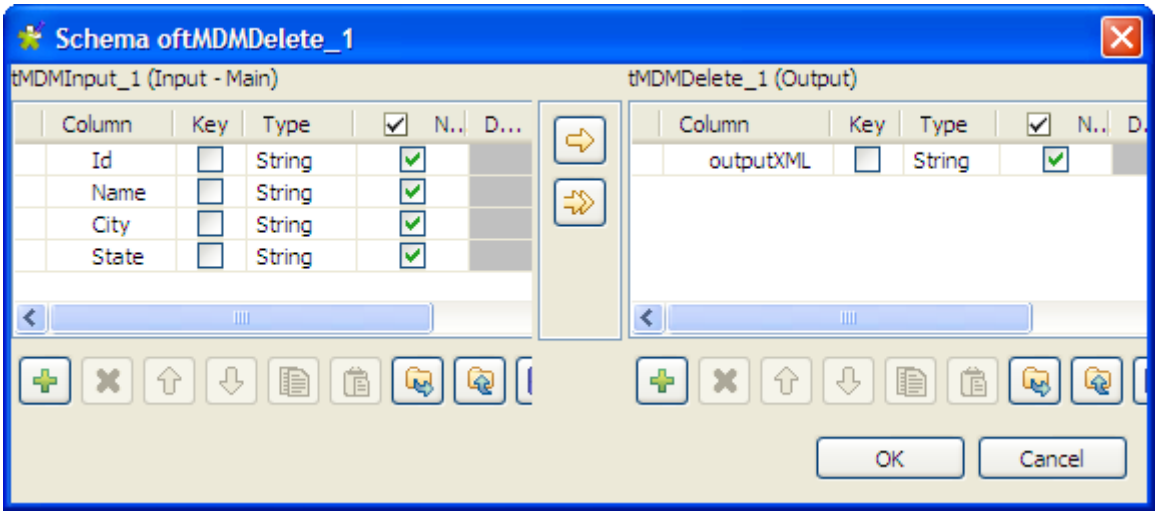
11. In the **Mapping** table and in the **XPath query** column, enter the name of the XML tag in which you want to collect the master data, next to the corresponding output column name, between quotation marks.

Configuring data record deletion

1. In the design workspace, double-click the **tMDMDelete** component to display the **Basic settings** view and set the component properties.



2. From the **Schema** list, select **Built-in** and click the three-dot button next to the **Edit Schema** field to describe the structure of the master data in the MDM hub.



3. Click the plus button to the right to add one column of the type **String**. In this example, name this column *outputXML*. Click **OK** to close the dialog box and proceed to the next step.

4. Select the **Use an existing connection** check box, and from the **Component List** list that appears, select the component you have configured to open your MDM server connection.

   In this scenario, only one MDM server connection exists, so simply use the default selection.

5. In the **Entity** field, enter the name of the business entity that holds the master data you want to delete, the *Agency* entity in this example.

6. In the **Data Container**, enter the name of the data container that holds the data to be deleted, *DStar* in this example.

7. In the **Keys** table, click the plus button to add a new line. In the **Keys** column, select the column that holds the key of the *Agency* entity. Here, the key of the *Agency* entity is set on the *Id* field.

If the entity has multiple keys, add as many line as required for the keys and select them in sequential order.

8. Select the **Logical delete** check box if you do not want to delete the master data permanently. This will send the deleted data to the Recycle bin. Once in the Recycle bin, the master data can be restored or permanently deleted. If you leave this check box clear, the master data will be permanently deleted.

9. Fill in the **Recycle bin path** field. Here, we left the default path but if your recycle bin is in a path different from the default, specify the path.

Saving and executing the Job

1. Press **Ctrl+S** to save your Job to ensure that all the parameters you have configured take effect.

2. Press **F6** to execute your Job.

The master data with the Id starting with “TA” have been deleted and sent to MDM Recycle bin.

tMDMInput

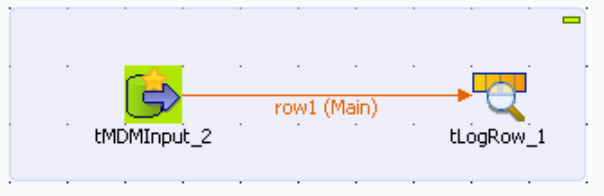
tMDMInput properties

Component family	Talend MDM	
Function	**tMDMInput** reads master data in the MDM Hub.	
Purpose	This component reads master data in an MDM Hub and thus makes it possible to process this data.	
Basic Settings	*Property Type*	Either Built in or Repository.
		Built-in: No property data stored centrally
		Repository: Select the repository file where properties are stored. The fields that follow are completed automatically using the fetched data
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in.
		Built-in: The schema will be created and stored for this component only. Related Topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the repository. You can reuse it in various projects and jobs. Related Topic: see *Talend Open Studio for Big Data User Guide*.
	Use an existing connection	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL to access the MDM server.
	Username and *Password*	Type in user authentication data for the MDM server.
	Version	Type in the name of the master data Version you want to connect to and to which you have access rights. Leave this field empty if you want to display the default Version.
	Entity	Type in the name of the business entity that holds the master data you want to read.
	Data Container	Type in the name of the data container that holds the master data you want to read.
	Use multiple conditions	Select this check box to filter the master data using certain conditions. **Xpath**: Enter between quotes the path and the XML node to which you want to apply the condition. **Function**: Select the condition to be used from the list. **Value**: Enter between inverted commas the value you want to use. Note that if the value contains XML special characters such as /, you must also enter the value in single quotes ('"ABC/XYZ"') or the value will be considered as an XPath. **Predicate**: Select a predicate if you use more than one condition. If you clear this check box, you have the option of selecting particular IDs to be displayed in the **ID value** column of the **IDS** table.

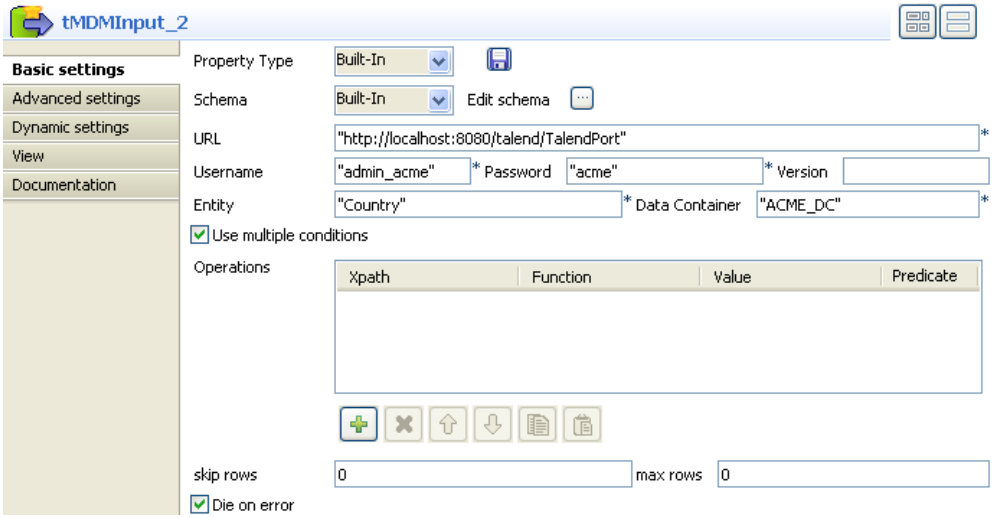
		If you clear the **Use multiple conditions** check box, the **Batch Size** option in the **Advanced Settings** tab will no longer be available
	Skip rows	Enter the number of lines to be ignored.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Die on error	Select this check box to skip the row in error and complete the process for error-free rows. If needed, you can retrieve the rows in error via a **Row** > **Rejects** link.
Advanced settings	*Batch Size*	Number of lines in each processed batch. This option is not displayed if you have cleared the **Use multiple conditions** check box in the **Basic settings** view.
	Loop XPath query	The XML structure node on which the loop is based.
	Mapping	**Column**: reflects the schema as defined in the Edit schema editor. **XPath query**: Type in the name of the fields to extract from the input XML structure. **Get Nodes**: Select this check box to retrieve the Xml node together with the data.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component as a start component. It needs an output flow.	

Scenario: Reading master data in an MDM hub

This scenario describes a two-component Job that reads master data on an MDM server. The master data is fetched and displayed in the log console.

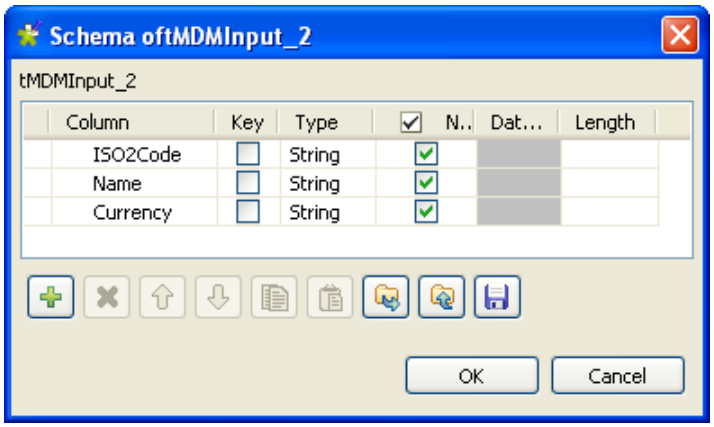


- From the **Palette,** drop **tMDMInput** and **tLogRow** onto the design workspace.
- Connect the two components together using a **Row Main** link.
- Double-click **tMDMInput** to open the **Basic settings** view and define the component properties.



- In the **Property Type** list, select **Built-In** to complete the fields manually. If you have stored the MDM connection information in the repository metadata, select **Repository** from the list and the fields will be completed automatically.

- In the **Schema** list, select **Built-In** and click the three-dot button next to **Edit schema** to open a dialog box. Here you can define the structure of the master data you want to read on the MDM server.



- The master data is collected in a three column schema of the type **String**: *ISO2Code*, *Name* and *Currency*. Click **OK** to close the dialog box and proceed to the next step.

- In the **URL** field, enter between inverted commas the URL of the MDM server.

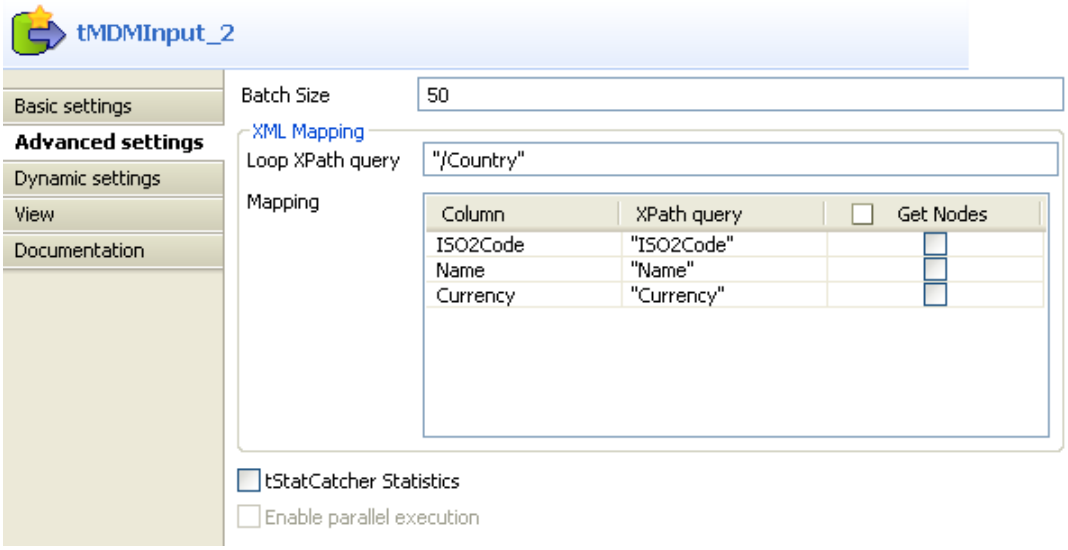
- In the **Username** and **Password** fields, enter your login and password to connect to the MDM server.

- In the **Version** field, enter between inverted commas the name of the master data Version you want to access. Leave this field empty to display the default Version.

- In the **Entity** field, enter between inverted commas the name of the business entity that holds the master data you want to read.

- In the **Data Container** field, enter between inverted commas the name of the data container that holds the master data you want to read.

- In the **Component** view, click **Advanced settings** to set the advanced parameters.



- In the **Loop XPath query** field, enter between inverted commas the structure and the name of the XML node on which the loop is to be carried out.
- In the **Mapping** table and in the **XPath query** column, enter between inverted commas the name of the XML tag in which you want to collect the master data, next to the corresponding output column name.
- In the design workspace, click on the **tLogRow** component to display the **Basic settings** in the **Component** view and set the properties.
- Click on **Edit Schema** and ensure that the schema has been collected from the previous component. If not, click **Sync Columns** to fetch the schema from the previous component.
- Save the Job and press **F6** to run it.

```
Starting job docPars at 16:18 03/11/2009.
[statistics] connecting to socket on port 3916
[statistics] connected
AD|Andorra|[EUR]
AE|United Arab Emirates|[AED]
AF|Afghanistan|[AFN]
AG|Antigua and Barbuda|[XCD]
AI|Anguilla|[XCD]
AL|Albania|[ALL]
AM|Armenia|[AMD]
AN|Netherlands Antilles|[ANG]
AO|Angola|[AOA]
AR|Argentina|[ARS]
AS|American Samoa|[USD]
AT|Austria|[EUR]
AU|Australia|[AUD]
AW|Aruba|[AWG]
```

The list of different countries along with their codes and currencies is displayed on the console of the **Run** view.

tMDMOutput

tMDMOutput properties

Component family	Talend MDM	
Function	**tMDMOutput** writes master data in an MDM Hub.	
Purpose	This component writes master data on the MDM server.	
Basic settings	*Property Type*	Either Built-in or Repository.
		Built-in: No property data stored centrally
		Repository: Select the repository file where the properties are stored. The fields which follow are filled in automatically using the fetched data.
	Input Schema and *Edit schema*	An input schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. Note that if you modify the schema, it automatically becomes built-in. Click **Sync columns** to collect the schema from the previous component.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job designs. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Build the document	Select this check box if you want to build the document from a flat schema If this is the case, double-click the component and map your schema in the dialog box that opens. If the check box is not selected, you must select the column in your schema that contains the document from the **Predefined XML document** list.
	Result of the XML serialization	Lists the name of the XML output column that will hold the XML data.
	Use an existing connection	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL of the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server. This user should have the right role in MDM, i.e. can connect through a Job or any other web service call. For further information, see *Talend Open Studio for MDM* Administrator Guide.
	Version	Type in the name of the master data management Version you want to connect to, for which you have the user rights required. Leave this field empty if you want to display the default perspective.
	Data Model	Type in the name of the data model against which the data to be written is validated.
	Data Container	Type in the name of the data container where you want to write the master data. This data container must already exist.

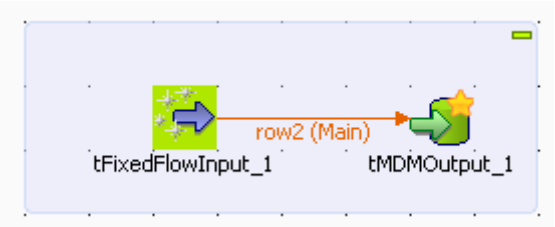
	Return Keys	**Columns corresponding to IDs in order**: in sequential order, set the output columns that will store the return key values (primary keys) of the item(s) that will be created.
	Is Update	Select this check box to update the modified fields. If you leave this check box unchecked, all fields will be replaced by the modified ones.
	Fire Create/Update event	Select this check box to add the actions carried out to a modification report. **Source Name**: Between quotes, enter the name of the application to be used to carry out the modifications. **Enable verification by "before saving" process**: Select this check box to verify the commit that has been just added; prior to saving.

	Use partial update	Select this check box if you need to update multi-occurrences elements (attributes) of an existing item (entity) from the content of a source XML stream.
		
		Once selected, you need to set the parameters presented below:
		
		- **Pivot**: type in the xpath to the multi-occurrences sub-element where data need to be added or replaced in the item of interest.
		
		For example, if you need to add a child sub-element to the below existing item:
		
		`<Person>`
		`  <Id>1</Id> <!-- record key is`
		`  mandatory -->`
		`  <Children>`
		`  <Child>[1234]</Child>`
		`  <!-- FK to a Person Entity -->`
		`  </Children>`
		`</Person>`
		
		then the Xpath you enter in this **Pivot** field must read as the following: `/Person/Children/Child` where the **Overwrite** check box is set to false.
		
		And, if you need to replace a child sub-element in an existing item:
		
		`<Person>`
		`  <Id>1</Id>`
		`  <Addresses>`
		`  <Address>`
		`  <Type>office</Type>`
		`  (...address elements`
		`  are here....)`
		`  </Address>`
		`  <Address>`
		`  <Type>home</Type>`
		`  (...address elements`
		`  are here....)`
		`  </Address>`
		`  <Addresses>`
		`</Person>`
		
		then the Xpath you enter in this **Pivot** field must read as the following: `/Person/Addresses/Adress` where the **Overwrites** check box is set to true, and the **Key** field is set to `/Type` .
		
		In such example, assuming the item in MDM only has an *office* address, the *office* address will be replaced, and the *home* address will be added.
		
		- **Overwrite**: select this check box if you need to replace or update the original sub-elements with the input sub-elements. Leave unselected if you want to add a sub-element.
		
		- **Key**: type in the xpath relative to the pivot that will help matching a sub-element of the source XML with a sub-element of the item. If a key is not

		supplied, all sub-elements of an item with an XPath matching that of the sub-element of the source XML will be replaced. -**Position**: type in a number to indicate the position after which the new elements (those that do not match the *key*) will be added. If you do not provide a value in this field, the new element will be added at the end.
	Die on error	Select this check box to skip the row in error and complete the process for error-free rows. If needed, you can retrieve the rows in error via a **Row** > **Rejects** link.
Advanced settings	*Extended Output*	Select this check box to commit master data in batches. You can specify the number of lines per batch in the **Rows to commit** field.
	Configure Xml Tree	Opens the interface which helps create the XML structure of the master data you want to write.
	Group by	Select the column to be used to regroup the master data.
	Create empty element if needed	This check box is selected by default. If the content of the interface's **Related Column** which enables creation of the XML structure is null, or if no column is associated with the XML node, this option creates an opening and closing tag at the required places.
	Advanced separator (for number)	Select this check box to modify the number of separators used by default. - **Thousands separator:** enter between inverted commas the separator for thousands. - **Decimal separator:** enter between inverted commas the decimal separator.
	Generation mode	Select the appropriate generation mode according to your memory availability. The available modes are: • **Slow and memory-consuming (Dom4j)** This option allows you to use dom4j to process the XML files of high complexity. • **Fast with low memory consumption**
	Encoding	Select the encoding type from the list or else select **Custom** and define it manually. This is an obligatory field for the manipulation of data on the server.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to write a data record and separate the fields using a specific separator.	

Scenario: Writing master data in an MDM hub

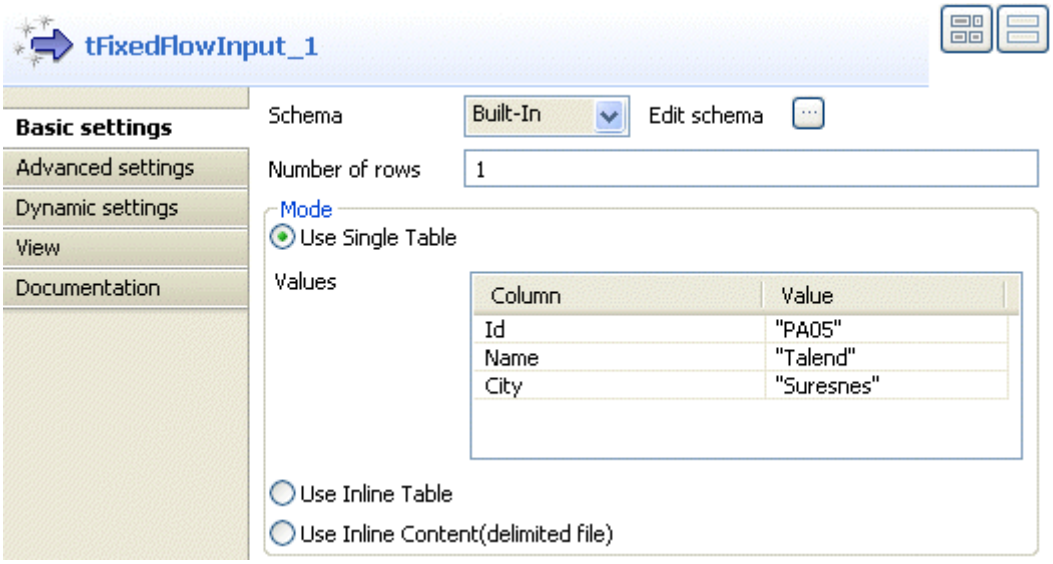
This scenario describes a two-component Job that generates a data record, transforms it into XML and loads it into the defined business entity in the MDM server.



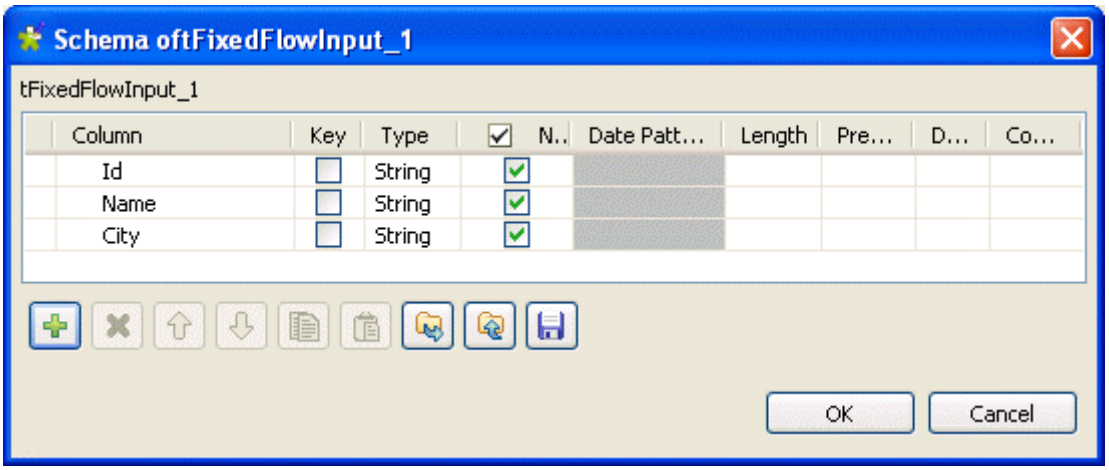
In this example, we want to load a new agency in the *Agency* business entity. This new agency should have an id, a name and a city.

- From the **Palette**, drop **tFixedFlowInput** and **tMDMOutput** onto the design workspace.
- Connect the components using a **Row Main** link.

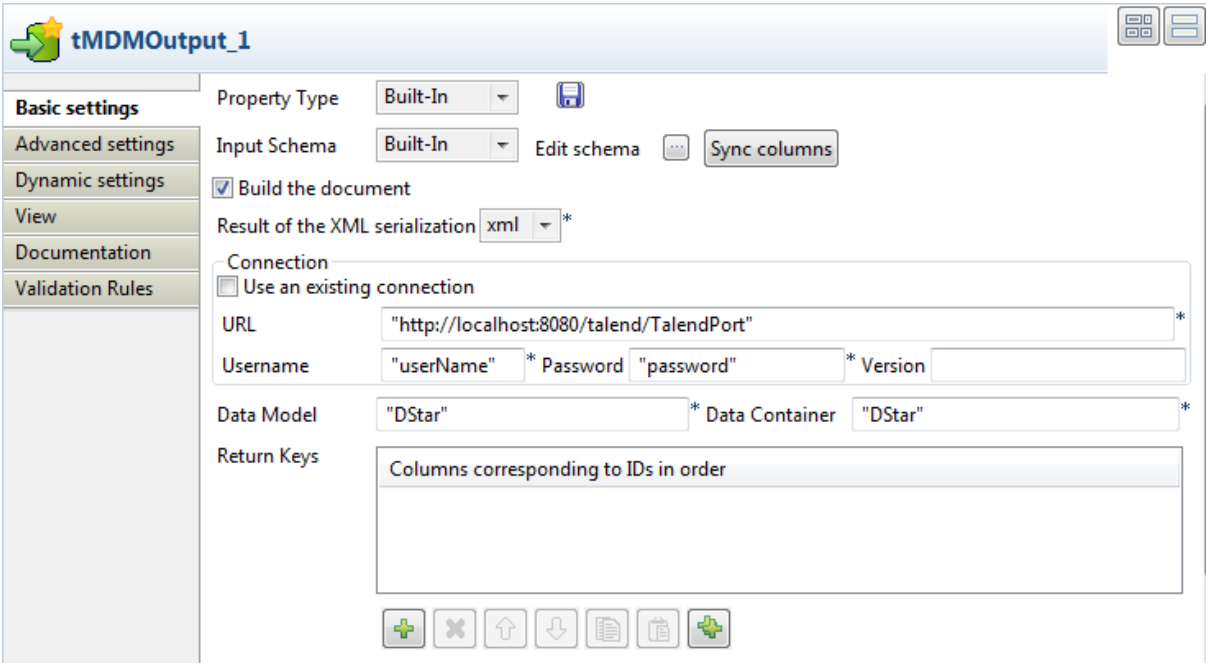
- Double-click **tFixedFlowInput** to view its **Basic settings**, in the **Component** tab and set the component properties.



- In the **Schema** list, select **Built-In** and click the three-dot button next to **Edit schema** to open a dialog box in which you can define the structure of the master data you want to write on the MDM server.



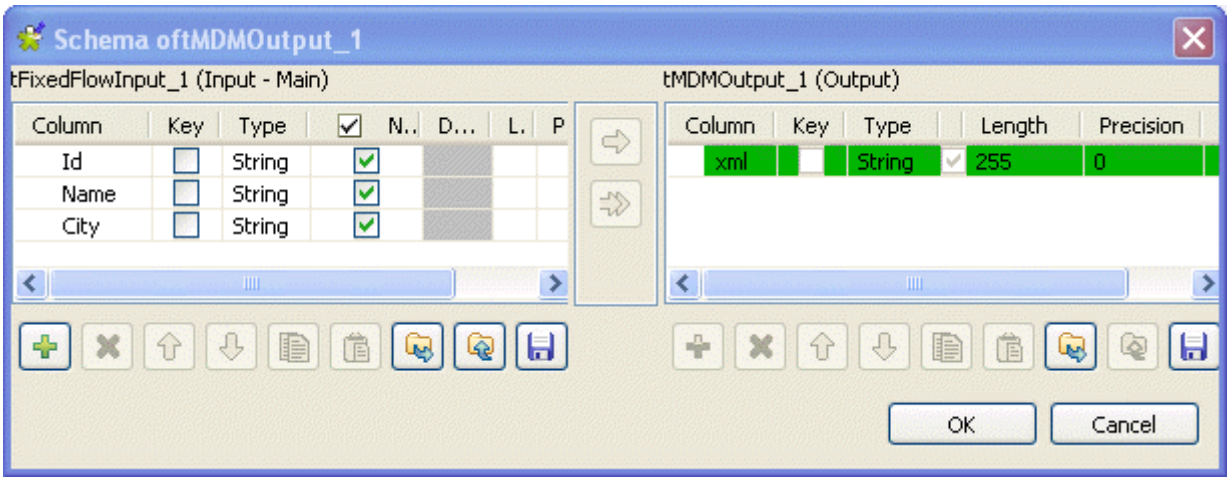
- Click the plus button and add three columns of the type **String**. Name the columns: *Id*, *Name* and *City*.
- Click **OK** to validate your changes and proceed to the next step.
- In the **Number of rows** field, enter the number of rows you want to generate.
- In the **Mode** area, select the **Use Single Table** option to generate just one table.
- In the **Value** fields, enter between inverted commas the values which correspond to each of the schema columns.
- In the design workspace, click **tMDMOutput** to open its **Basic settings** view and set the component properties.



- In the **Property Type** list, select **Built-In** and complete the fields manually.

  If you have saved the MDM connection information under **Metadata** in the repository, select **Repository** from the list and the fields which follow will be completed automatically.

- In the **Schema** list, select **Built-In** and, if required, click on the three dot button next to the **Edit Schema** field to see the structure of the master data you want to load on the MDM server.



The **tMDMOutput** component basically generates an XML document, writes it in an output field, and then sends it to the MDM server, so the output schema always has a read-only *xml* column.

- Click **OK** to proceed to the next step.

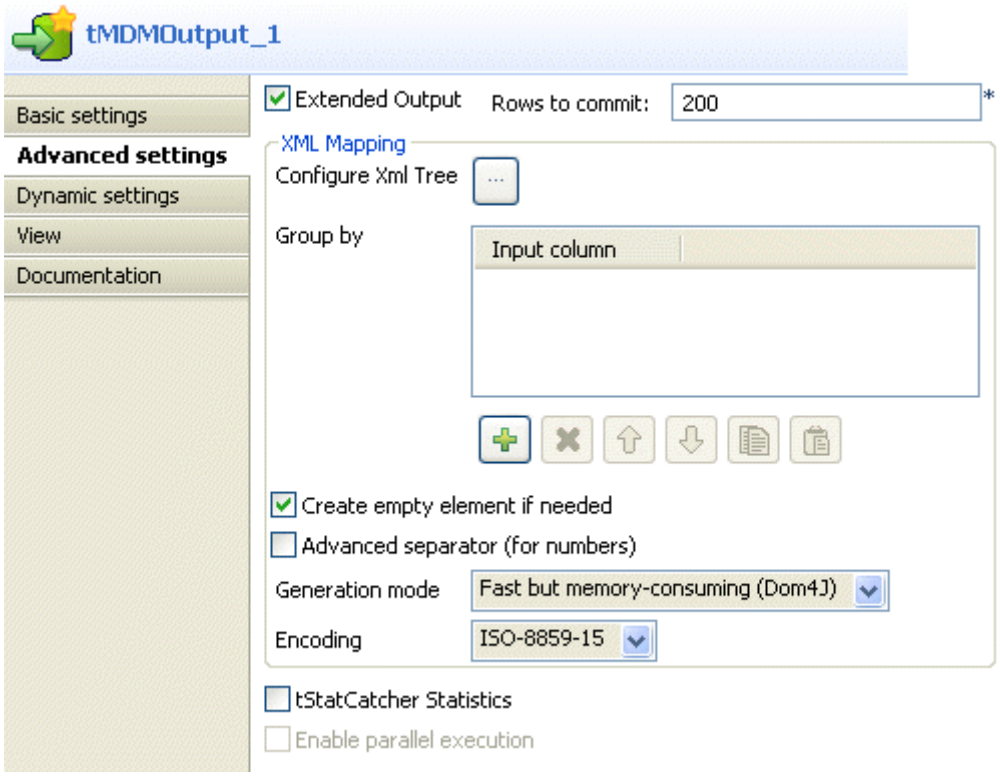
  The **Result of the XML serialization** list in the **Basic settings** view is automatically filled in with the output *xml* column.

- In the **URL** field, enter the URL of the MDM server.

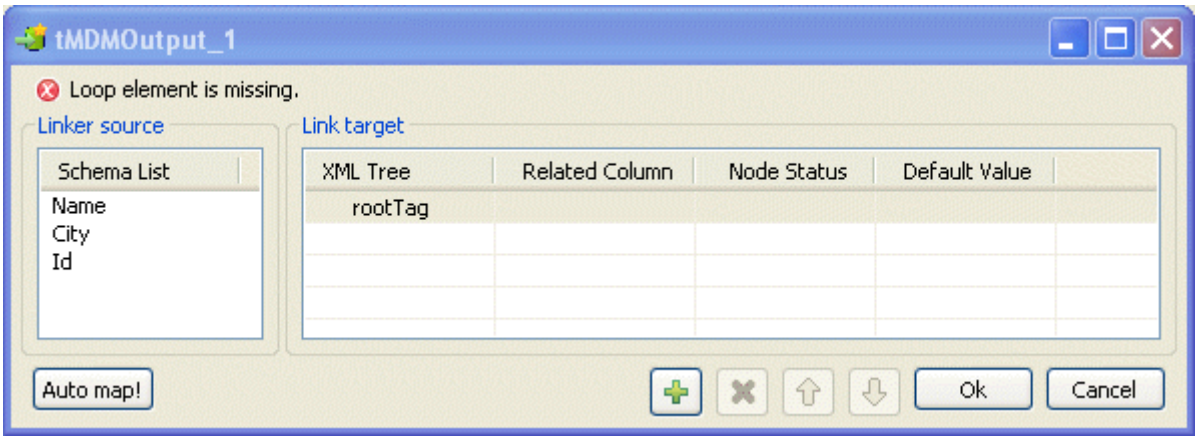
- In the **Username** and **Password** fields, enter the authentication information required to connect to the MDM server.

- In the **Version** field, enter between inverted commas the name of the master data Version you want to access, if more than one exists on the server. Leave the field blank to access the default Version.

- In the **Data Model** field, enter between inverted commas the name of the data model against which you want to validate the master data you want to write.
- In the **Data Container**, enter between inverted commas the name of the data container into which you want to write the master data.
- In the **Component** view, click **Advanced settings** to set the advanced parameters for the **tMDMOutput** component.

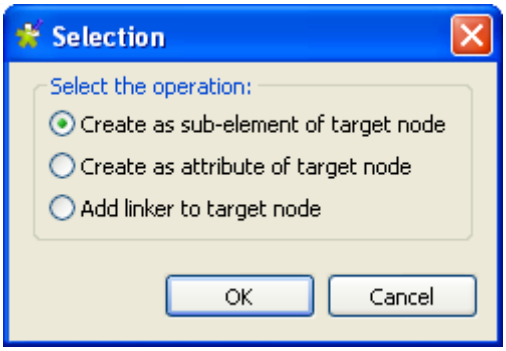


- Select the **Extended Output** check box if you want to commit master data in batches. You can specify the number of lines per batch in the **Rows to commit** field.
- Click the three-dot button next to **Configure Xml Tree** to open the **tMDMOutput** editor.

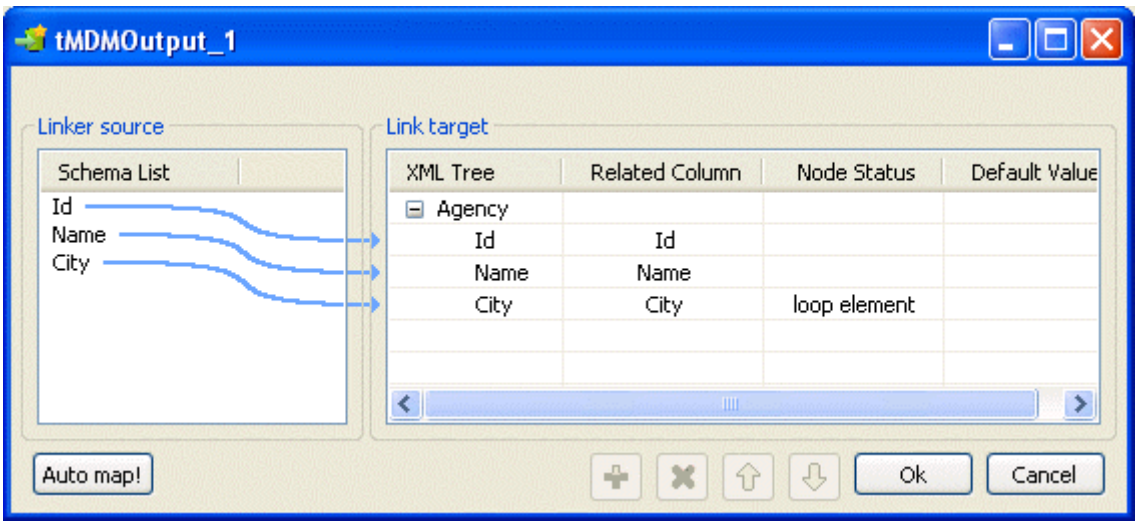


- In the **Link target** area to the right, click in the **Xml Tree** field and then replace **rootTag** with the name of the business entity in which you want to insert the data record, *Agency* in this example.
- In the **Linker source** area, select your three schema columns and drop them on the **Agency** node.

The **[Selection]** dialog box displays.

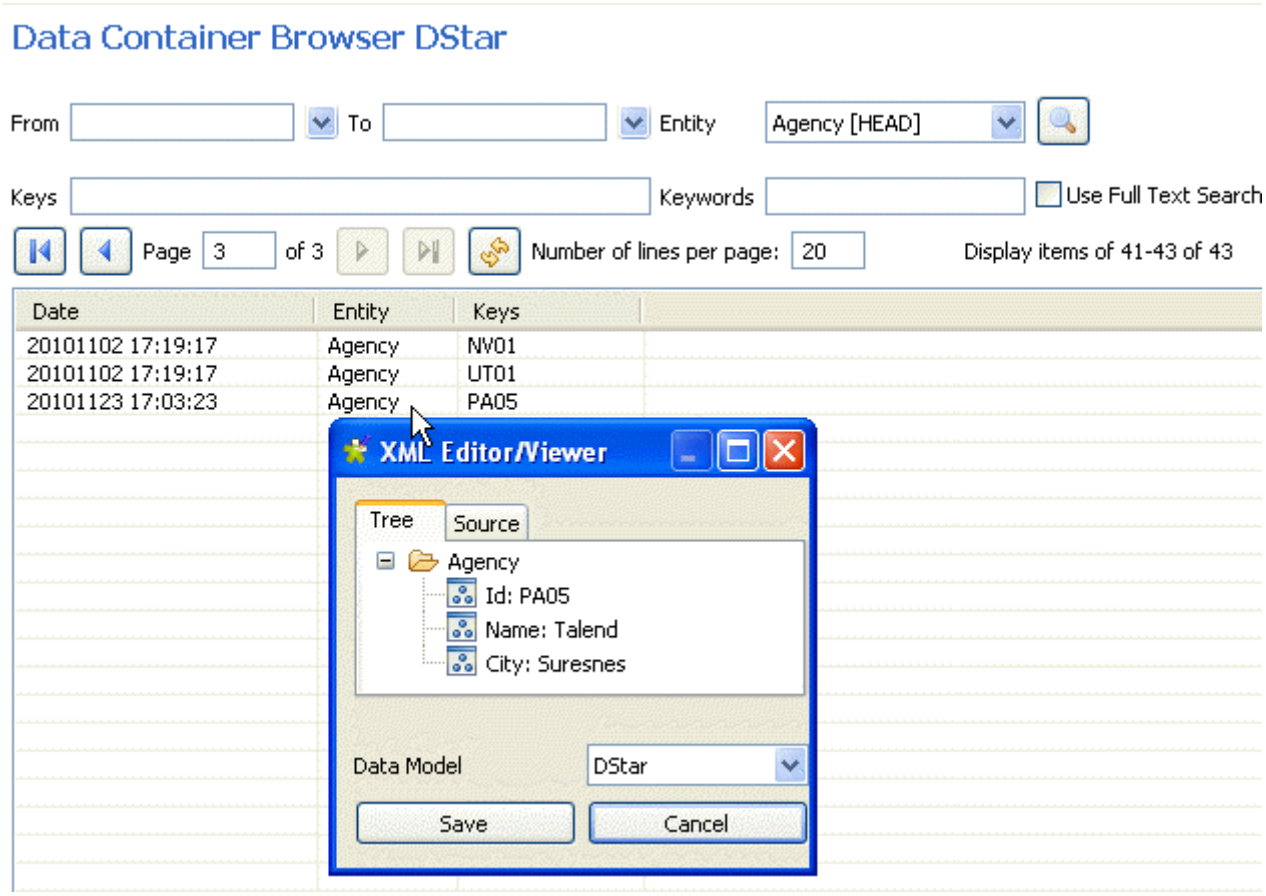


- Select the **Create as sub-element of target node** option so that the three columns are linked to the three XML sub-elements of the **Agency** node and then click **OK** to close the dialog box.



- Right-click the element in the **Link Target** area you want to set as a loop element and select **Set as Loop Element** from the contextual menu. In this example, we want **City** to be the iterating object.
- Click **OK** to validate your changes and close the dialog box.
- Save your Job and press **F6** to run it.

The new data record is inserted in the *Agency* business entity in the *DStar* data container on the MDM server. This data records holds, as you defined in the schema, the agency id, the agency name and the agency city.


Data Container Browser DStar
From
To
Entity
Agency [HEAD]
Keys
Keywords
Use Full Text Search
Page 3 of 3
Number of lines per page: 20
Display items of 41-43 of 43
Date
Entity
Keys
20101102 17:19:17 Agency NV01
20101102 17:19:17 Agency UT01
20101123 17:03:23 Agency PA05
XML Editor/Viewer
Tree
Source
Agency
Id: PA05
Name: Talend
City: Suresnes
Data Model
DStar
Save
Cancel

tMDMReceive

tMDMReceive properties

Component family	Talend MDM	
Function	**tMDMReceive** receives an MDM record in XML from MDM triggers or MDM processes.	
Purpose	This component decodes a context parameter holding MDM XML data and transforms it into a flat schema.	
Basic Settings	*Property Type*	**Either Built in or Repository**.
		Built-in: No property data stored centrally
		Repository: Select the repository file where properties are stored. The fields that follow are completed automatically using the fetched data
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in.
		Built-in: The schema will be created and stored for this component only. Related Topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the repository. You can reuse it in various projects and jobs. Related Topic: see *Talend Open Studio for Big Data* **User Guide**.
	XML Record	Enter the context parameter allowing to retrieve the last changes made to the MDM server. For more information about creating and using a context parameter, see *Talend Open Studio for Big Data* **User Guide**.
	XPath Prefix	If required, select from the list the looping xpath expression which is a concatenation of the prefix + looping xpath. **/item**: select this xpath prefix when the component receives the record from a process because processes encapsulate the record within an item element only. **/exchange/item**: select this xpath prefix when the component receives the record from a trigger because triggers encapsulate the record within an item element which is within an exchange element.
	Loop XPath query	Set the XML structure node on which the loop is based.
	Mapping	**Column**: reflects the schema as defined in the **Edit schema** editor. **XPath query**: Type in the name of the fields to extract from the input XML structure. **Get Nodes**: Select this check box to retrieve the XML node together with the data.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.

Usage	Use this component as a start component. It needs an output flow.

Related scenario

No scenario is available for this component yet.

tMDMRouteRecord

tMDMRouteRecord properties

Component family	Talend MDM	
Function	**tMDMRouteRecord** submits the primary key of a record stored in your MDM Hub to **Event Manager** in order for **Event Manager** to trigger the due process(es) against some specific conditions that you can define in the process or trigger pages of the MDM Studio. For more information on **Event Manager** and on a MDM process, see *Talend Open Studio for MDM* **Administrator Guide**.	
Purpose	This component helps **Event Manager** identify the changes which you have made on your data so that correlative actions can be triggered.	
Basic Settings	*Use an existing connection*	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL of the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the master data management Version you want to connect to, for which you have the user rights required. Leave this field empty if you want to display the default perspective.
	Data Container	Type in the name of the data container that holds the record you want **Event Manager** to read.
	Entity Name	Type in the name of the business entity that holds the record you want **Event Manager** to read.
	IDS	Specify the primary key(s) of the record(s) you want **Event Manager** to read.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Global Variables		**Number of Lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data* **User Guide**.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Component Ok; On Component Error, On Subjob Ok, On Subjob Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if, On Component Ok, On Component Error, On Subjob Ok, On Subjob Error For further information regarding connections, see *Talend Open Studio for Big Data* **User Guide**.

Usage	Use this component as a start component. It needs an output flow.

Scenario: Routing a record to Event Manager

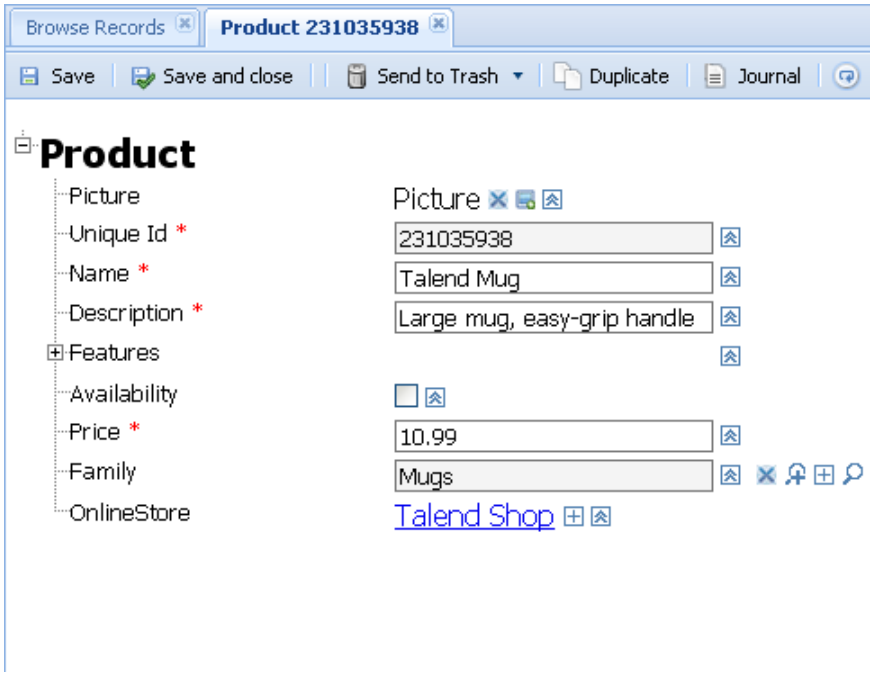
In this scenario, the **tMDMRouteRecord** component is used to submit the primary key of a record noting an update to **Event Manager** in order for this element to trigger a process that informs the user of this update.

***Talend MDM** is case-sensitive, so respect the differences of uppercase and lowercase when realizing the scenario.*

Scenario prerequisites

The following prerequisites must be met in order to replicate this scenario:

- A data container stores several records using a specific model. In this scenario, the container is named *Product*, and a record in the container is entered against the model named *Product*:



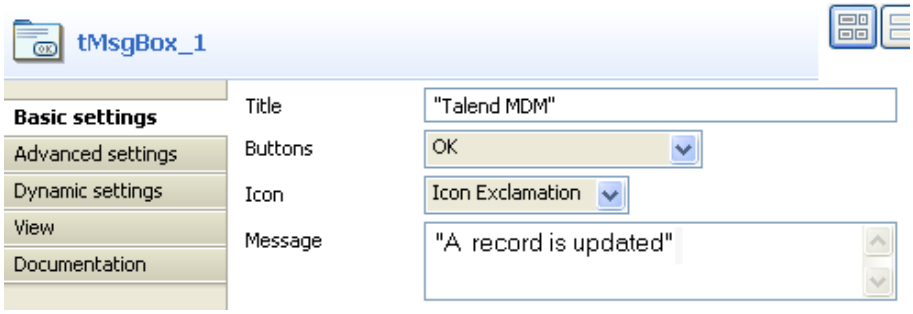
This figure shows one of the stored product records with all of its viewable attributes.

For further information about how to create a data container, a data model, see your *Talend Open Studio for MDM* Administrator Guide.

For further information about how to create a record and access its viewable attributes, see *Talend MDM Web User Interface* User Guide.

- A Job used to inform the user of the update and already deployed on the MDM server. In this scenario, the Job is called *message*, using only the **tMsgBox** component.

- Double-click the component to display and configure its **Basic settings** :



- In the **Title** field, type in "*Talend MDM*".
- In the **Message** field to be popped up, type in "*A record is updated*".

For further information about the **tMsgBox** component, see section *tMsgBox*.

For further information about how to deploy a Job onto the **MDM server**, see *Talend Open Studio for MDM* Administrator Guide.

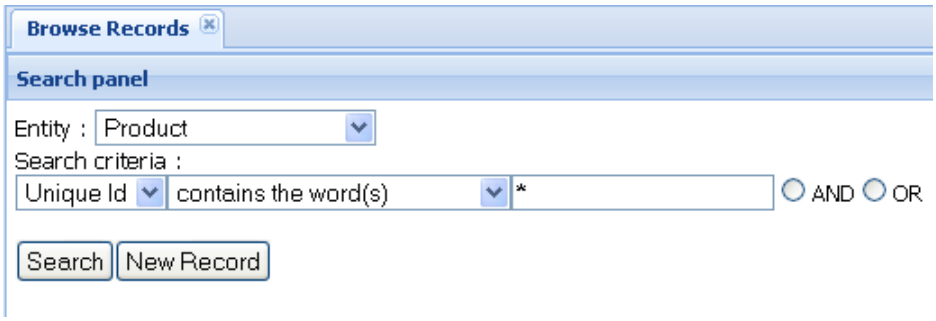
Routing a record to trigger the corresponding process

This section shows you how to replicate the whole scenario using **tMDMRouteRecord** to trigger a process.

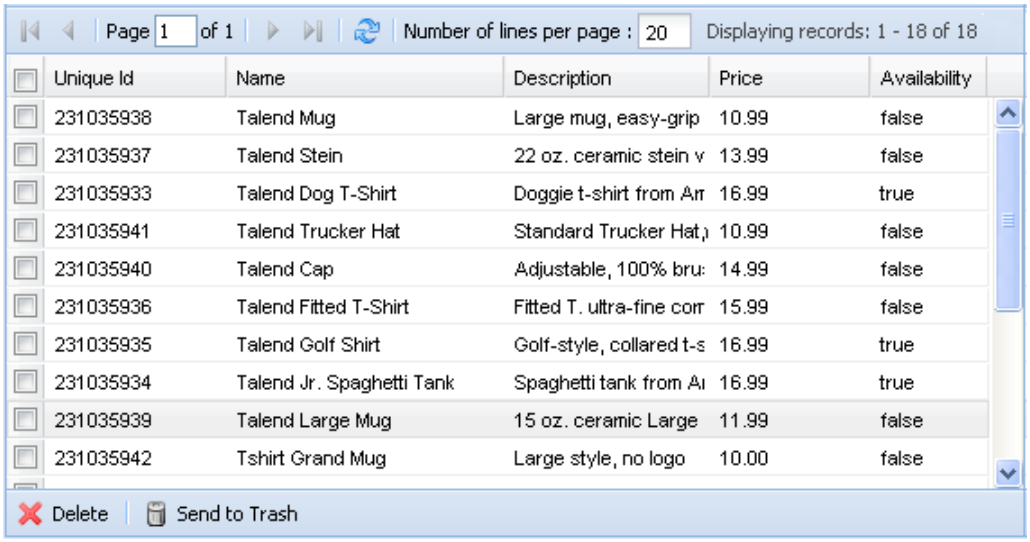
- Log onto your **Talend MDM Web UI** and click **Browse Records**.

  For further details about how to log onto the **Talend MDM Web UI** and open the **Browse Records** view, see *Talend MDM Web User Interface* User Guide.

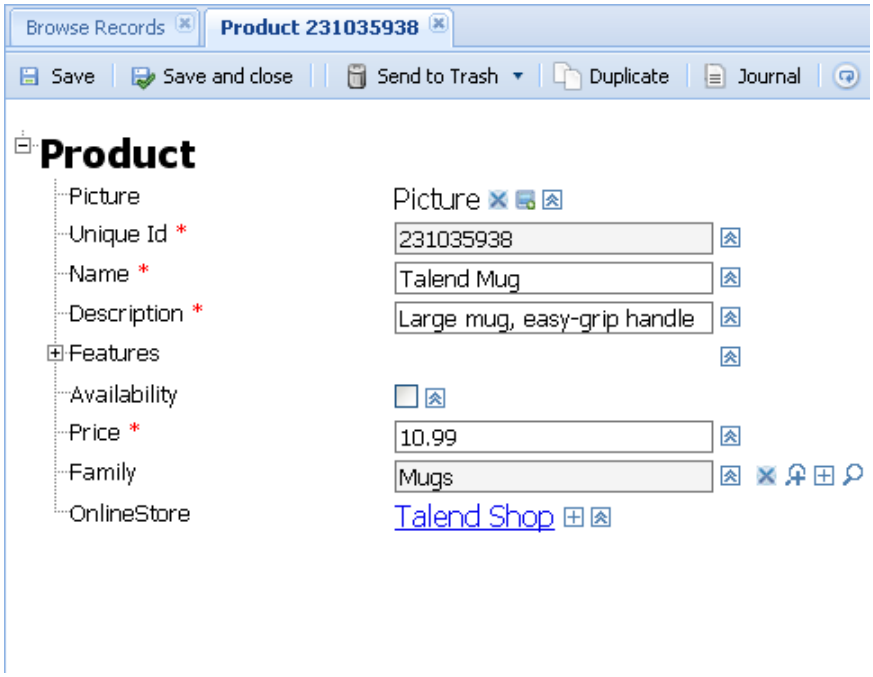
- In the upper right corner of the web page, click on the button to show the **Actions** panel.
- On the **Actions** panel on the right, select the required data container and data model in which is the record to be updated. In this scenario, the data container and the data model are both *Product*.
- Click **Save** to save the selected data container and data model.
- In the **Browse Records** view, select the entity of your interest. In this example, it is *Product*.



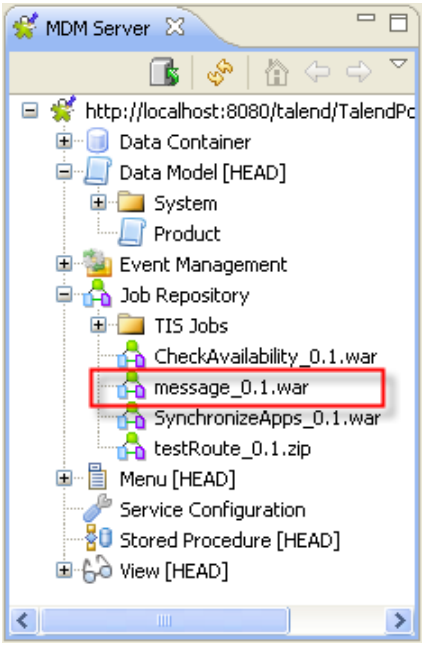
- Click **Search** to open the record list on the lower part of the Web page.



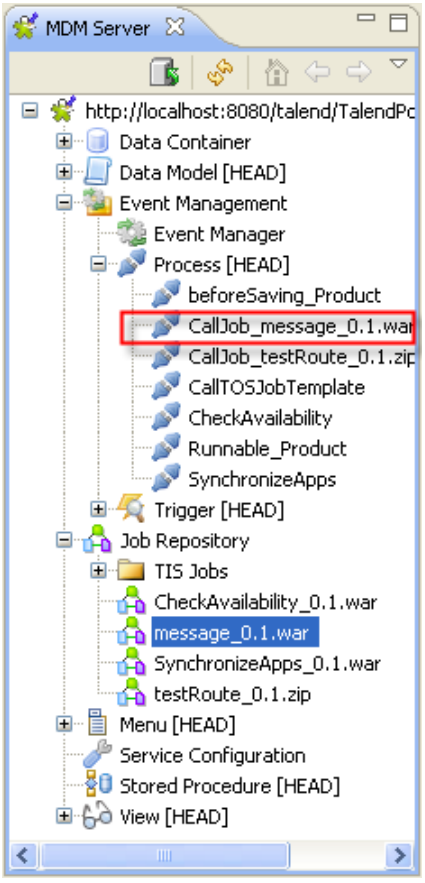
- Double-click one of the product records to display its viewable attributes in a new view dedicated to this product. For example, open the product *Talend Mug* with unique Id *231035938*.



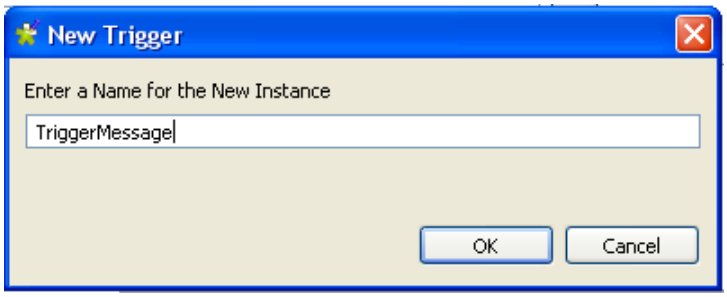
- In this view, modify one of the attribute values. You can, for example, update this product and make it available by selecting the **Availability** check box.
- Click **Save** to validate this update.
- Open your **Talend MDM studio** and access the MDM Hub. For further information about how to launch the **Talend MDM studio** and connect it to the MDM hub, see *Talend Open Studio for MDM* Administrator Guide.



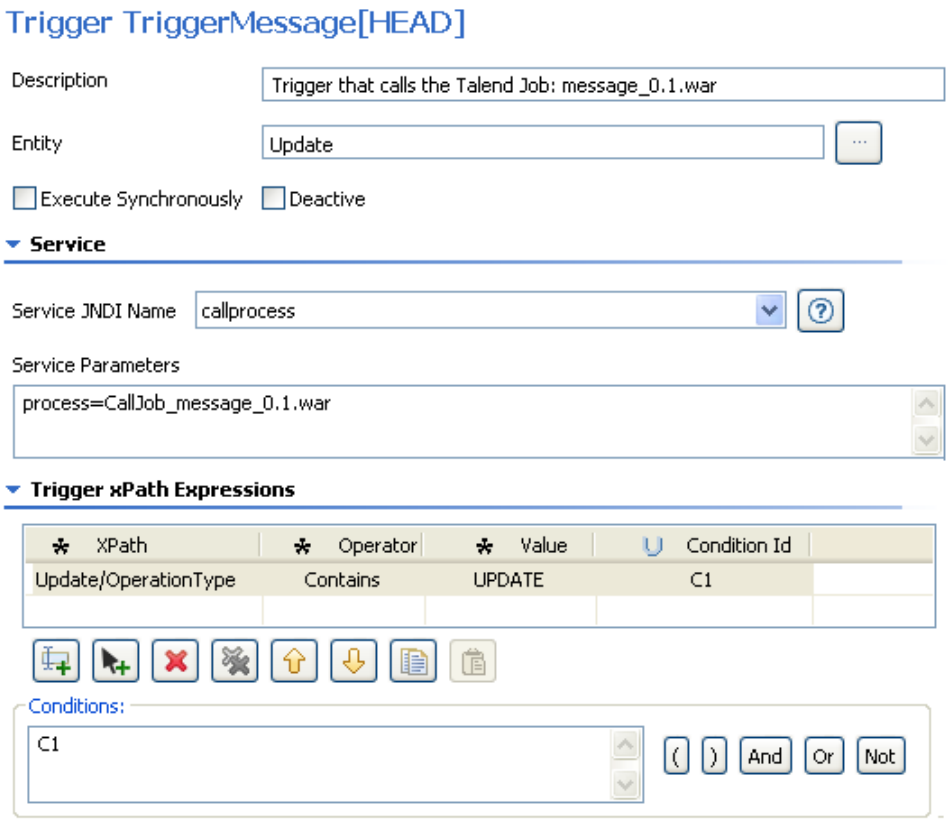
- Under the **Job Repository** node of the **MDM Server** tree view, right click the *message* Job.
- In the contextual menu, select **Generate Talend Job Caller Process**.The process used to call this Job is generated and displays in the directory **Event Management** > **Process**.



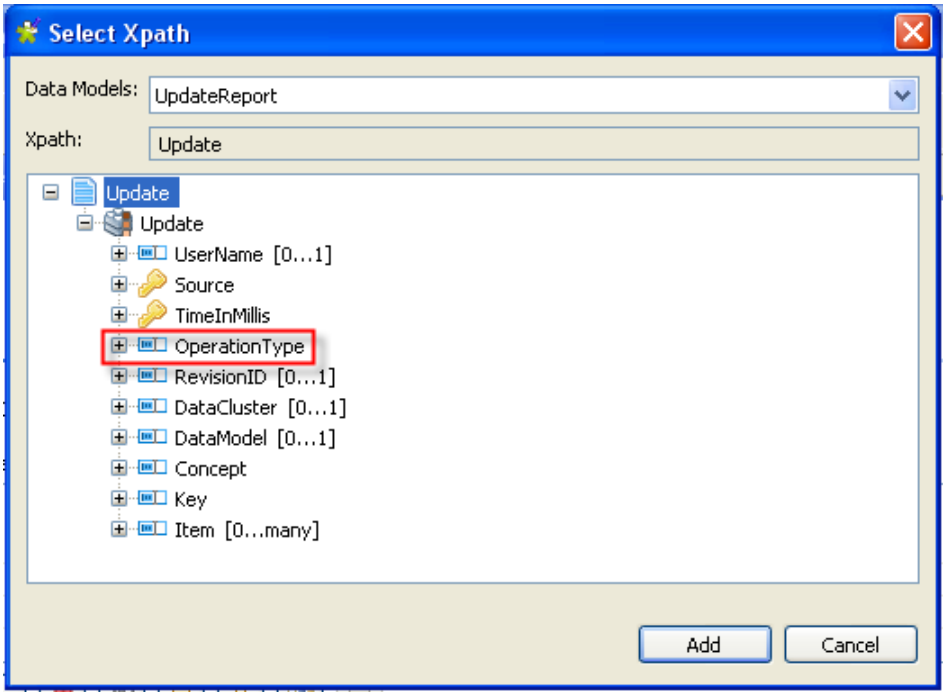
- Under the **Event Management** node, right click **Trigger**.
- In the contextual menu, select **New**.
- In the pop-up **New Trigger** wizard, name the trigger as, for example, *TriggerMessage*.



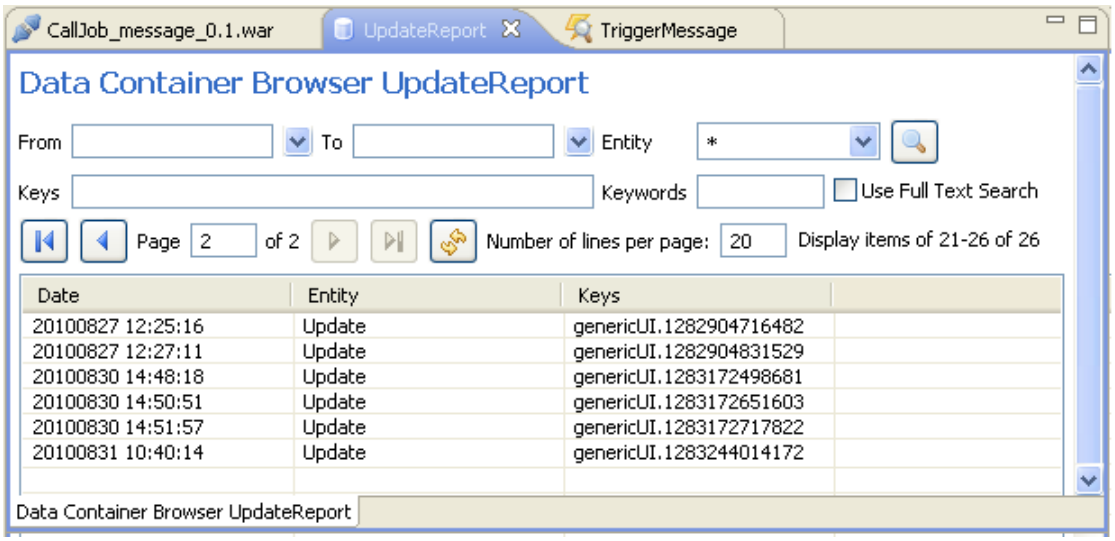
- Click **OK** to open the new trigger's view in the workspace of your studio.
- In the trigger's view, configure the trigger to make it launch the process that calls the *message* Job once an update is done.



- In the **Description** field, enter, for example, *Trigger that calls the Talend Job: message_0.1.war* to describe the trigger being created.
- In the **Entity** field, select or type in the business entity you want to trigger the process on. In this example, it is exactly *Update*.
- In the **Service JNDI Name** field, select **callprocess** from the drop-down list.
- In the Service Parameters field, complete the parameter definition by giving the value: *CallJob_message_0.1.war*. This value is the name of the process to be called that you can find in the directory **Event Management > Process** in the **MDM server** tree view.
- In the **Trigger xPath Expressions** area, click the button under the table to add a new XPath line.
- In the newly added line, click the three-dot button to open a dialog box where you can select the entity or element on which you want to define conditions. In this example, it is *Update/OperationType*.

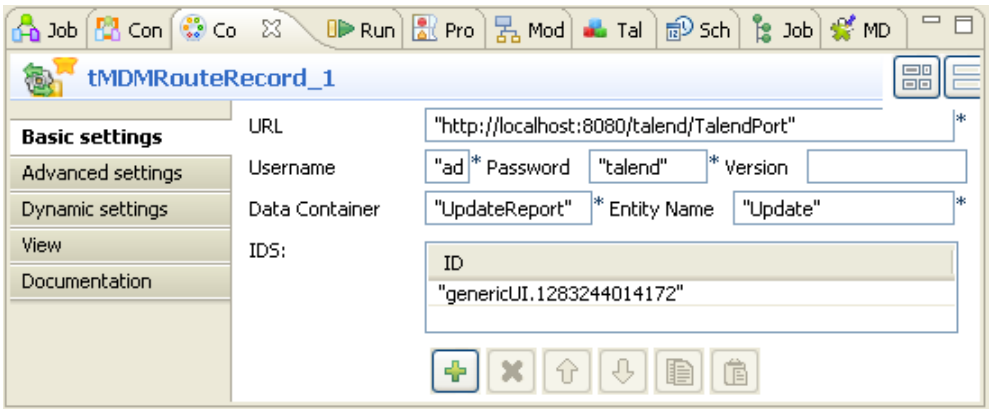


- In the **Value** column, enter a value for this line. In this example, it is exactly *UPDATE*.
- In the **Condition Id** column, enter a unique identifier for the condition you want to set, for example, *C1*.
- In the **Conditions** area, enter the query you want to undertake on the data record using the condition ID *C1* you set earlier.
- Press **Ctrl+S** to save the trigger.
- In the **MDM server** tree view, double click **Data container** > **system** > **UpdateReport** to open the **Data Container Browser UpdateReport** view. An Update Report is a complete track of all create, update or delete actions on any master data

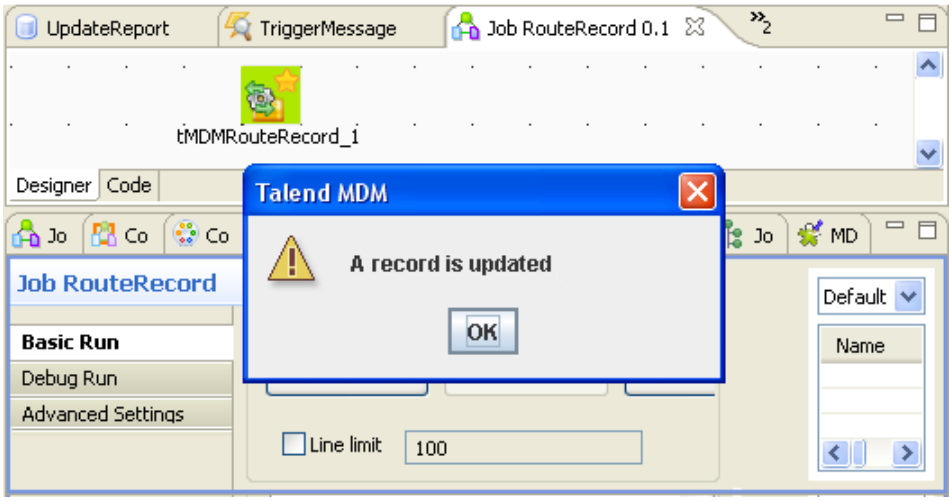


- Next to the **Entity** field of this view, click the button to search all the action records in the **UpdateReport**. Note that the **Update** entity does not necessarily mean that the corresponding action recorded is the update, as it is just the entity name defined by the data model of **UpdateReport** and may record different actions including create, delete, update.

- The last record corresponds to what is done on the product record at the beginning of the scenario. The primary key of this record is *genericUI.1283244014172* and this is the record that will be routed to **Event trigger**.
- On the menu bar of the studio, click **Window** > **Perspective** > **Integration** to design the Job routing a record.
- On the **Integration** perspective, create a Job and name it *RouteRecord*.
- To do so, right-click **Job Designs**, in the **Repository** tree view. In the contextual menu, select **Create Job**.
- A wizard opens. In the **Name** field, type in *RouteRecord*, and click **Finish**.
- Drop the **tMDMRouteRecord** component from the **Palette** onto the design workspace.
- Double click this component to open its **Component** view.



- In the **URL** field, enter the address of your **MDM server**. This example uses *http://localhost:8080/talend/TalendPort*.
- In the **Username** and the **Password** fields, type in the connection parameters.
- In the **Data Container** field, enter the data container name that stores the record you want to route. It is *UpdateReport* in this example.
- In the **Entity Name** field, enter the entity name that the record you want to route belongs to. In this example, the entity name is *Update*.
- In the **IDS** area, click the plus button under the table to add a new line.
- In the newly added line, fill in the primary key of the record to be routed to **Event Manager**, that is, *genericUI.1283244014172*, as was read earlier from the **Data Container Browser UpdateReport**.
- Press **F6** to run this Job. **Event Manager** calls the process to execute the *message* Job and generate the dialog box informing the user that this recorded has been updated.



This component submits the primary key of the record noting the update to **Event Manager**. When **Event Manager** checks this record and finds that this record meets the conditions you have defined on the trigger *TriggerMessage*'s configuration view, it calls the process that launches the *message* Job to pop up the dialog box informing the user of this update.

tMDMSP

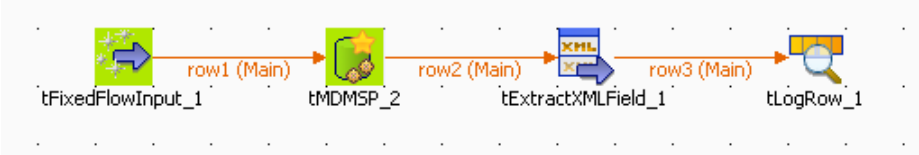
tMDMSP Properties

Component family	Talend MDM	
Function	**tMDMSP** calls the MDM Hub stored procedure.	
Purpose	**tMDMSP** offers a convenient way to centralize multiple or complex queries in a MDM Hub and call them easily.	
Basic settings	*Schema* and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: see *Talend Open Studio for Big Data User Guide*.
	Use an existing connection	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL of the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the master data management Version you want to connect to, for which you have the user rights required. Leave this field empty if you want to display the default perspective.
	Data Container	Type in the name of the data container that stores the procedure you want to call.
	SP Name	Type in the exact name of the Stored Procedure
	Parameters (in order)	Click the Plus button and select the various **Input Columns** that will be required by the procedures. The SP schema can hold more columns than there are parameters used in the procedure.
Connections		Outgoing links (from one component to another): **Row**: Main **Trigger**: Run if; On Component Ok; On Component Error, On Subjob Ok, On Subjob Error. Incoming links (from one component to another): **Row**: Main, Iterate; **Trigger**: Run if, On Component Ok, On Component Error, On Subjob Ok, On Subjob Error For further information regarding connections, see *Talend Open Studio for Big Data User Guide*.
Usage	This component is used as intermediary component. It can be used as start component but only no input parameters are thus needed for the procedure to be called. An output link is required.	

Limitation	N/A

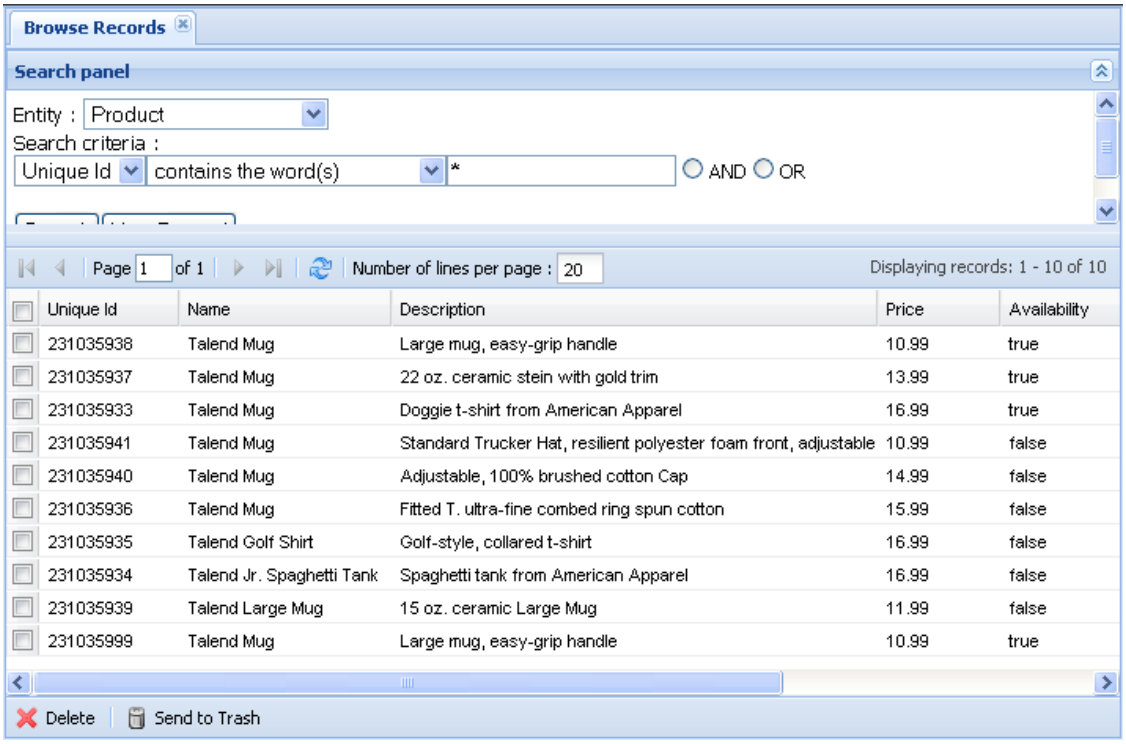
Scenario: Executing a stored procedure in the MDM Hub

The following job is intended for calculating the total price of each kind of products recorded on your MDM Web UI.

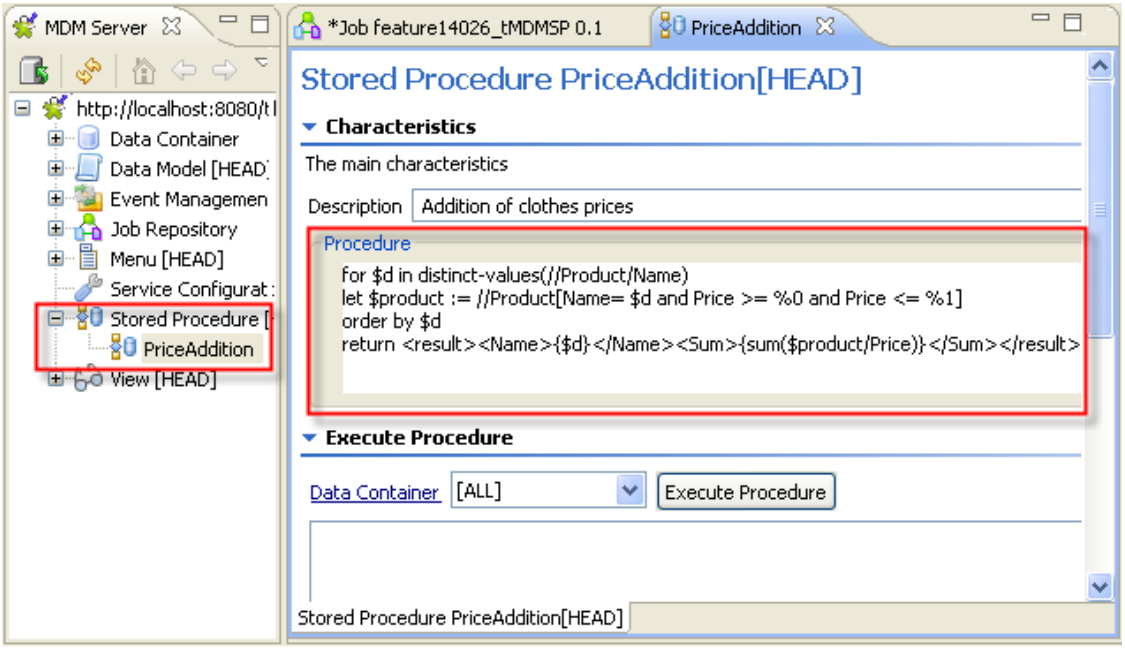


This Job will generate parameters used to execute a stored procedure in the MDM Hub, then extract the desired data from the returned XML-format result and present the extracted data in the studio.

The products of which the prices are to be treated are listed on your MDM Web UI.



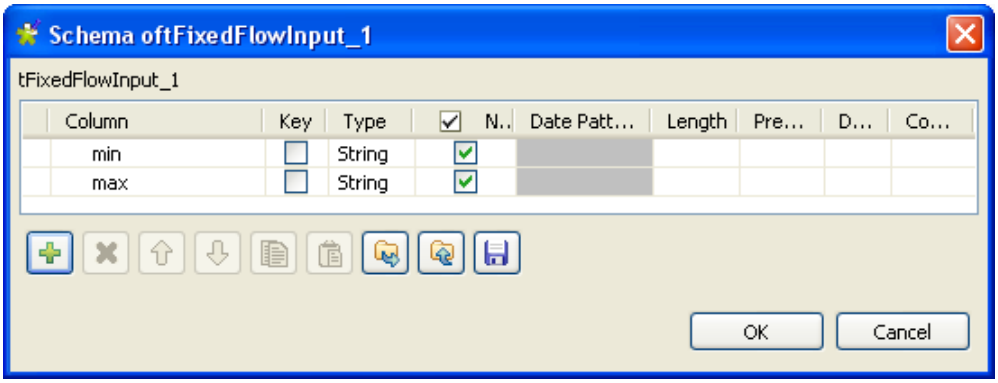
The stored procedure to be executed can be found in **Stored Procedure** node of the MDM server's tree view and reads as follows:



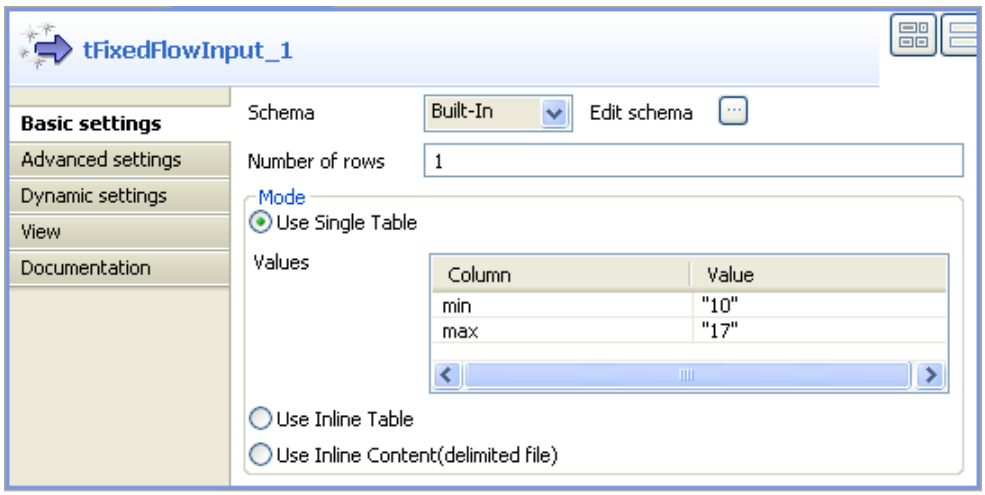
For more information on a stored procedure in the MDM server, see *Talend Open Studio for MDM* Administrator Guide.

To realize this Job, proceed as follows:

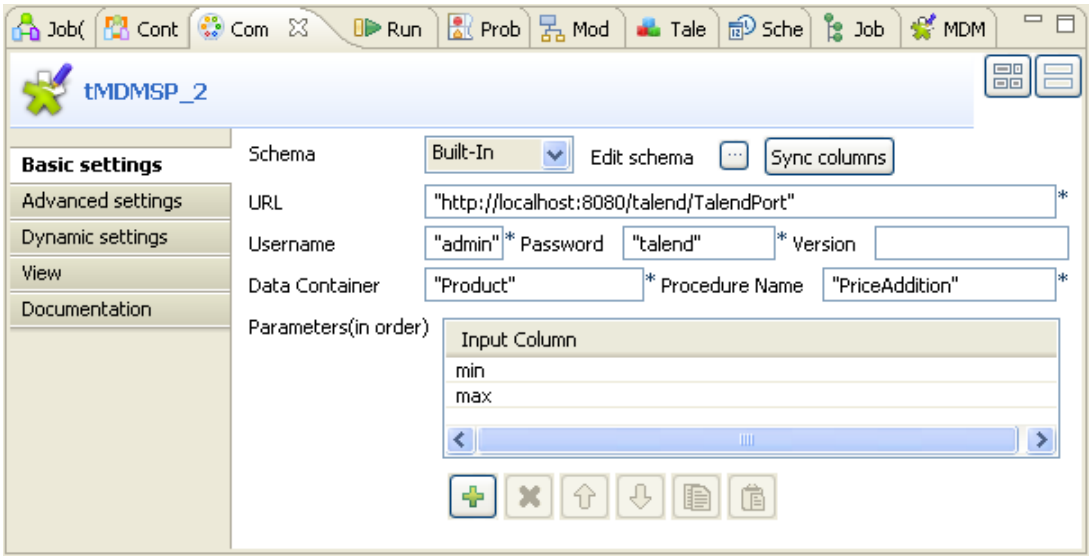
- Drag and drop the following components used in this example: **tFixedFlowInput**, **tMDMSP**, **tExtractXMLField**, **tLogRow**.
- Connect the components using the **Row Main** link.
- The **tFixedFlowInput** is used to generate the price range of your interest for this calculation. In this example, define 10 as the minimum and 17 as the maximum in order to cover all of the products.
- Double-click on **tFixedFlowInput** to open its **Component** view.
- On the **Component** view, click the **[...]** button next to **Edit schema** to open the schema editor of this component.
- In the schema editor, add the two parameters *min* and *max* that are used to define the price range.



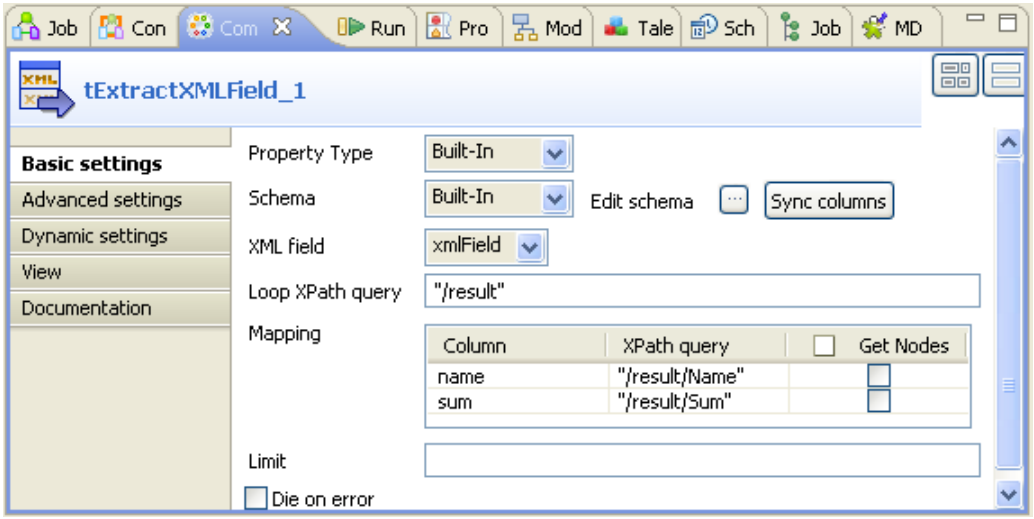
- Click **OK** to validate this editing.
- On the **Values** table in the **Mode** area of the **Component** view, the two parameters *min* and *max* that you have defined in the schema editor of this component display.
- In the **Value** column of the **Values** table, enter 10 for the *min* parameter and 17 for the *max* parameter.



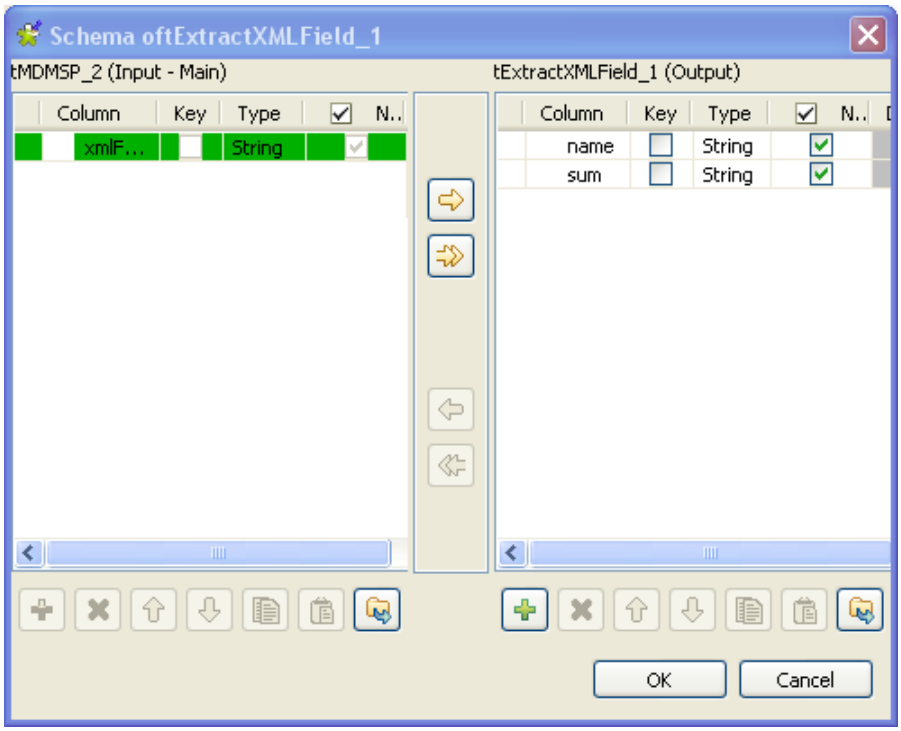
- Double-click on **tMDMSP** to open its **Component** view.



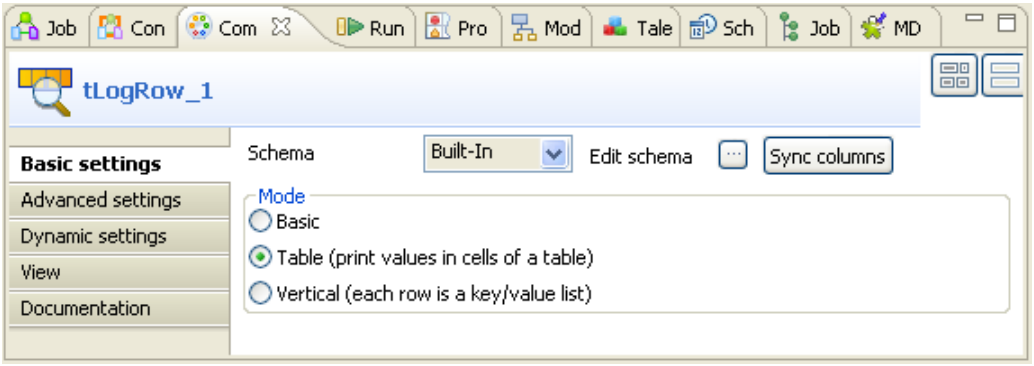
- In the **URL** field of the **Component** view, type in the MDM server address, in this example, *http://localhost:8080/talend/TalendPort*.
- In **Username** and **Password**, enter the authentication information, in this example, *admin* and *talend*.
- In **Data Container** and **Procedure Name**, enter the exact names of the data container *Product* and of the stored procedure *PriceAddition*.
- Under the **Parameters (in order)** table, click the plus button two times to add two rows in this table.
- In the **Parameters (in order)** table, click each of both rows you have added and from the drop-down list, select the *min* parameter for one and the *max* parameter for the other.
- Double-click on **tExtractXMLField** to open its **Component** view.



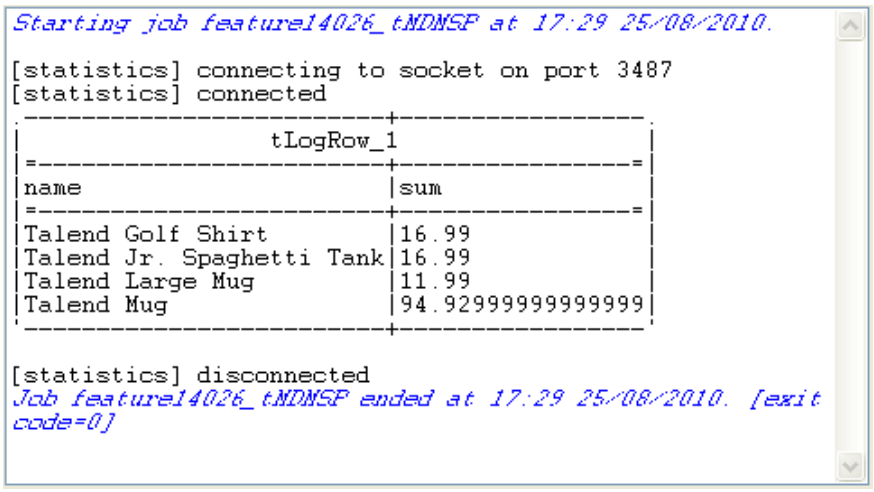
- On the **Component** view, click the **[...]** button next to **Edit schema** to open the schema editor of this component.
- In the schema editor, add two columns to define the structure of the outcoming data. These two columns are *name* and *sum*. They represent respectively the name and the total price of each kind of product recorded in the MDM Web UI.



- Click **OK** to validate the configuration and the two columns display in the **Mapping** table of the **Component** view.
- In the **Loop XPath query** field, type in the node of the XML tree, which the loop is based on. In this example, the node is */result* as you can read in the procedure code: `return <result><Name>{$d}</Name><Sum>{sum($product/Price)}</Sum></result>`.
- In **XPath query** of the **Mapping** table, enter the exact node name on which the loop is applied. They are */result/Name* used to extract the product names and */result/Sum* used to extract the total prices.
- Eventually, double-click **tLogRow** to open its **Component** view.



- Synchronize the schema with the preceding component.
- And select the **Print values in cells of a table** check box for reading convenience.
- Then press **F6** to execute the Job.
- See the outcoming data in the console of the **Run** view.



The output lists the four kinds of products recorded in the MDM Web UI and the total price for each of them.

tMDMTriggerInput

tMDMTriggerInput properties

Component family	Talend MDM	
Function	Once executed, **tMDMTriggerInput** reads the XML message (**Document** type) sent by MDM and passes them to the component that follows. This component works alongside the new trigger service and process plug-in in MDM version 5.0 and higher. The MDM Jobs, triggers and processes developed in previous MDM versions remain supported. However, we recommend using this component when designing new MDM Jobs.	
Purpose	Every time when you save a change in your MDM, the corresponding change record is generated in XML format. At runtime, this component reads this record and sends the relative information to the following component. With this component, you do not need to configure your Job any more in order to communicate the data changes from MDM to your Job.	
Basic settings	*Property Type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally. **tMDMTriggerInput** is expected to use this option in order to apply the default read-only schema. **MDM_message** is the only column of this schema.
		Repository: Select the repository file where properties are stored. The fields that follow are completed automatically using the fetched data. As **tMDMTriggerInput** provides a fixed read-only schema, you are expected to use the **Built-in** option.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in.
		Built-in: The schema will be created and stored for this component only. Related Topic: see *Talend Open Studio for Big Data* **User Guide**. This is the default option for **tMDMTriggerInput**. With this option, the read-only schema is used to deal with the XML-format MDM message.
		Repository: The schema already exists and is stored in the repository. You can reuse it in various projects and jobs. Related Topic: see *Talend Open Studio for Big Data* **User Guide**. As **tMDMTriggerInput** provides a fixed read-only schema, you are expected to use the **Built-in** option.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component as a start component. It needs an output flow. To receive the message from MDM, you need to deploy the Job using this component on your MDM server and generate the corresponding trigger and process in MDM to invoke this Job. For further information about how to deploy a Job onto MDM server and how to generate a trigger or a process, see *Talend Open Studio for MDM* **Administrator Guide**.	

	For further information about how to change a record in MDM, see *Talend MDM Web User Interface* **User Guide**.
Limitation	During the deployment of this component on the MDM server, you need to select the **Hosted (Zip)** type as the format of the deployed Job. If you deploy it in the **Distributed (War)** type, the relative Job cannot be invoked. For further information about the available types, see *Talend Open Studio for MDM* **Administrator Guide**.

Scenario: Exchanging the event information about an MDM record

In this scenario, a four-component Job is used to exchange the event information about a product record. Using an established MDM connection from the **Repository**, this Job is triggered by *Talend Open Studio for MDM* once you have updated a product record.

To replicate this scenario, accomplish the following tasks sequentially:

1. Create an MDM connection of the **Receive** type in the **Repository** of the Studio. This connection is to the MDM hub holding the record you want to update.

2. Create the Job receiving and sending the MDM update message.

3. Generate the process invoking this Job created.

4. Update a specific MDM record.

To create the MDM records, model and container used in this scenario, you can execute the Jobs in the MDM demo project in the integration Studio and then update the MDM server to deploy the objects thus created for them to be taken into account at runtime. You will use this server all through this scenario.

For further information about how to import a demo project, see *Talend Open Studio for Big Data User Guide*.

For further information about how to update the server for deploying objects, see *Talend Open Studio for MDM Administrator Guide*.

For further information about an MDM event and the event management, see *Talend Open Studio for MDM Administrator Guide*.

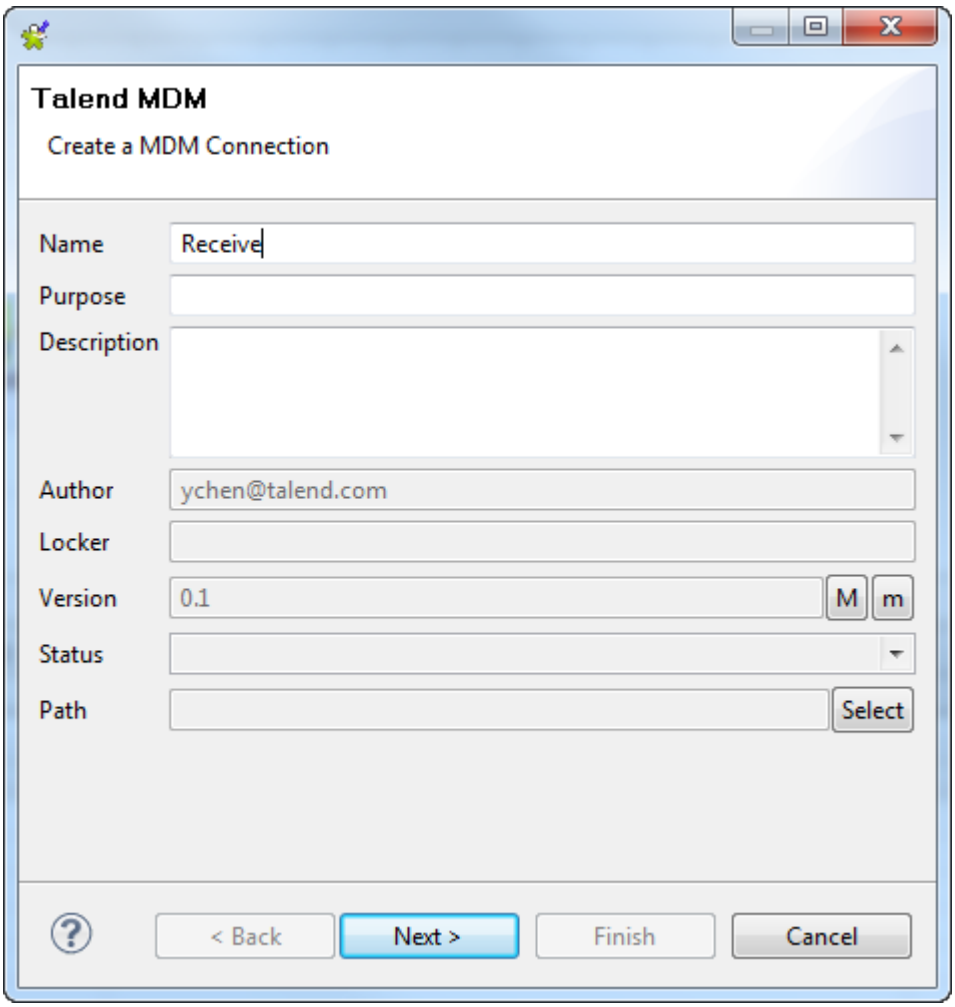
Creating an MDM connection

Establishing the connection

1. Launch the MDM server with which you need to communicate the update message.

2. In the **Integration** perspective of the Studio, expand the **Metadata** node in the **Repository**.

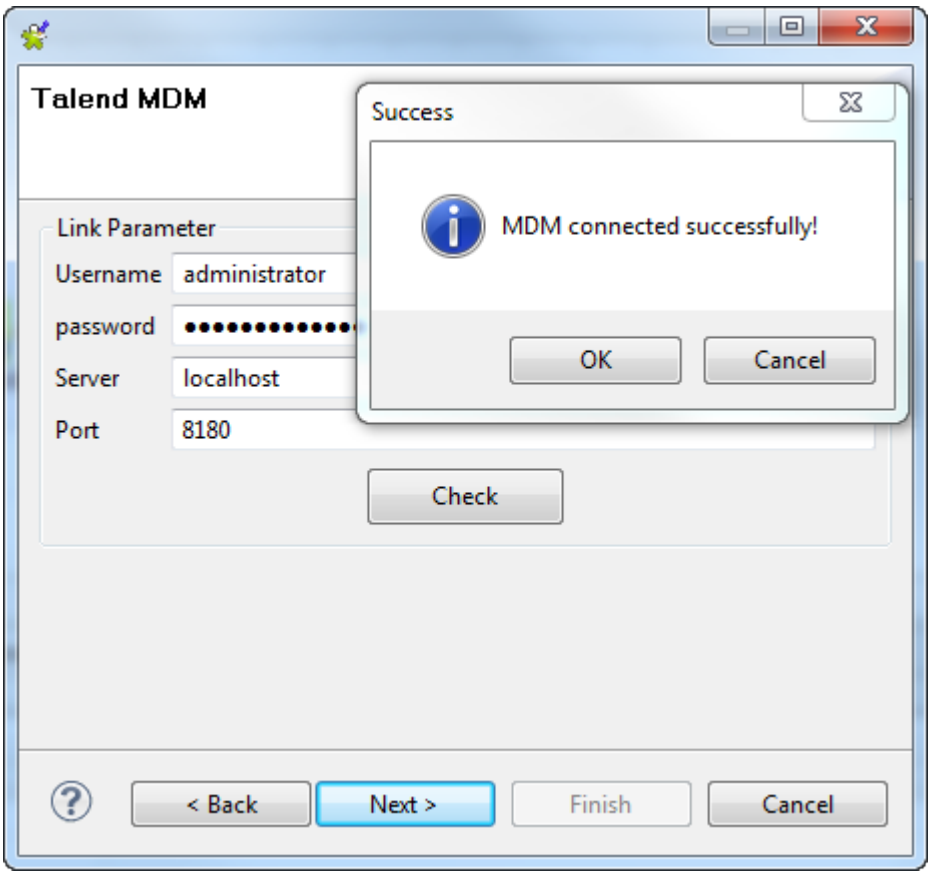
3. Right-click the **Talend MDM** item and select **Create MDM connection**.



4. Enter the **Name** you want to use for this connection and if required, added the **Purpose** and the **Description** in the corresponding fields. For example, we name this connection as *receive_update*.

5. In the **Next** step, enter the authentication information used to connect to the MDM web service through which you manage the record to be updated.

   Once you click the **Check** button and the connection is shown successful, the **Next** button becomes clickable.



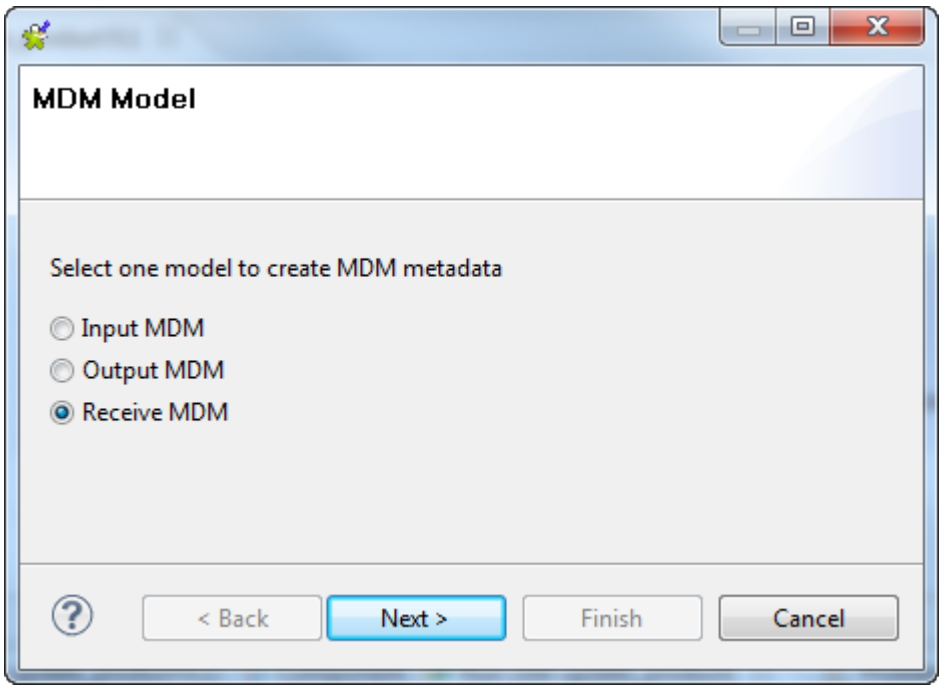
6. In the **Next** step, select the **Version**, the **Data model** and the **Data Container** used by the record to be updated. In this scenario, the model and the container are both *Product*.

7. Click **Finish** to validate the creation. The connection created appears under the **Metadata** node in the **Repository**.

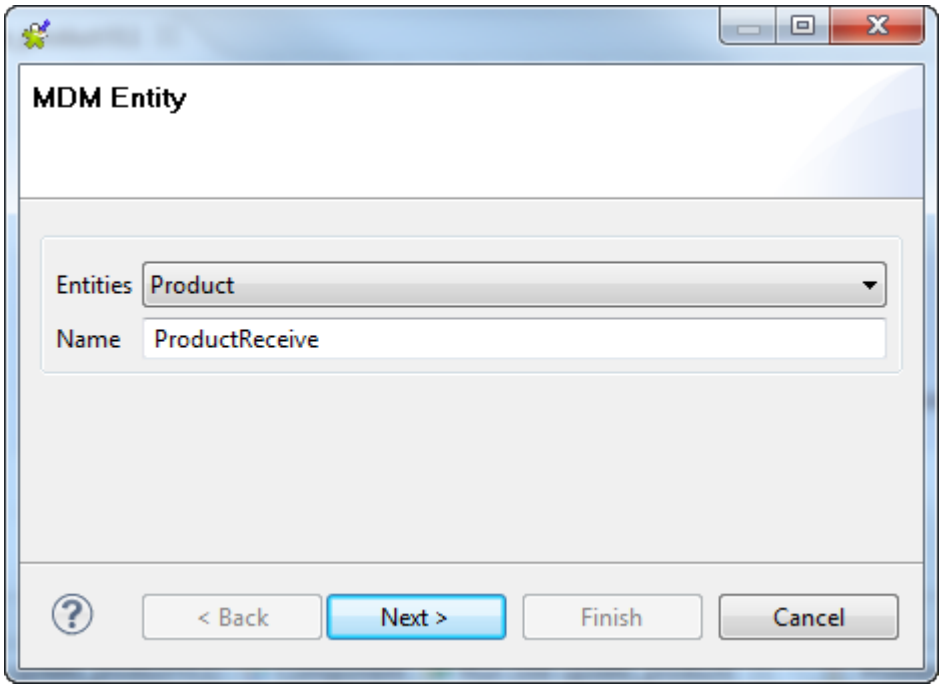
Retrieving entities

1. Right-click the connection created and from the contextual menu, select **Retrieve entities**. Then the wizard appears.

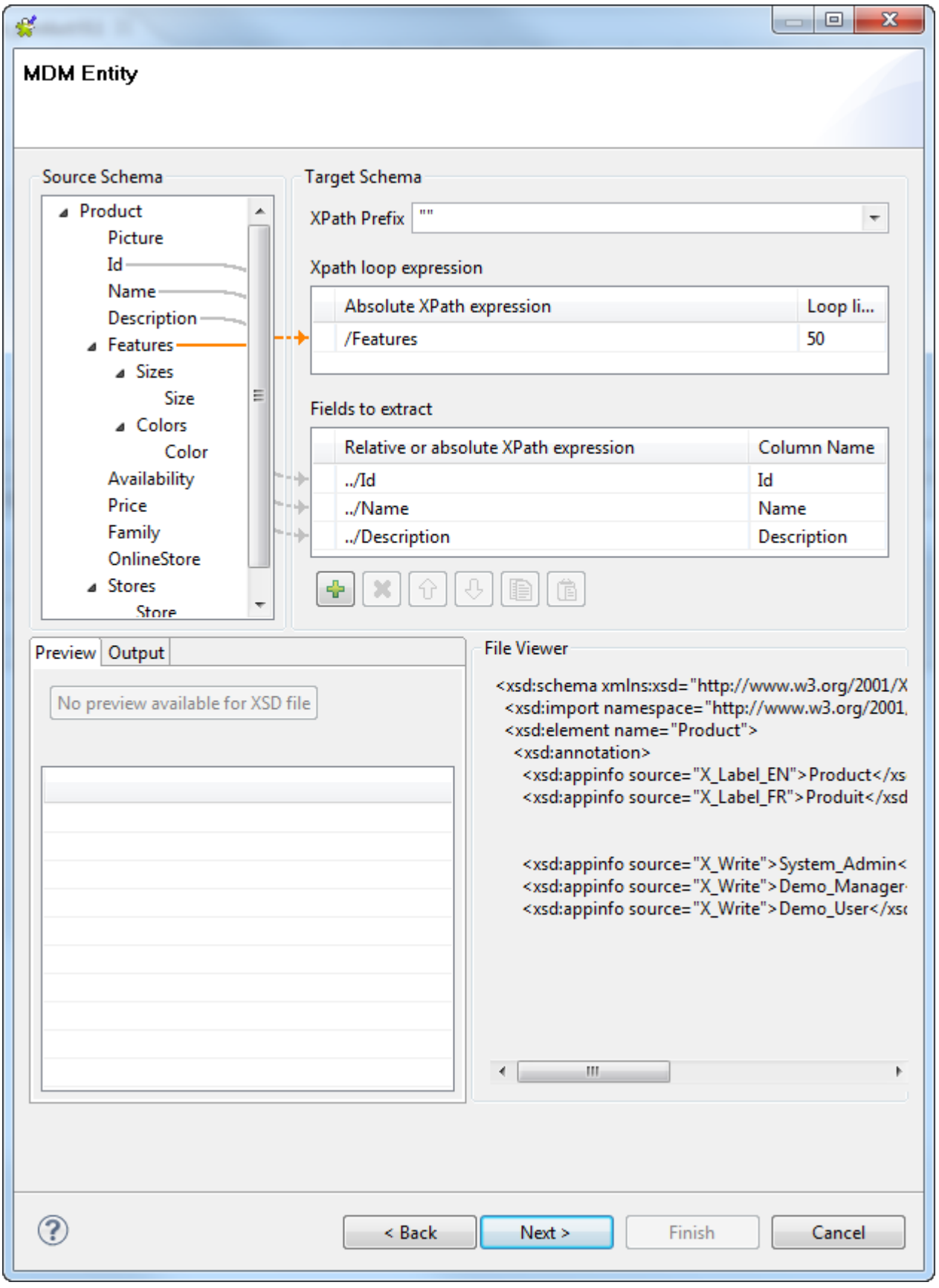
2. Select **Receive MDM** and click **Next** to continue.



3. Select the entity to be retrieved. In this scenario, it is *Product*. Then the name field is entered automatically.



4. In the **Next** step, drop the elements you need to retrieve from the **Source Schema** area to the **Target Schema** area. In this scenario, the *Features* element is the loop and the *Id*, the *Name* and the *Description* elements are the fields to extract.



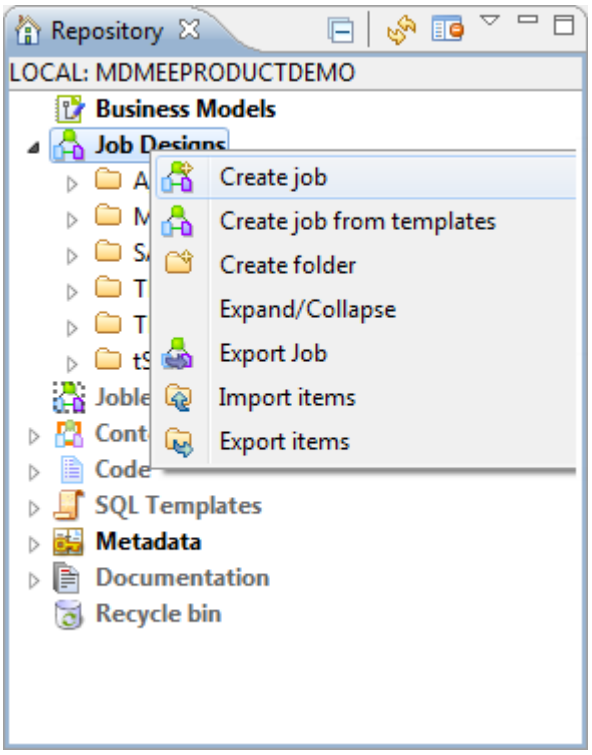
5. In the **Next** step, if required, change the description of the schema retrieved; otherwise, click **Finish** to finalize retrieving this entity. In this scenario, we keep the default schema description and click **Finish**.

The schema of the product entity is retrieved. For further information about the container and the data model used by the MDM, see *Talend Open Studio for MDM Administrator Guide*.

Creating the Job communicating the MDM message

Linking the components

1. In the **Integration** perspective of the Studio, select **Create Job** from the **Job Design** node in the **Repository** tree view. Then the **New Job** wizard appears.

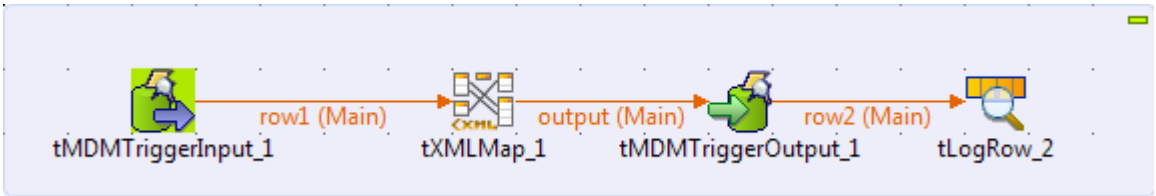


2. Name this new Job and click **Finish** to close the wizard and validate the creation. An empty Job is opened on the workspace of the Studio.

3. Drop **tMDMTriggerInput**, **tXMLMap**, **tMDMTriggerOutput** and **tLogRow** from **Palette** onto the workspace.

4. Right-click **tMDMTriggerInput** and from the contextual menu, select the **Row** > **Main** link to connect it to **tXMLMap**.

5. Do the same to connect **tXMLMap** to **tMDMTriggerOutput**. When doing so, a dialog box appears to prompt you to name this link created.



6. Double-click **tMDMTriggerOutput** to open its **Component** view.

7. Click **Edit schema** to open the editor.

8. Select the single pre-defined column of **tMDMTriggerOutput**, then, click to reproduce this column on the input side (left).

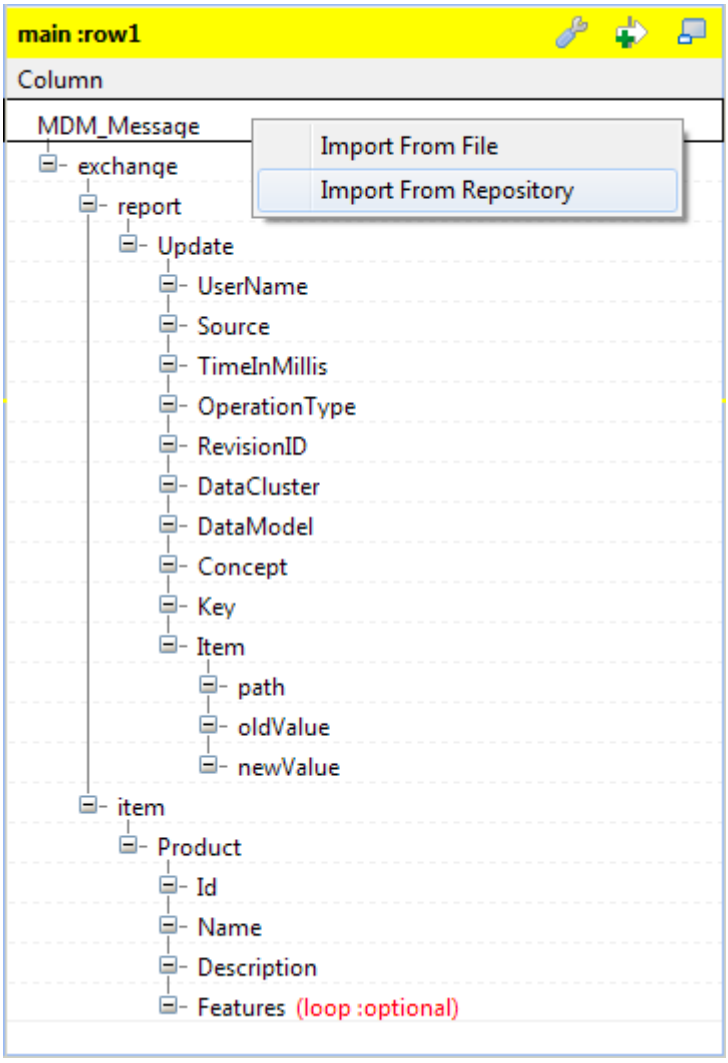
Configuring the transformation of the MDM message

1. Double-click **tXMLMap** to open its editor.

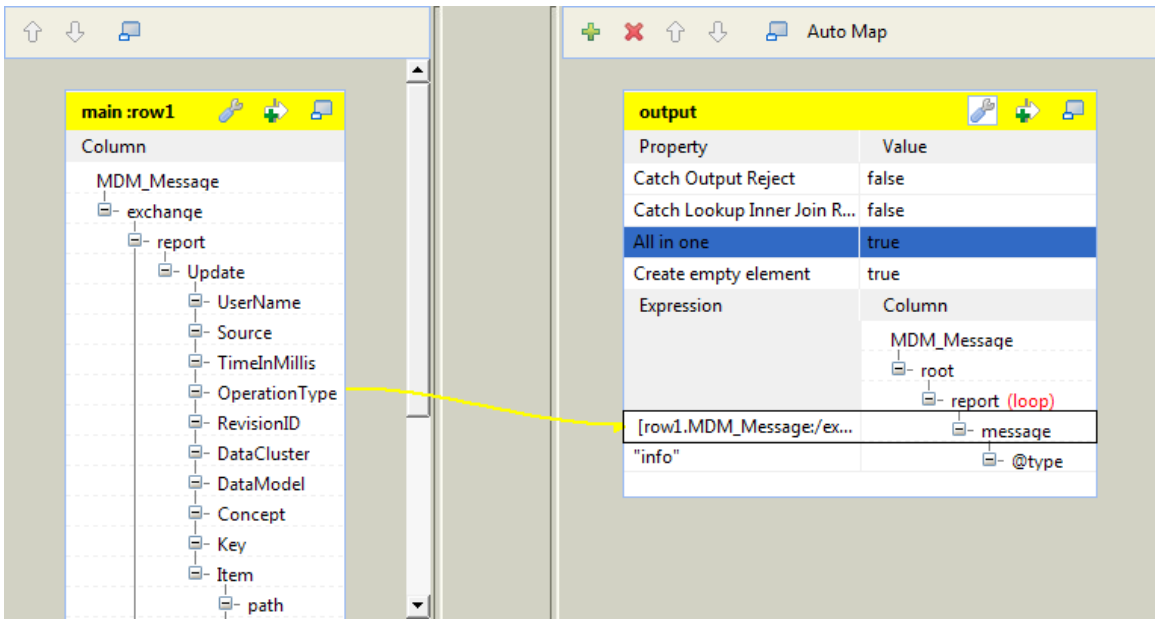
2. In the table representing the input flow (up-left of the editor), right-click the column name **MDM_Message** on the top of the XML tree and select **Import from repository**. The **[Metadata]** wizard appears.

3. Select the entity schema retrieved earlier using the **Receive MDM** model, then click OK. In this scenario, the entity schema is *ProductReceive*.

4. A dialog box appears prompting you to add the schema of the Update Report to the input XML tree. Click **OK** to accept it. This builds a complete input document for an MDM event. In the input XML tree, the *Features* element is set as loop element automatically.



5. In the table representing the output flow (up-right of the editor), develop the output XML tree as presented in the figure below. This tree is constructed depending on the required static model of the MDM output report.



The XML construct required to return the validation-success message is

```
<report><message type="info">message</message></report>
```

The XML construct required to return the validation-failure message is

```
<report><message type="error">message</message></report>
```

6. Map the *OperationType* element on the input side with the *message* element on the output side. This will output the information about the type of the event occurring on the MDM record.

   To get more information, you can build the concatenation of the input elements you need to extract in the **Expression** column of this message element. Both **tMap** and **tXMLMap** allow you to edit expressions using the expression editor. For further information about how to edit an expression, see *Talend Open Studio for Big Data User Guide*.

7. In the **Expression** column, enter `"info"` in the row corresponding to *@type*.

8. Click the pincer icon to display the output settings panel, then set the **All in one** option as **true**.

9. Click **OK** to close the editor and validate these changes.

10. Double click **tLogRow** to open its **Component** view, then, click **Sync columns**.

This Job is finalized. For further information about the input document and the output report of an MDM event, see *Talend Open Studio for MDM Administrator Guide*.

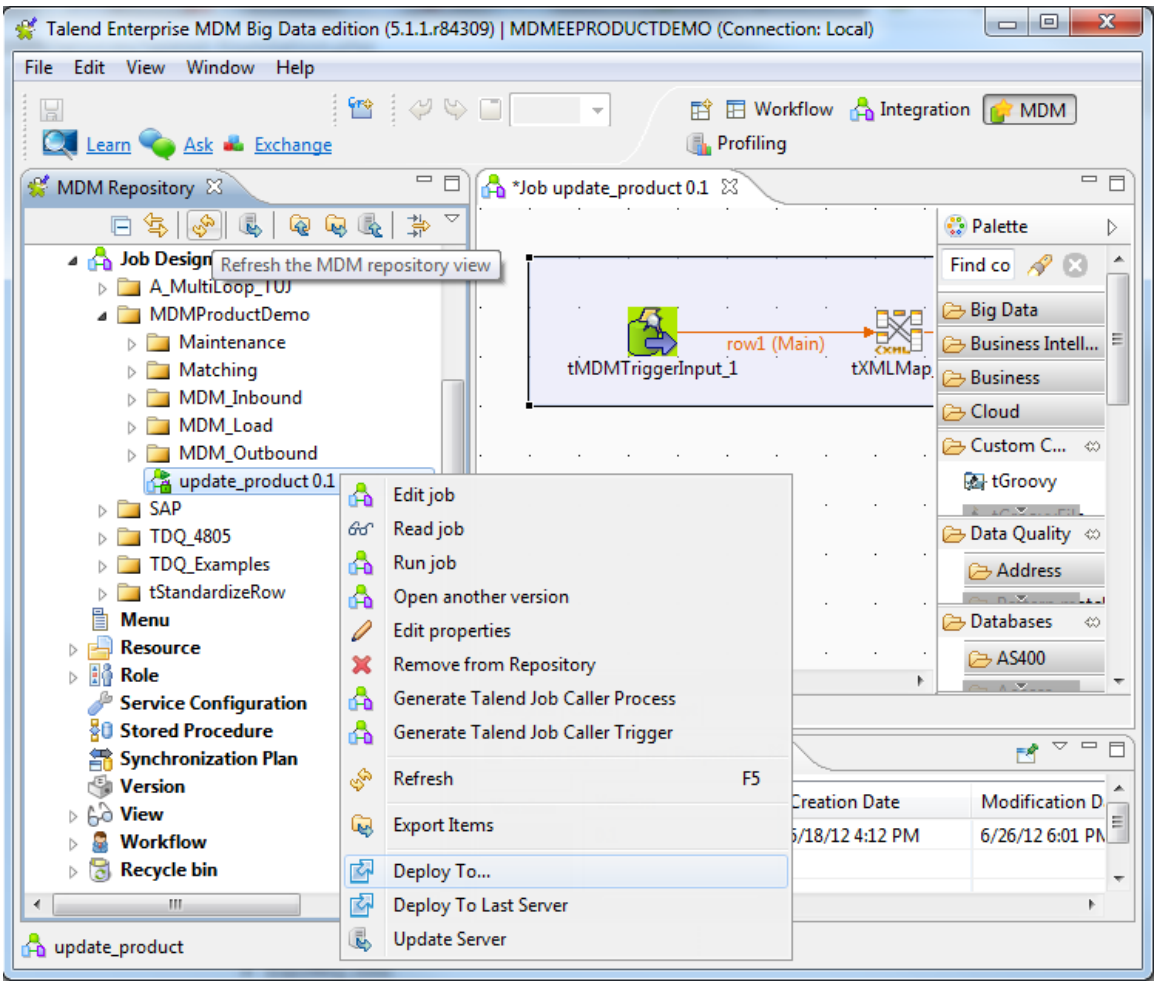
Generating the process invoking the Job created

Deploying the Job to be called onto the MDM server

1. Switch to the **MDM** perspective by clicking the corresponding button in the up-right corner of the Studio.

2. In **MDM Repository**, click the refresh button so that the Job created appears under the **Job Designs** node of this Repository's tree view.

3. Right-click this Job created, *update_product* in this scenario, and from the contextual menu, select **Deploy to** in order to deploy it to the MDM server.

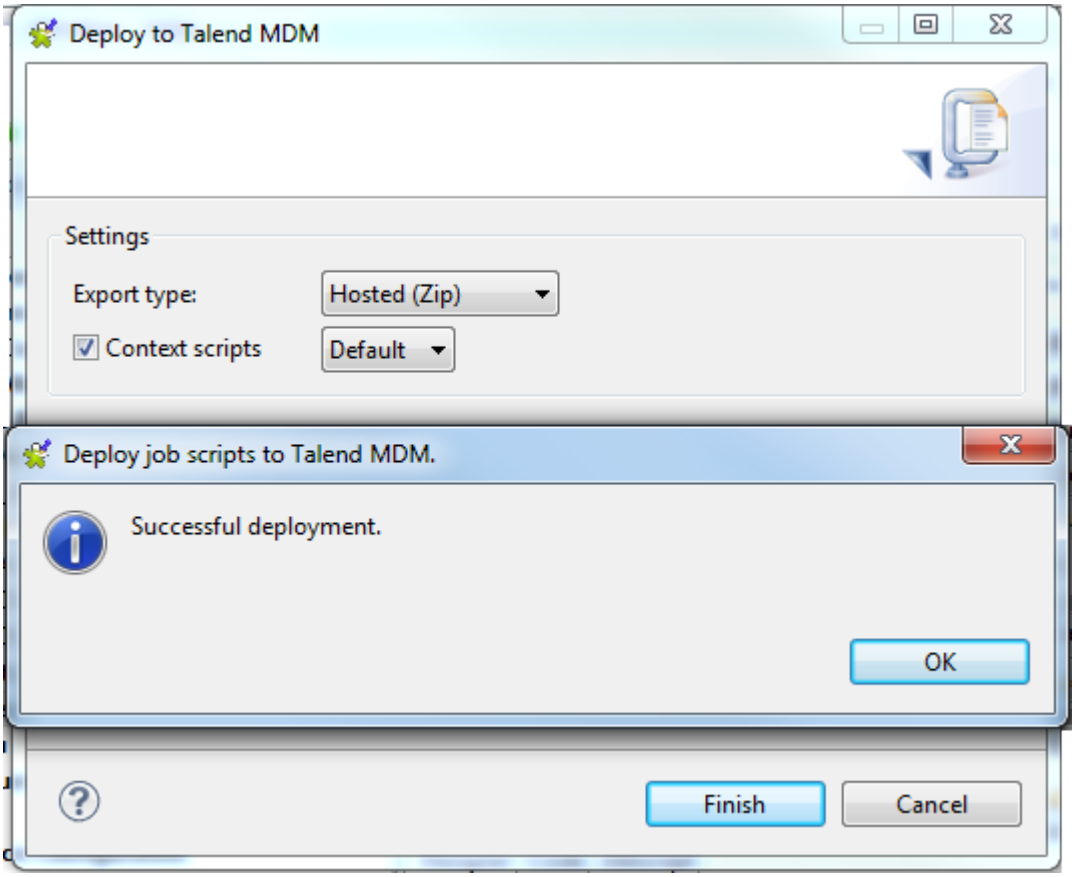


4. The deployment wizard appears. From the server list, select the MDM server you are using, then click **OK**.

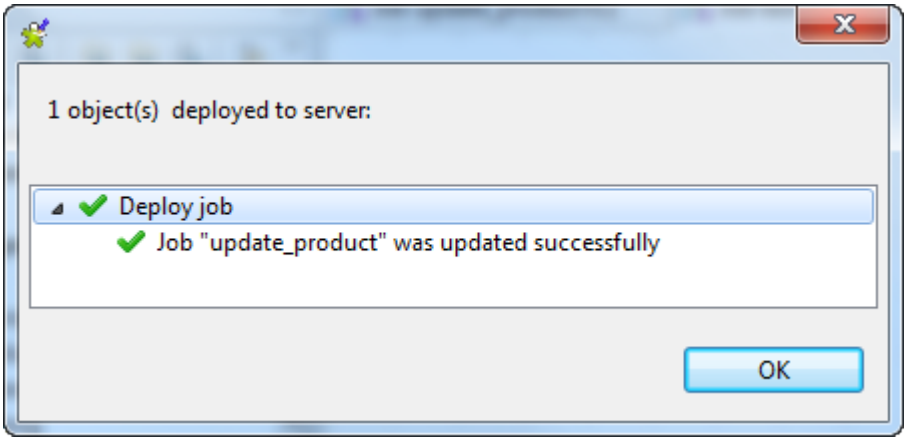
5. In the **[Deploy to Talend MDM]** window that pops up, select the **Export type** and the **Context scripts** for the Job to be deployed. In this scenario, keep the default settings: **Export type** is **Hosted (zip)** and **Context scripts** is **Default**.

   For further information about these settings, see *Talend Open Studio for MDM Administrator Guide*.

6. Click **Finish** to validate these settings and start the deployment. When the deployment is done, a message box pops up to indicate that the deployment is successful.



7. Click **OK** to close this message box, then a window pops up to list the objects deployed. In this scenario, it is the Job, *update_product*.

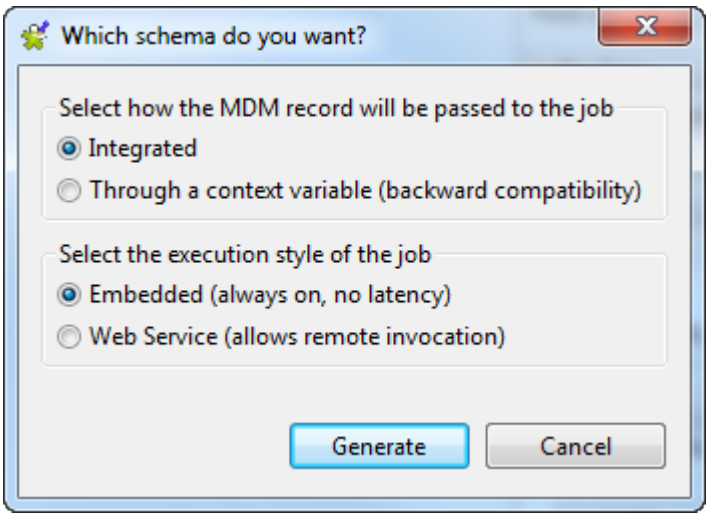


8. Click **OK** to terminate the deployment procedure.

Generating the process used to call the Job

1. Right-click the Job *update_product* again and select **Generate Talend Job Caller Process** from the contextual menu.

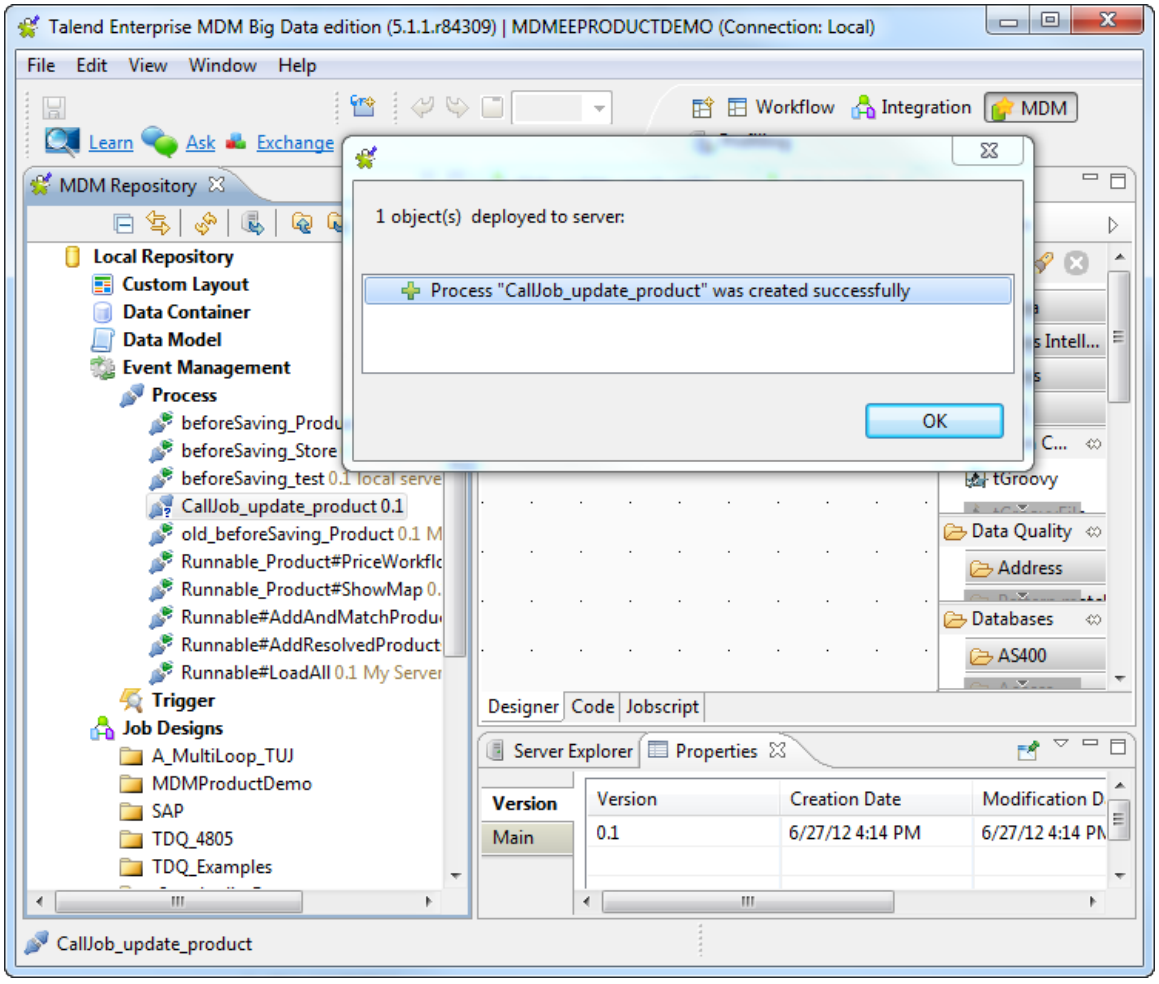
2. In the pop-up window, keep the default settings for this scenario: **Integrated** and **Embedded**. For further information about the available options in this window, see *Talend Open Studio for MDM Administrator Guide*.



3. Click **Generate** to start the generation. Once done, a process named *CallJob_update_product* appears under the **Process** node in **MDM Repository**.

4. Right-click this process, then select **Deploy to** from the contextual menu to deploy it onto the MDM server.

5. In the pop-up wizard, select the server you are using, then , click **OK** to open the window listing the objects deployed.



6. Click **OK** to close this window and finalize the deployment. The question mark disappears from the icon of this process.

7. In **MDM Repository**, right-click the *CallJob_update_prodcut* process, then select **Rename** from the contextual menu.

8. In the pop-up window, rename this process as *beforeSaving_update_product* depending on the required process naming pattern. Then click **OK** to validate it.

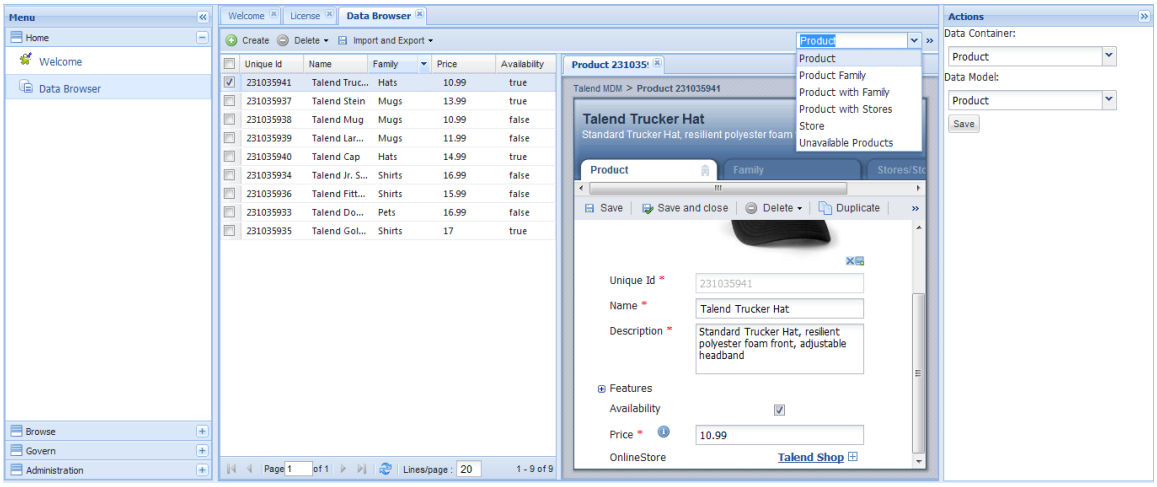
9. Deploy this process again as described earlier.

Updating a product record

1. Log in the web service of the MDM hub you are using.

2. In the **Actions** panel on the right side, verify the **Data Container** and the **Data Model** you are using are both *Product*.

3. In the **Data Browser** page, launch the search in the product entities so as to list all the available product records



4. Select the product record you need to update from the list, for example, *Talend Trucker Hat*. The details of this record appears in the *Product* tab view.

5. Update one of its attributes. For example, update the price to *11.00*, then click **Save**.

   The message about the operation type of this event has been sent to the MDM server and thanks to **tLogRow**, this message is displayed on the window of this MDM server.

```
10:33:21,244 INFO  [STDOUT] <?xml version="1.0" encoding="UTF-8"?>
<root><report><message type="info">UPDATE</message></report></root>
```

For further information about how to use the MDM web service, see *Talend MDM Web User Interface User Guide*

tMDMTriggerOutput

tMDMTriggerOutput properties

Component family	Talend MDM	
Function	**tMDMTriggerOutput** receives an XML flow (**Document** type) from its preceding component. This component works alongside the new trigger service and process plug-in in MDM version 5.0 and higher. The MDM Jobs, triggers and processes developed in previous MDM versions remain supported. However, we recommend using this component when designing new MDM Jobs.	
Purpose	This component receives an XML flow to set the MDM message so that MDM retrieves this message at runtime. With this component, you do not need to configure your Job any more in order to communicate the data changes from MDM to your Job.	
Basic settings	*Property Type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally. **tMDMTriggerOutput** is expected to use this option in order to apply the default read-only schema. **MDM_message** is the only column of this schema.
		Repository: Select the repository file where properties are stored. The fields that follow are completed automatically using the fetched data. As **tMDMTriggerOutput** provides a fixed read-only schema, you are expected to use the **Built-in** option.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to modify the schema. If you modify the schema, it automatically becomes built-in.
		Built-in: The schema will be created and stored for this component only. Related Topic: see *Talend Open Studio for Big Data* **User Guide**. This is the default option for **tMDMTriggerOutput**. With this option, the read-only schema is used to deal with the XML-format MDM message.
		Repository: The schema already exists and is stored in the repository. You can reuse it in various projects and jobs. Related Topic: see *Talend Open Studio for Big Data* **User Guide**. As **tMDMTriggerOutput** provides a fixed read-only schema, you are expected to use the **Built-in** option.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component as an end component. It needs an input flow. To send message to MDM, you need to deploy the Job using this component on your MDM server and generate the corresponding trigger and process to invoke this Job in MDM. For further information about how to deploy a Job onto MDM server and how to generate a trigger or a process, see *Talend Open Studio for MDM* **Administrator Guide**.	
Limitation	During the deployment of this component on the MDM server, you need to select the **Hosted (Zip)** type as the format of the deployed Job. If you deploy it in the **Distributed (War)** type, the relative	

	Job cannot be invoked. For further information about the available types, see *Talend Open Studio for MDM* **Administrator Guide**.

Related scenario

For a related scenario, see section *Scenario: Exchanging the event information about an MDM record*

tMDMViewSearch

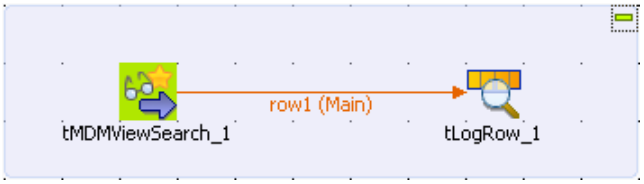
tMDMViewSearch properties

Component family	Talend MDM	
Function	**tMDMViewSearch** selects records from an MDM Hub by applying filtering criteria you have created in a specific view. The resulting data is in XML structure. For more information on a view on which you can define filtering criteria, see *Talend Open Studio for MDM* Administrator Guide.	
Purpose	This component allows you to retrieve the MDM records from an MDM hub.	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either **Built-in** or remote in the **Repository**. Click **Edit Schema** to modify the schema. Note that if you modify the schema, it automatically becomes built-in. Click **Sync columns** to collect the schema from the previous component.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	XML Field	Select the name of the column in which you want to write the XML data.
	Use an existing connection	Select this check box if you want to use a configured **tMDMConnection** component.
	URL	Type in the URL of the MDM server.
	Username and *Password*	Type in the user authentication data for the MDM server.
	Version	Type in the name of the master data management Version you want to connect to, for which you have the user rights required. Leave this field empty if you want to display the default perspective.
	Data Container	Type in the name of the data container that holds the master data you want to read.
	View Name	Type in the name of the view whose filters will be applied to process the records.
	Operations	Complete this table to create the WHERE clause. The parameters to be set are: - **XPath**: define the path expression to select the XML node at which point the filtering is operated. - **Functions**: select an operator from the drop-down list, like **Contains**, **Starts with**, **Equals**, etc. - **Value**: type in the value you want to retrieve. - **Predicate**: select the predicate to combine the filtering conditions in different manners. The predicate may be **none**, **or**, **and**, **exactly**, etc. *The parameters are case sensitive.*
	Order (One Row)	Complete this table to decide the presentation order of the retrieved records. The parameters to be set are:

		- **XPath**: define the path expression to select the XML node at which point the sorting operation is performed. - **Order**: select the presentation order that may be **asc** (ascending) or **desc** (descending). *The parameters are case sensitive.* *For the time being, only the first row created in the Order table is valid.*
	Spell Threshold	Set it to -1 to deactivate this threshold. This threshold is used to decide the spell checking level.
	Skip Rows	Type in the count of rows to be ignored to specify from which row the process should begin. For example, if you type *8* in the field, the process will begin from the *9th* row.
	Max Rows	Type in the maximum number of rows to be processed. If Limit = 0, no row is read or processed. By default, the limit is -1, meaning that no limit is set.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to retrieve specific records.	
Global Variables		**Number of Lines**: Indicates the number of lines processed. This is available as an **After** variable. Returns an integer. For further information about variables, see *Talend Open Studio for Big Data* **User Guide**.
Connections		Outgoing links (from one component to another): **Row**: Iterate **Trigger**: Run if; On Component Ok; On Component Error, On Subjob Ok, On Subjob Error. Incoming links (from one component to another): **Row**: Iterate; **Trigger**: Run if, On Component Ok, On Component Error, On Subjob Ok, On Subjob Error For further information regarding connections, see *Talend Open Studio for Big Data* **User Guide**.
Limitation	n/a	

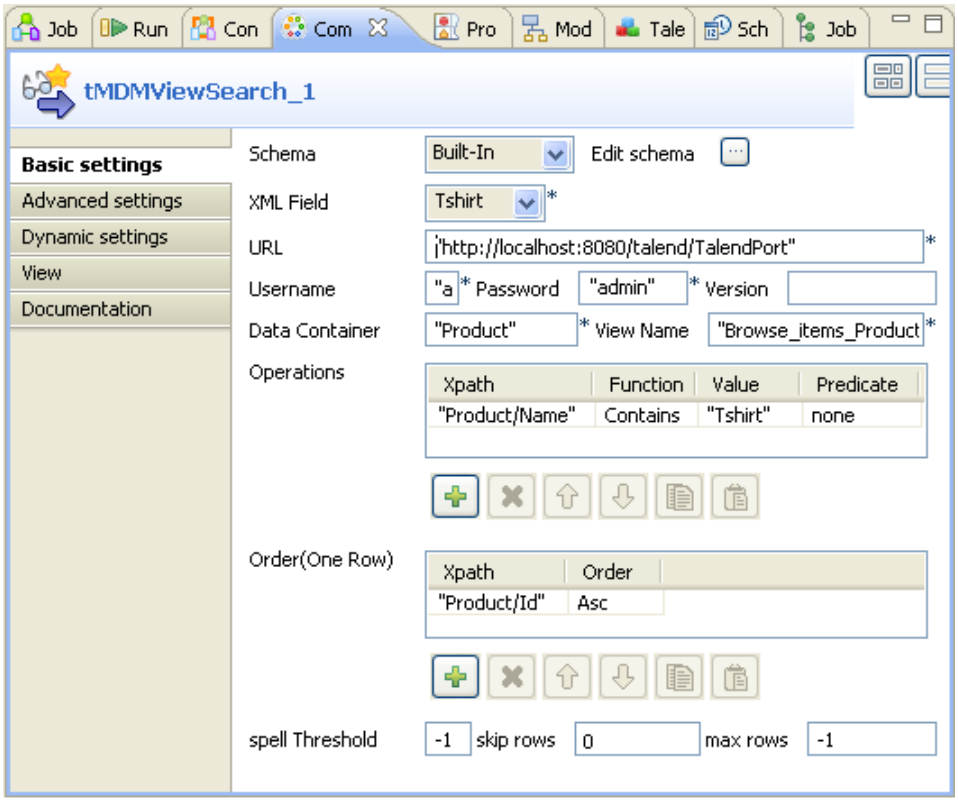
Scenario: Retrieving records from an MDM hub via an existing view

This scenario describes a two-component Job that retrieves a data record in XML structure.

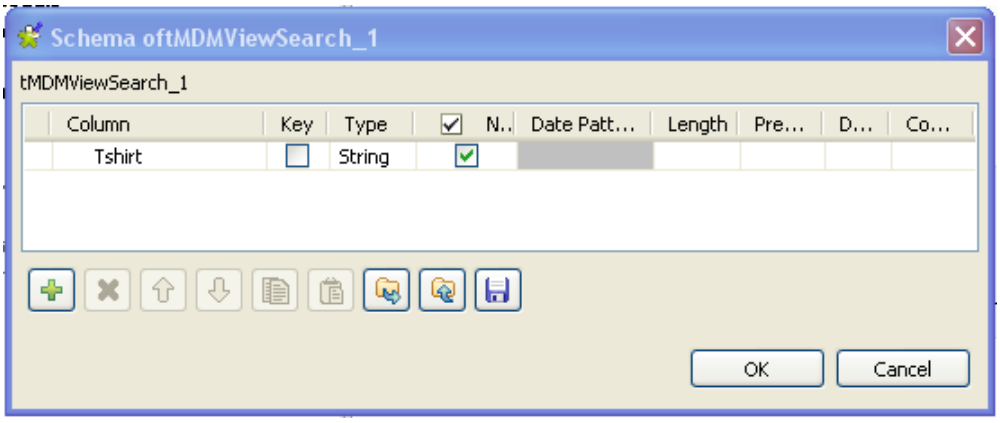


In this example, you will select the T-shirt information from the *Product* entity via the *Browse_items_Product* view created from *Talend Open Studio for Big Data*. Each record in the entity contains the details defined as filtering criteria: *Id*, *Name*, *Description* and *Price*.

- From the **Palette**, drop **tMDMViewSearch** and **tLogRow** onto the design workspace.
- Connect the components using a **Row Main** link.
- Double-click **tMDMViewSearch** to view its **Basic settings**, in the **Component** tab and set the component properties.

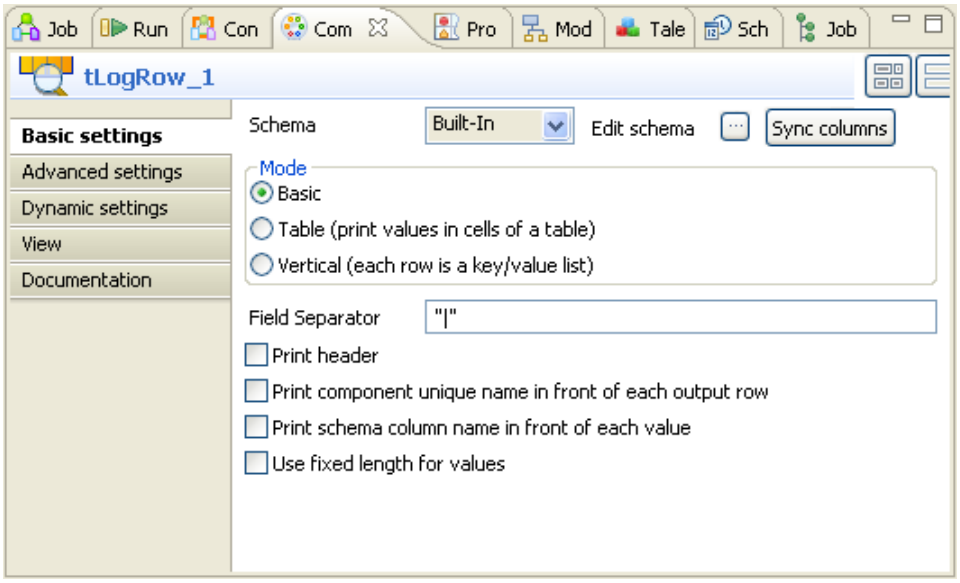


- In the **Schema** list, select **Built-In** and click the three-dot button next to **Edit schema** to open a dialog box in which you can define the structure of the XML data you want to write in.



- Click the plus button and add one column of the type **String**. Name the column as *Tshirt*.
- Click **OK** to validate your creation and proceed to the next step.
- In the **XML Field** field, select *Tshirt* as the column you will write the retrieved data in.

- Use your MDM server address in the **URL** field and type in the corresponding connection data in the **Username** and the **Password** fields. In this example, use the default url, then enter *admin* as username as well as password.
- In the **Data Container** field, type in the container name: *Product*.
- In the **View Name** field, type in the view name: *Browse_item_Product*.
- Below the **Operations** table, click the plus button to add one row in this table.
- In the **Operations** table, define the **XPath** as *Product/Name*, meaning that the filtering operation will be performed at the *Name* node, then select *Contains* in the **Function** column and type in *Tshirt* in the **Value** column.
- Below the **Order (One Row)** table, click the plus button to add one row in this table.
- In the **Order (One Row)** table, define the **XPath** as *Product/Id* and select the **asc** order for the **Order** column.
- In the design workspace, click **tLogRow** to open its **Basic settings** view and set the properties.



- Next to the three-dot button used for editing schema, click **Sync columns** to acquire the schema from the preceding component.
- Press **F6** to execute the Job.

```
Starting job feature14031_tMDMViewSearch at 17:37 10/08/2010.

[statistics] connecting to socket on port 3920
[statistics] connected
<result>
    <Id>1</Id>
    <Name>Tshirt-V</Name>
    <Description>green</Description>
    <Price>16.00 </Price>
</result>
<result>
    <Id>2</Id>
    <Name>Tshirt-G</Name>
    <Description>Green</Description>
    <Price>17.00</Price>
</result>
[statistics] disconnected
Job feature14031_tMDMViewSearch ended at 17:37 10/08/2010.
[exit code=0]
```

In the console docked in the **Run** view, you can read the retrieved *Tshirt* records in XML structure, which are sorted in the ascending order.

Technical components

This chapter details the components you can find in the **Technical** group of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The Technical components are Java-oriented components that perform very technical actions such as loading data in memory (in small subset of information) and keep it to allow its reuse at various stage of the processing.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tHashInput

tHashInput Properties

This component is used along with **tHashOutput**. It reads from the cache memory data loaded by **tHashOutput**. Together, these twin components offer high-speed data access to facilitate transactions involving a massive amount of data.

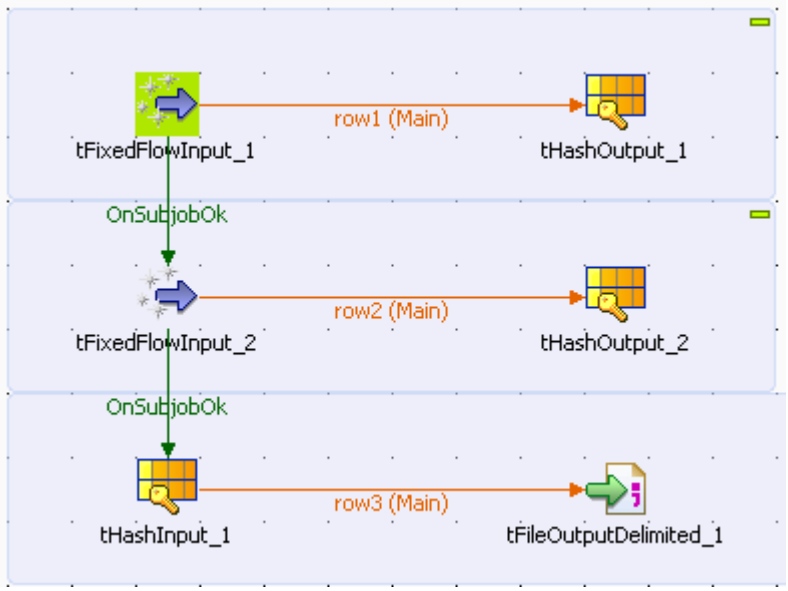
Component family	Technical	
Function	**tHashInput** reads from the cache memory data loaded by **tHashOutput** to offer high-speed data stream.	
Purpose	This component reads from the cache memory data loaded by **tHashOutput** to offer high-speed data feed, facilitating transactions involving a large amount of data.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema is created and stored locally for this component only. Related topic: see the *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: see the *Talend Open Studio for Big Data User Guide*.
	Link with a tHashOutput	Select this check box to connect to a **tHashOutput** component. It is always selected by default.
	Component list	Drop-down list of available **tHashOutput** components.
	Clear cache after reading	Select this check box to clear the cache after reading the data loaded by a certain **tHashOutput** component. This way, the following **tHashInput** components, if any, will not be able to read the cached data loaded by that **tHashOutput** component.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is used along with **tHashOutput**. It reads from the cache memory data loaded by **tHashOutput**. Together, these twin components offer high-speed data access to facilitate transactions involving a massive amount of data.	
Limitation	n/a	

Scenario 1: Reading data from the cache memory for high-speed data access

The following Job reads from the cache memory a huge amount of data loaded by two **tHashOutput** components and pass it to a **tFileOutputDelimited**. The goal of this scenario is to show the speed at which mass data is read and written. In practice, data feed generated in this way can be used as lookup table input for some use cases where a big amount of data needs to be referenced.

Dropping and linking the components

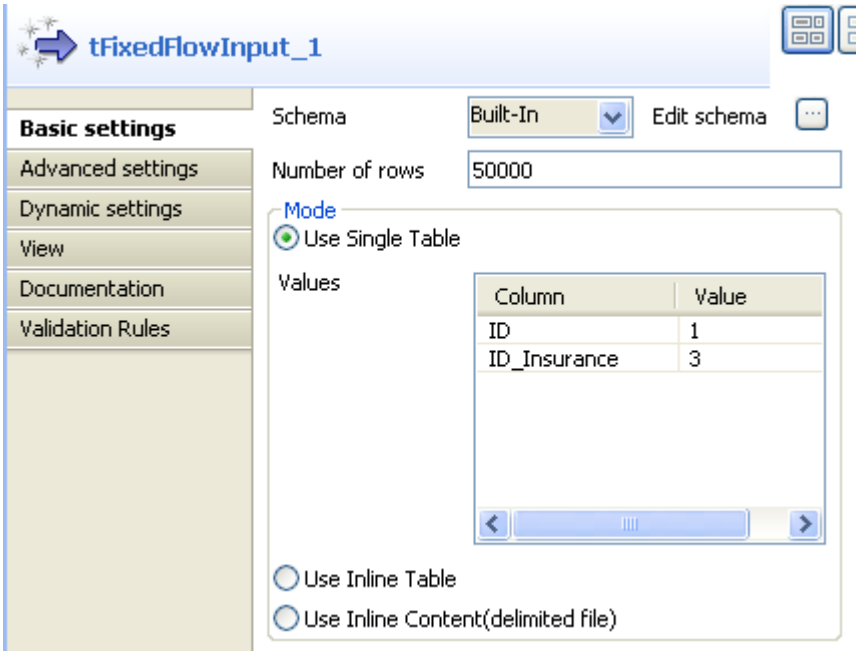
1. Drag and drop the following components from the **Palette** to the workspace: **tFixedFlowInput** (X2), **tHashOutput** (X2), **tHashInput** and **tFileOutputDelimited**.
2. Connect the first **tFixedFlowInput** to the first **tHashOutput** using a **Row** > **Main** link.
3. Connect the second **tFixedFlowInput** to the second **tHashOutput** using a **Row** > **Main** link.
4. Connect the first subjob (from **tFixedFlowInput_1**) to the second subjob (to **tFixedFlowInput_2**) using an **OnSubjobOk** link.
5. Connect **tHashInput** to **tFileOutputDelimited** using a **Row** > **Main** link.
6. Connect the second subjob to the last subjob using an **OnSubjobOk** link.



Configuring the components

Configuring data inputs and hash cache

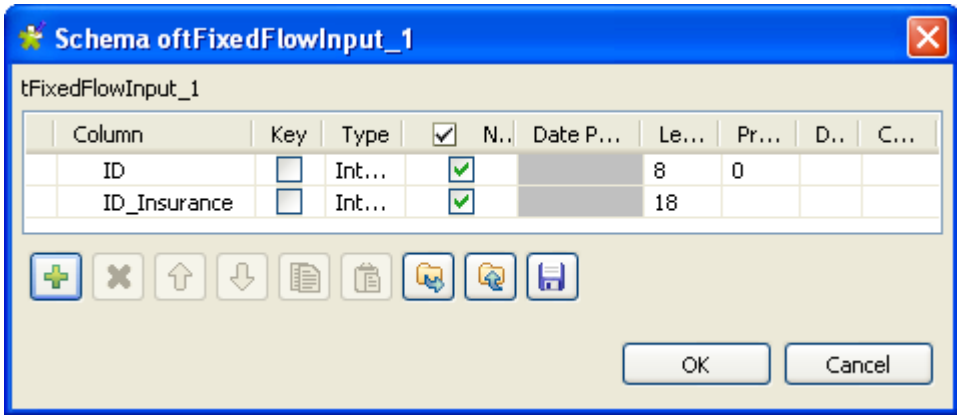
1. Double-click the first **tFixedFlowInput** component to display its **Basic settings** view.



2. Select **Built-In** from the **Schema** drop-down list.

You can select **Repository** from the **Schema** drop-down list to fill in the relevant fields automatically if the relevant metadata has been stored in the **Repository**. For more information about **Metadata**, see the *Talend Open Studio for Big Data User Guide*.

3. Click **Edit schema** to define the data structure of the input flow. In this case, the input has two columns: *ID* and *ID_Insurance*, and then click **OK** to close the dialog box.

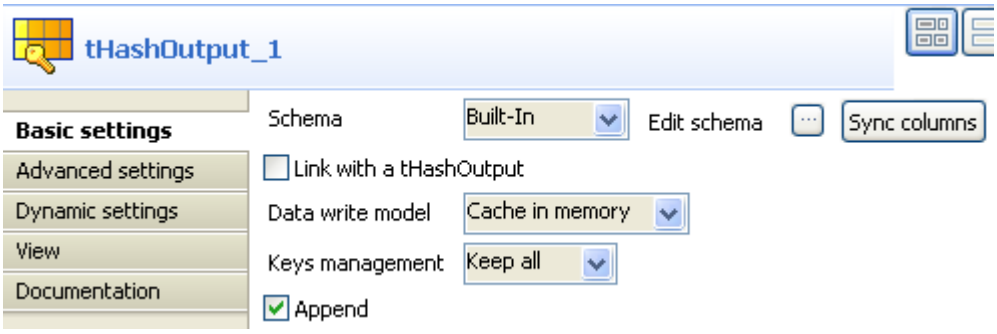


4. Fill in the **Number of rows** field to specify the entries to output, e.g. *50000*.

5. Select the **Use Single Table** check box. In the **Values** table and in the **Value** column, assign values to the columns, e.g. *1* for *ID* and *3* for *ID_Insurance*.

6. Perform the same operations for the second **tFixedFlowInput** component, with the only difference in the values. That is, *2* for *ID* and *4* for *ID_Insurance* in this case.

7. Double-click the first **tHashOutput** to display its **Basic settings** view.

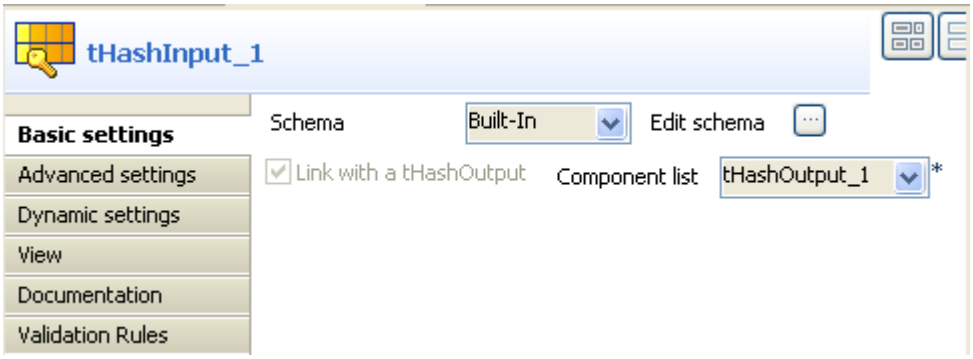


8. Select **Built-In** from the **Schema** drop-down list and click **Sync columns** to retrieve the schema from the previous component. Select **Keep all** from the **Keys management** drop-down list and keep the **Append** check box selected.

9. Perform the same operations for the second **tHashOutput** component, and select the **Link with a tHashOutput** check box.

Configuring data retrieval from hash cache and data output

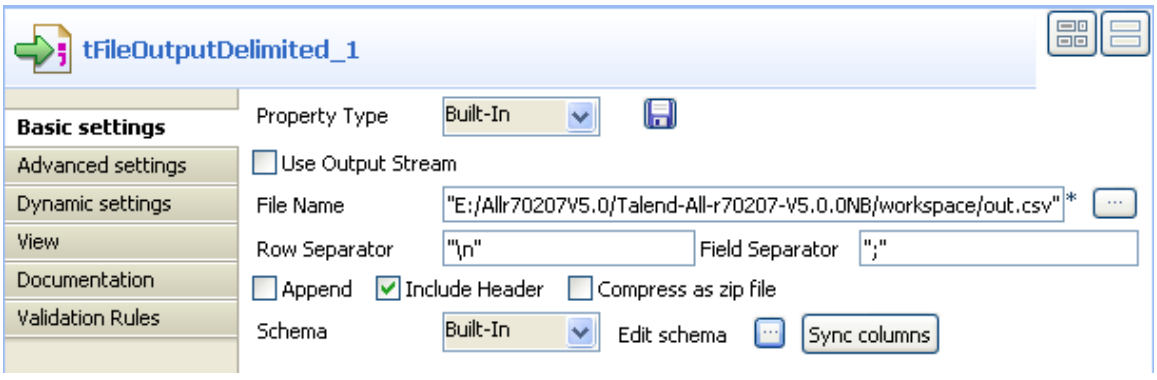
1. Double-click **tHashInput** to display its **Basic settings** view.



2. Select **Built-In** from the **Schema** drop-down list. Click **Edit schema** to define the data structure, which is the same as that of **tHashOutput**.

3. Select **tHashOutput_1** from the **Component list** drop down list.

4. Double-click **tFileOutputDelimited** to display its **Basic settings** view.



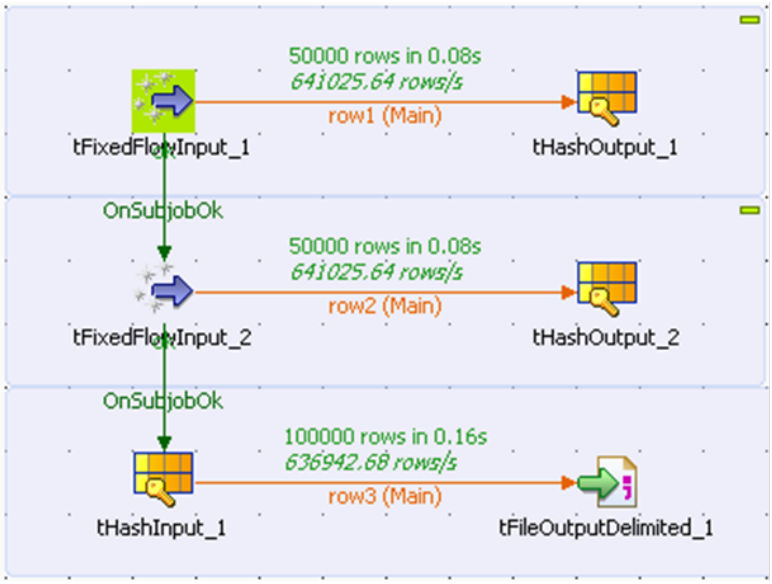
5. Select **Built-In** from the **Property Type** drop-down list. In the **File Name** field, enter the full path and name of the file, e.g. *"E:/Allr70207V5.0/Talend-All-r70207-V5.0.0NB/workspace/out.csv"*.

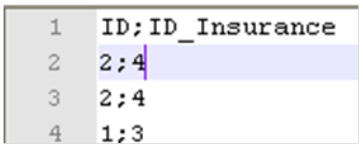
6. Select the **Include Header** check box and click **Sync columns** to retrieve the schema from the previous component.

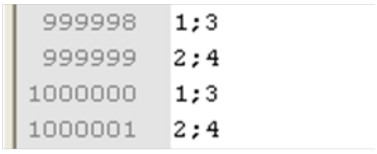
Saving and executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6**, or click **Run** on the **Run** tab to execute the Job.

   You can find that mass entries are written and read very rapidly.







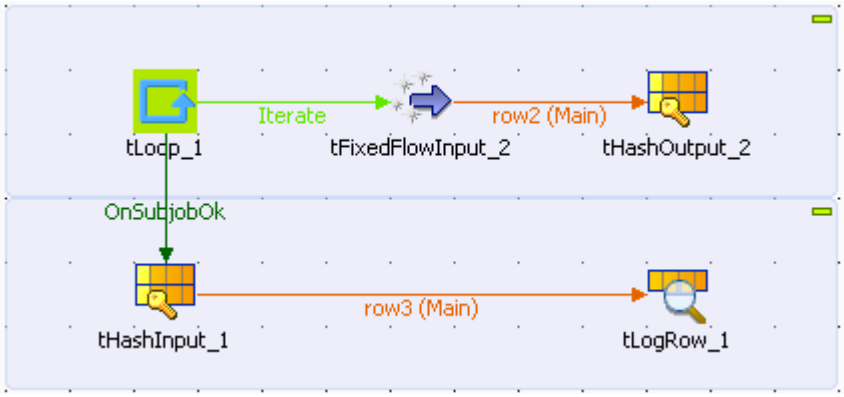
Scenario 2: Clearing the memory before loading data to it in case an iterator exists in the same subjob

In this scenario, the usage of the **Append** option of **tHashOutput** is demonstrated as it helps remove repetitive or unwanted data in case an iterator exists in the same subjob as **tHashOutput**.

To build the Job, do the following:

Dropping and linking the components

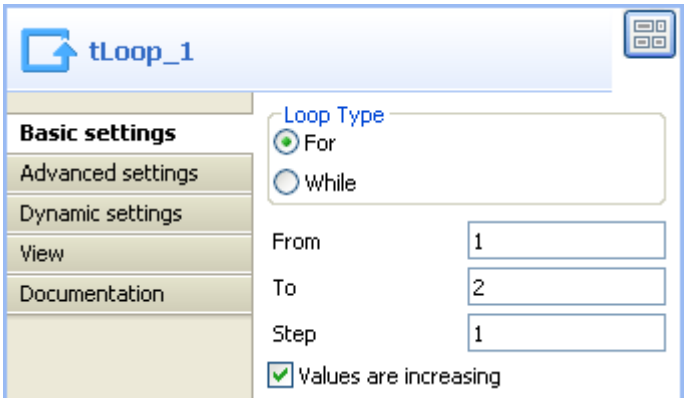
1. Drag and drop the following components from the **Palette** to the workspace: **tLoop**, **tFixedFlowInput**, **tHashOutput**, **tHashInput** and **tLogRow**.
2. Connect **tLoop** to **tFixedFlowInput** using a **Row** > **Iterate** link.
3. Connect **tFixedFlowInput** to **tHashOutput** using a **Row** > **Main** link.
4. Connect **tHashInput** to **tLogRow** using a **Row** > **Main** link.
5. Connect **tLoop** to **tHashInput** using an **OnSubjobOk** link.



Configuring the components

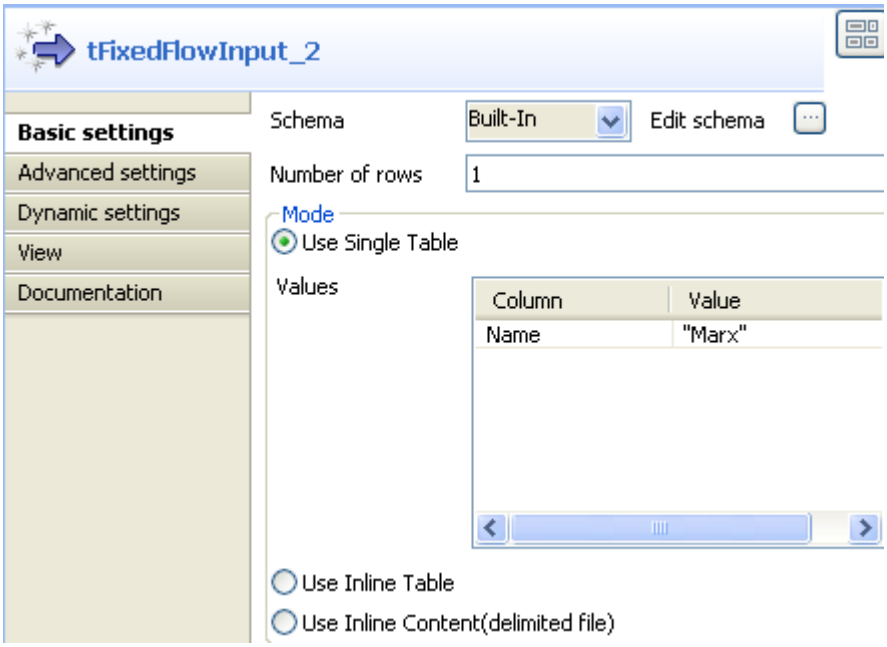
Configuring data input and hash cache

1. Double-click the **tLoop** component to display its **Basic settings** view.



2. Select **For** as the loop type. Type in *1*, *2 1* in the **From**, **To** and **Step** fields respectively. Keep the **Values are increasing** check box selected.

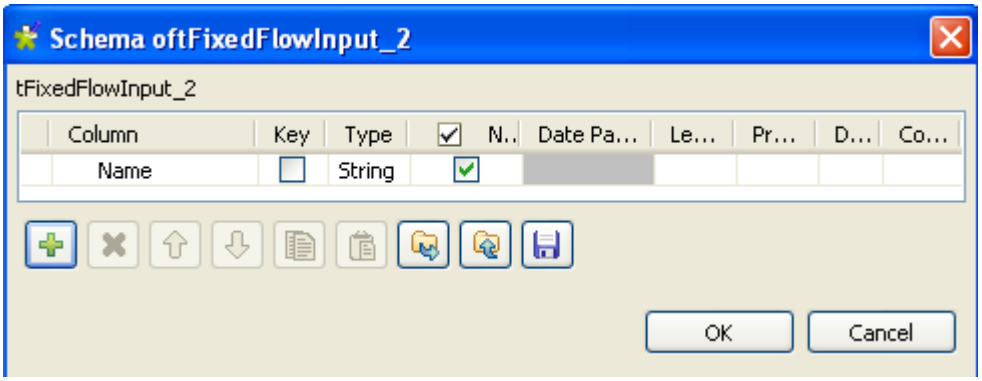
3. Double-click the **tFixedFlowInput** component to display its **Basic settings** view.



4. Select **Built-In** from the **Schema** drop-down list.

You can select **Repository** from the **Schema** drop-down list to fill in the relevant fields automatically if the relevant metadata has been stored in the **Repository**. For more information about **Metadata**, see the *Talend Open Studio for Big Data User Guide*.

5. Click **Edit schema** to define the data structure of the input flow. In this case, the input has one column: *Name*.

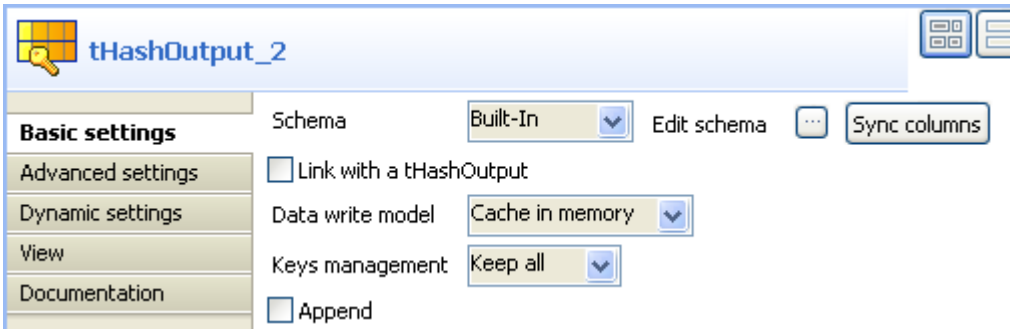


6. Click **OK** to close the dialog box.

7. Fill in the **Number of rows** field to specify the entries to output, for example *1*.

8. Select the **Use Single Table** check box. In the **Values** table, assign a value to the **Name** field, e.g. *Marx*.

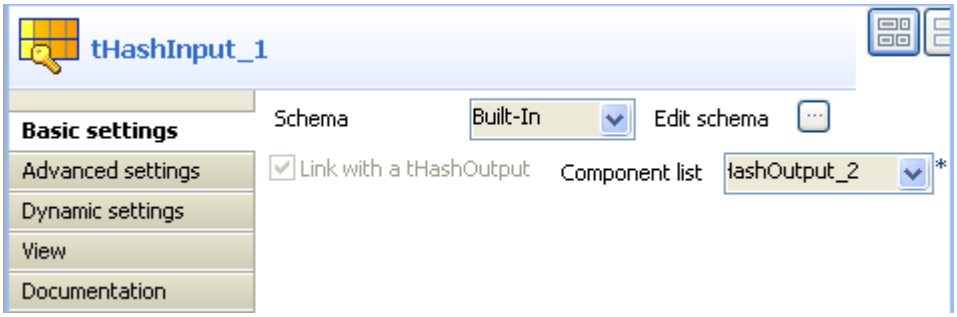
9. Double-click **tHashOutput** to display its **Basic settings** view.



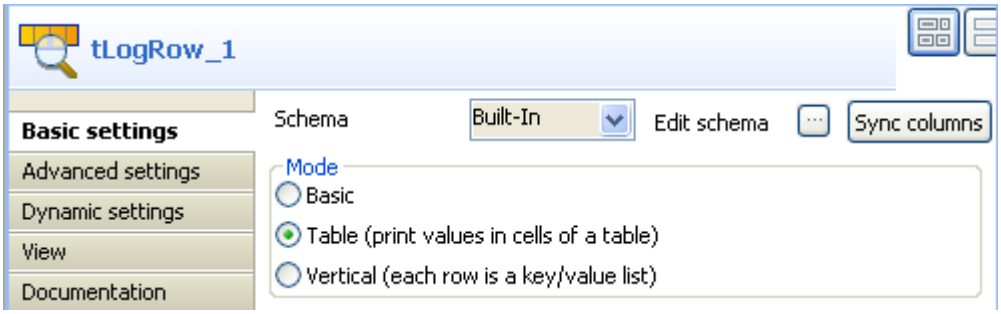
10. Select **Built-In** from the **Schema** drop-down list and click **Sync columns** to retrieve the schema from the previous component. Select **Keep all** from the **Keys management** drop-down list and deselect the **Append** check box.

Configuring data retrieval from hash cache and data output

1. Double-click **tHashInput** to display its **Basic settings** view.



2. Select **Built-In** from the **Schema** drop-down list. Click **Edit schema** to define the data structure, which is the same as that of **tHashOutput**.
3. Select **tHashOutput_2** from the **Component list** drop-down list.
4. Double-click **tLogRow** to display its **Basic settings** view.

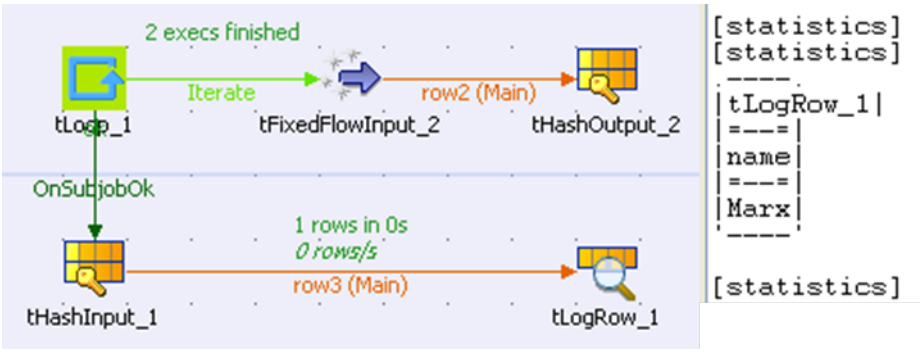


5. Select **Built-In** from the **Schema** drop-down list and click **Sync columns** to retrieve the schema from the previous component. In the **Mode** area, select **Table (print values in cells of a table)**.

Saving and executing the Job

1. Press **Ctrl+S** to save the Job.
2. Press **F6**, or click **Run** on the **Run** tab to execute the Job.

   You can find that only one row was output although two rows were generated by **tFixedFlowInput**.



tHashOutput

tHashOutput Properties

This component writes data to the cache memory and is closely related to **tHashInput**. Together, these twin components offer high-speed data access to facilitate transactions involving a massive amount of data.

Component family	Technical	
Function	**tHashOutput** writes data to the cache memory for high-speed access.	
Purpose	This component loads data to the cache memory to offer high-speed access, facilitating transactions involving a large amount of data.	
Basic settings	*Schema* and *Edit schema*	A schema is a row description, i.e. it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema is created and stored locally for this component only. Related topic: see the *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: see the *Talend Open Studio for Big Data User Guide*.
	Link with a tHashOutput	Select this check box to connect to a **tHashOutput** component. If multiple **tHashOutput** components are linked in this way, the data loaded to the cache by all of them can be read by a **tHashInput** component that is linked with any of them.
	Component list	Drop-down list of available **tHashOutput** components.
	Data write model	Drop-down list of available data write modes.
	Keys management	Drop-down list of available keys management modes.
	Append	Selected by default, this option is designed to append data to the memory in case an iterator exists in the same subjob. If it is unchecked, **tHashOutput** will clear the memory before loading data to it. If **Link with a tHashOutput** is selected, this check box will be hidden but is always enabled.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component writes data to the cache memory and is closely related to **tHashInput**. Together, these twin components offer high-speed data access to facilitate transactions involving a massive amount of data.	
Limitation	n/a	

Related scenarios

For related scenarios, see:

- section *Scenario 1: Reading data from the cache memory for high-speed data access*.
- section *Scenario 2: Clearing the memory before loading data to it in case an iterator exists in the same subjob*.

XML components

This chapter details the main components that you can find in the **XML** family of the **Palette** in the **Integration** perspective of the **Talend** Studio.

The XML family groups together the components dedicated to XML related tasks such as parsing, validation, XML structure creation and so on.

*For Talend Open Studio for Big Data, the **Property type**, **Schema** and **Query Type** of components are always **Built-in**. For how to edit a Built-in schema, see Talend Open Studio for Big Data User Guide.*

tAdvancedFileOutputXML

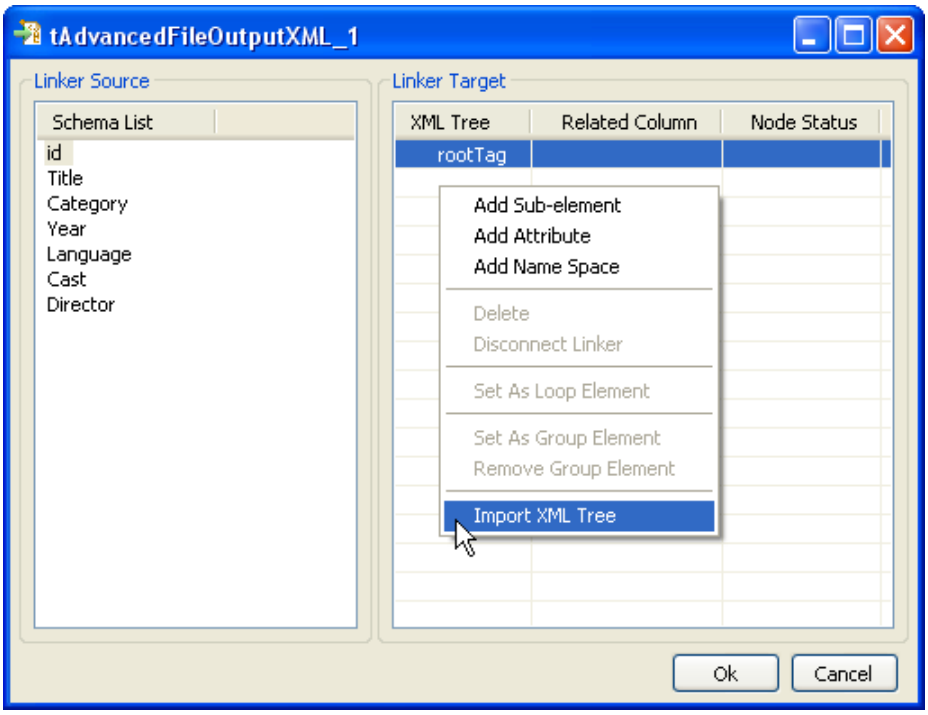
tAdvancedFileOutputXML properties

Component family	XML or File/Output	
Function	**tAdvancedFileOutputXML** outputs data to an XML type of file and offers an interface to deal with loop and group by elements if needed.	
Purpose	**tAdvancedFileOutputXML** writes an XML file with separated data values according to an XML tree structure.	
Basic settings	*Use Output Stream*	Select this check box process the data flow of interest. Once you have selected it, the **Output Stream** field displays and you can type in the data flow of interest. The data flow to be processed must be added to the flow in order for this component to fetch these data via the corresponding representative variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl +Space**) to fill the current field on condition that this variable has been properly defined. For further information about how to use a stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	File name	Name or path to the output file and/or the variable to be used. This field becomes unavailable once you have selected the **Use Output Stream** check box. Related topic: see *Talend Open Studio for Big Data* **User Guide**
	Configure XML tree	Opens the dedicated interface to help you set the XML mapping. For details about the interface, see section *Defining the XML tree*.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: see *Talend Open Studio for Big Data* **User Guide**.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
	Append the source xml file	Select this check box to add the new lines at the end of your source XML file.

	Generate compact file	Select this check box to generate a file that does not have any empty space or line separators. All elements then are presented in a unique line and this will reduce considerably file size.
	Include DTD or XSL	Select this check box to to add the DOCTYPE declaration, indicating the root element, the access path and the DTD file, or to add the processing instruction, indicating the type of stylesheet used (such as XSL types), along with the access path and file name.
Advanced settings	*Split output in several files*	If the XML file output is big, you can split the file every certain number of rows.
	Trim data	This check box is activated when you are using the dom4j generation mode. Select this check box to trim the leading or trailing whitespace from the value of a XML element.
	Create directory only if not exists	This check box is selected by default. It creates a directory to hold the output XML files if required.
	Create empty element if needed	This box is selected by default. If no column is associated to an XML node, this option will create an open/close tag in place of the expected tag.
	Create attribute even if its value is NULL	Select this check box to generate XML tag attribute for the associated input column whose value is null.
	Create attribute even if it is unmapped	Select this check box to generate XML tag attribute for the associated input column that is unmapped.
	Create associated XSD file	If one of the XML elements is defined as a Namespace element, this option will create the corresponding XSD file. To use this option, you must select **Dom4J** as the generation mode.
	Advanced separator (for number)	Select this check box to change the expected data separator. **Thousands separator:** define the thousands separator, between inverted commas **Decimal separator:** define the decimals separator between inverted commas
	Generation mode	Select the appropriate generation mode according to your memory availability. The available modes are: • **Slow and memory-consuming (Dom4j)** This option allows you to use dom4j to process the XML files of high complexity. • **Fast with low memory consumption** Once you select **Append the source xml file** in the **Basic settings** view, this field disappears because in this situation, your generation mode is set automatically as dom4j.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Don’t generate empty file	Select the check box to avoid the generation of an empty file.
	tStatCatcher Statistics	Select the check box to collect the log data at a Job level as well as at each component level.
Usage	Use this component to write an XML file with data passed on from other components using a Row link.	
Limitation	n/a	

Defining the XML tree

Double-click on the **tAdvancedFileOutputXML** component to open the dedicated interface or click on the three-dot button on the **Basic settings** vertical tab of the **Component Settings** tab.



To the left of the mapping interface, under **Schema List**, all of the columns retrieved from the incoming data flow are listed (on the condition that an input flow is connected to the **tAdvancedFileOutputXML** component).

To the right of the interface, define the XML structure you want to obtain as output.

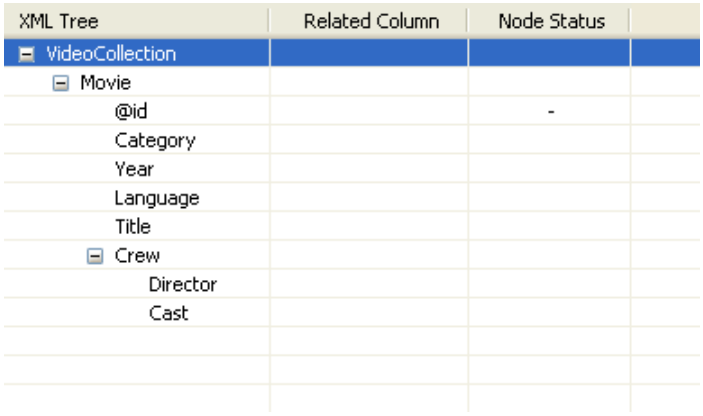
You can easily import the XML structure or create it manually, then map the input schema columns onto each corresponding element of the XML tree.

Importing the XML tree

The easiest and most common way to fill out the XML tree panel, is to import a well-formed XML file.

1. Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
2. Right-click on the root tag to display the contextual menu.
3. On the menu, select **Import XML tree**.
4. Browse to the file to import and click **OK**.

- You can import an XML tree from files in XML, XSD and DTD formats.
- When importing an XML tree structure from an XSD file, you can choose an element as the root of your XML tree.



The **XML Tree** column is hence automatically filled out with the correct elements. You can remove and insert elements or sub-elements from and to the tree:

1. Select the relevant element of the tree.
2. Right-click to display the contextual menu
3. Select **Delete** to remove the selection from the tree or select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** to enrich the tree.

Creating the XML tree manually

If you don’t have any XML structure defined as yet, you can create it manually.

1. Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
2. Right-click on the root tag to display the contextual menu.
3. On the menu, select Add sub-element to create the first element of the structure.

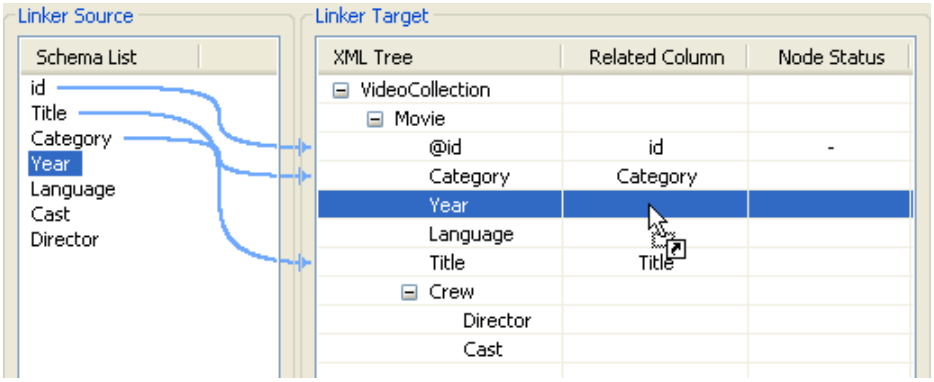
You can also add an attribute or a child element to any element of the tree or remove any element from the tree.

1. Select the relevant element on the tree you just created.
2. Right-click to the left of the element name to display the contextual menu.
3. On the menu, select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** or **Delete**.

Mapping XML data

Once your XML tree is ready, you can map each input column with the relevant XML tree element or sub-element to fill out the **Related Column**:

1. Click on one of the **Schema column name**.
2. Drag it onto the relevant sub-element to the right.
3. Release to implement the actual mapping.



A light blue link displays that illustrates this mapping. If available, use the **Auto-Map** button, located to the bottom left of the interface, to carry out this operation automatically.

You can disconnect any mapping on any element of the XML tree:

1. Select the element of the XML tree, that should be disconnected from its respective schema column.
2. Right-click to the left of the element name to display the contextual menu.
3. Select **Disconnect linker**.

The light blue link disappears.

Defining the node status

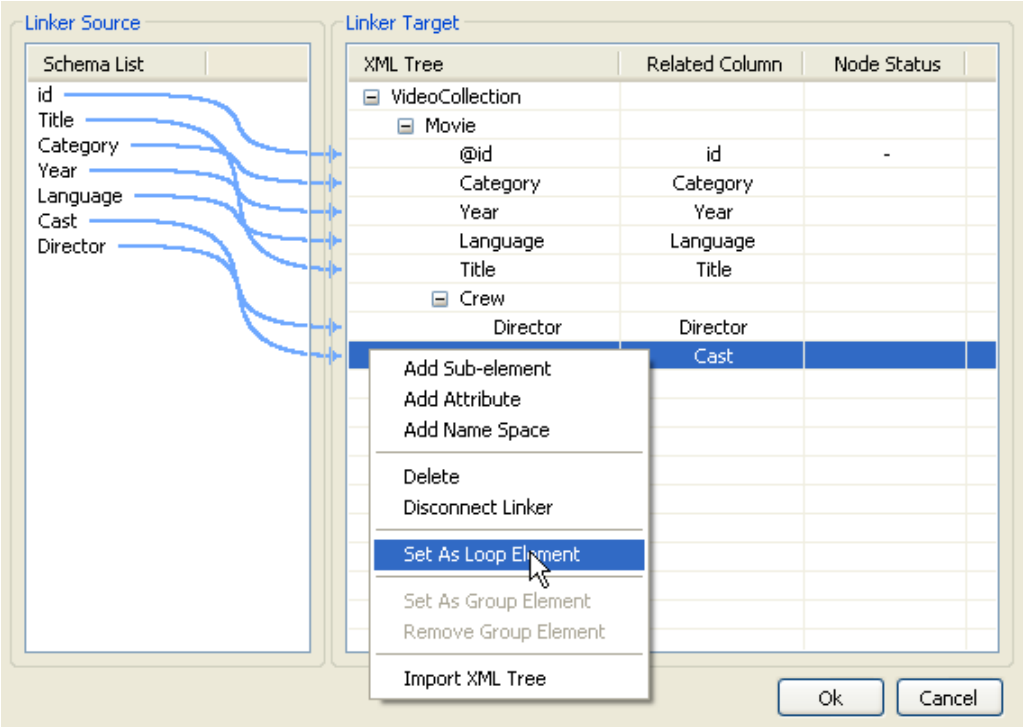
Defining the XML tree and mapping the data is not sufficient. You also need to define the loop element and if required the group element.

Loop element

The loop element allows you to define the iterating object. Generally the Loop element is also the row generator.

To define an element as loop element:

1. Select the relevant element on the XML tree.
2. Right-click to the left of the element name to display the contextual menu.
3. Select **Set as Loop Element**.



The **Node Status** column shows the newly added status.

There can only be one loop element at a time.

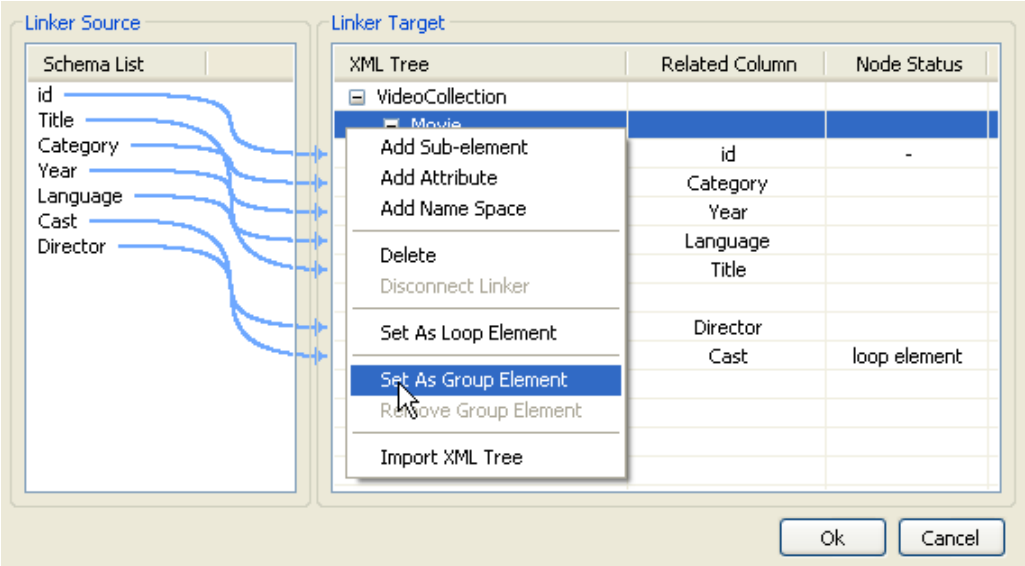
Group element

The group element is optional, it represents a constant element where the groupby operation can be performed. A group element can be defined on the condition that a loop element was defined before.

When using a group element, the rows should sorted, in order to be able to group by the selected node.

To define an element as group element:

1. Select the relevant element on the XML tree.
2. Right-click to the left of the element name to display the contextual menu.
3. Select **Set as Group Element**.



The **Node Status** column shows the newly added status and any group status required are automatically defined, if needed.

Click **OK** once the mapping is complete to validate the definition and continue the job configuration where needed.

Scenario: Creating an XML file using a loop

The following scenario describes the creation of an XML file from a sorted flat file gathering a video collection.

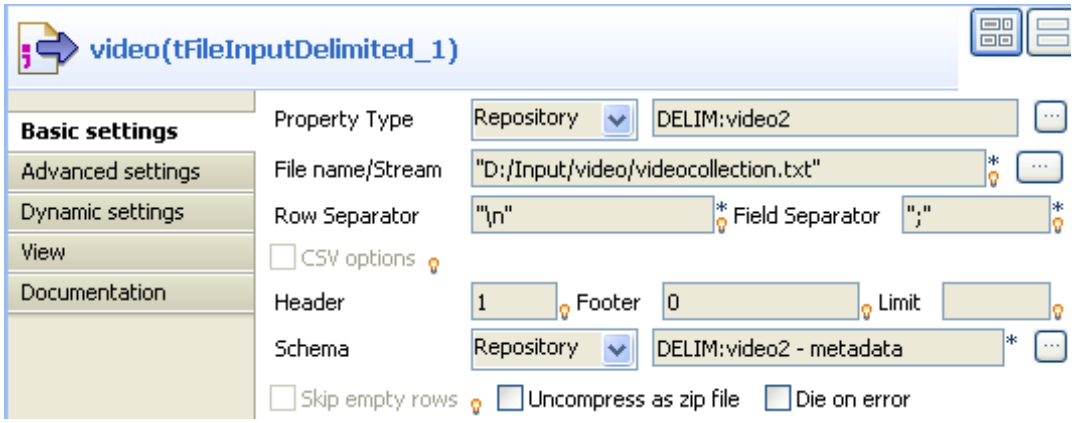
Configuring the source file

1. Drop a **tFileInputDelimited** and a **tAdvancedFileOutputXML** from the **Palette** onto the design workspace.

2. Alternatively, if you configured a description for the input delimited file in the **Metadata** area of the **Repository**, then you can directly drag & drop the metadata entry onto the editor, to set up automatically the input flow.

3. Right-click on the input component and drag a row main link towards the **tAdvancedFileOutputXML** component to implement a connection.

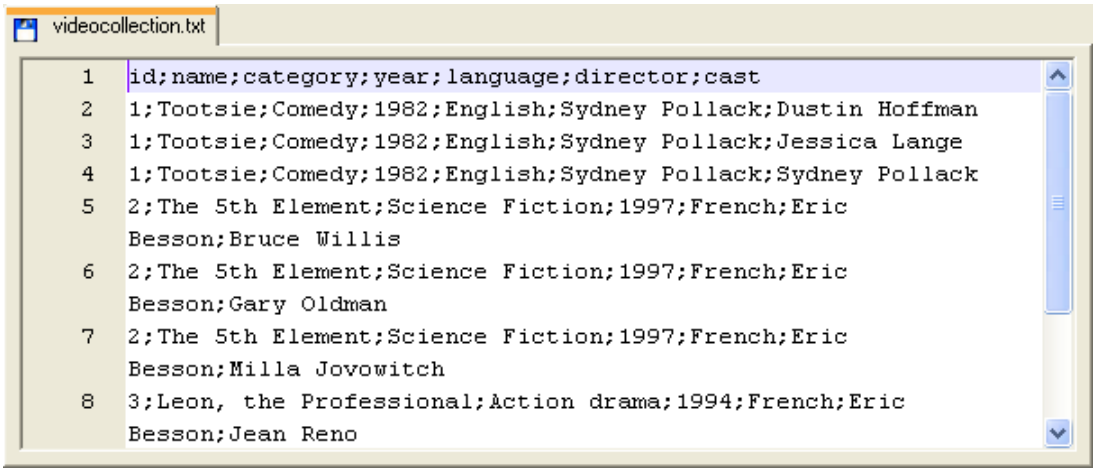
4. Select the **tFileInputDelimited** component and display the **Component settings** tab located in the tab system at the bottom of the Studio.



5. Select the **Property type**, according to whether you stored the file description in the Repository or not. If you dragged & dropped the component directly from the Metadata, no changes to the setting should be needed.

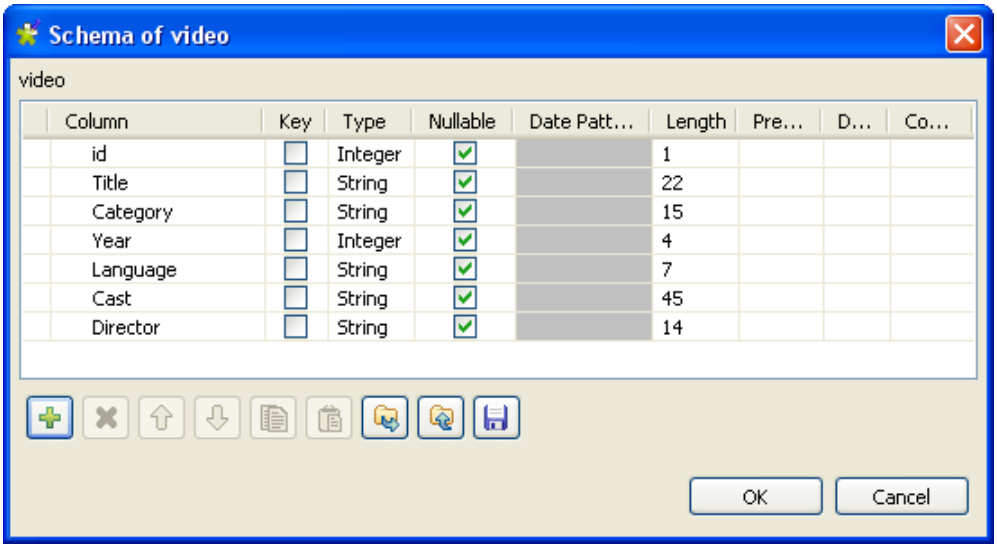
6. If you didn't setup the file description in the **Repository**, then select **Built-in** and manually fill out the fields displayed on the **Basic settings** vertical tab.

   The input file contains the following type of columns separated by semi-colons: *id*, *name*, *category*, *year*, *language*, *director* and *cast*.



   In this simple use case, the **Cast** field gathers different values and the id increments when changing movie.

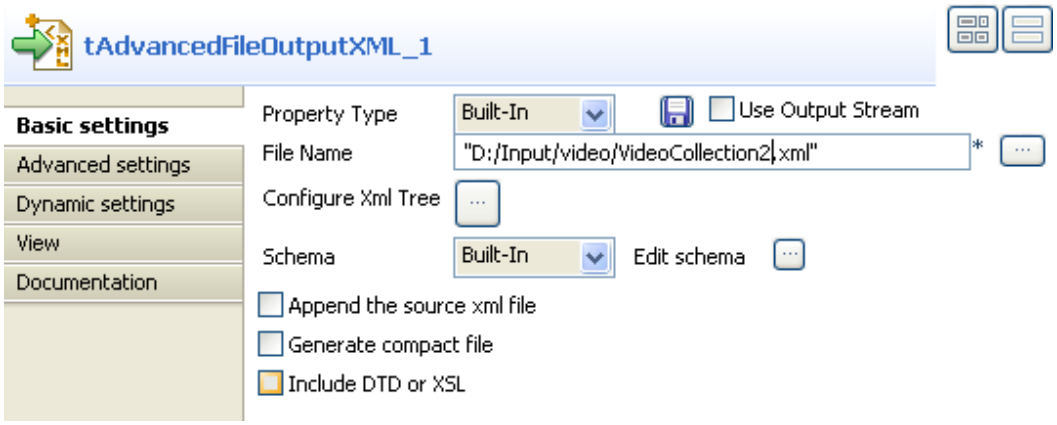
7. If needed, define the **tFileDelimitedInput** schema according to the file structure.



8. Once you checked that the schema of the input file meets your expectation, click on **OK** to validate.

Configuring the XML output and mapping

1. Then select the **tAdvancedFileOutputXML** component and click on the **Component settings** tab to configure the basic settings as well as the mapping. Note that a double-click on the component will open directly the mapping interface.



2. In the **File Name** field, browse to the file to be written if it exists or type in the path and file name that needs to be created for the output.

   By default, the schema (file description) is automatically propagated from the input flow. But you can edit it if you need.

3. Then click on the three-dot button or double-click on the **tAdvancedFileOutputXML** component on the design workspace to open the dedicated mapping editor.

   To the left of the interface, are listed the columns from the input file description.

4. To the right of the interface, set the XML tree panel to reflect the expected XML structure output.

   You can create the structure node by node. For more information about the manual creation of an XML tree, see section *Defining the XML tree*.

   In this example, an XML template is used to populate the XML tree automatically.

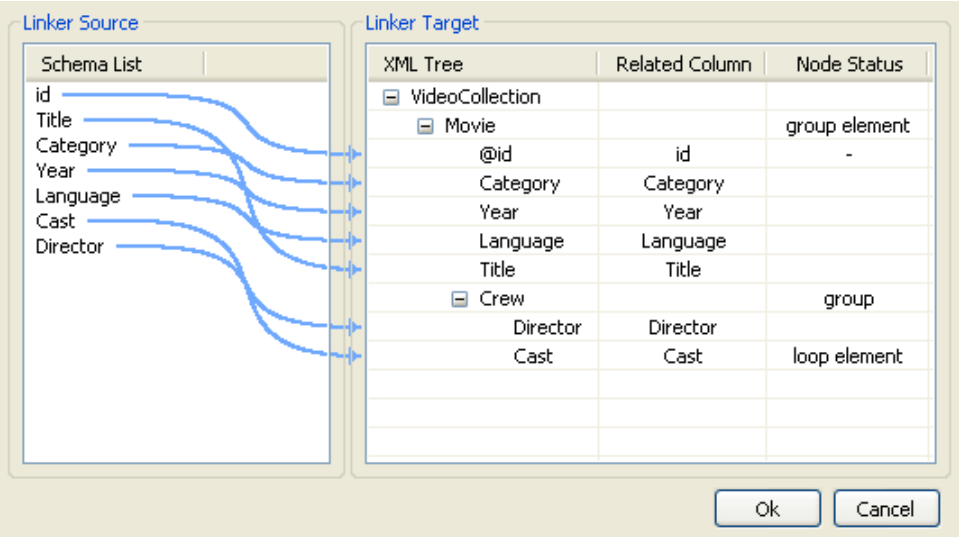
5. Right-click on the **root tag** displaying by default and select **Import XML tree** at the end of the contextual menu options.

6. Browse to the XML file to be imported and click **OK** to validate the import operation.

   You can import the XML structure from XSML, XSD and STS files.

7. Then drag & drop each column name from the **Schema List** to the matching (or relevant) **XML tree** elements as described in section *Mapping XML data*.

   The mapping is shown as blue links between the left and right panels.



Finally, define the node status where the loop should take place. In this use case, the *Cast* being the changing element on which the iteration should operate, this element will be the loop element.

Right-click on the Cast element on the XML tree, and select **Set as loop element**.

8. To group by movie, this use case needs also a group element to be defined.

   Right-click on the Movie parent node of the XML tree, and select **Set as group element**.

   The newly defined node status show on the corresponding element lines.

9. Click **OK** to validate the configuration.

10. Press **F6** to execute the Job.

VideoCollection2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-15"?>
2
3  <VideoCollection>
4    <Movie id="1">
5      <Category>Comedy</Category>
6      <Year>1982</Year>
7      <Language>English</Language>
8      <Title>Tootsie</Title>
9      <Crew>
10       <Director>Sydney Pollack</Director>
11       <Cast>Dustin Hoffman</Cast>
12       <Cast>Jessica Lange</Cast>
13       <Cast>Sydney Pollack</Cast>
14     </Crew>
15   </Movie>
16   <Movie id="2">
17     <Category>Science Fiction</Category>
18     <Year>1997</Year>
19     <Language>French</Language>
20     <Title>The 5th Element</Title>
21     <Crew>
22       <Director>Eric Besson</Director>
23       <Cast>Bruce Willis</Cast>
```

The output XML file shows the structure as defined.

tDTDValidator

tDTDValidator Properties

Component family	XML	
Function	Validates the XML input file against a DTD file and sends the validation log to the defined output.	
Purpose	Helps at controlling data and structure quality of the file to be processed	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but in this case, the schema is read-only. It contains standard information regarding the file validation.
	DTD file	Filepath to the reference DTD file.
	XML file	Filepath to the XML file to be validated.
	If XML is valid, display If XML is not valid detected, display	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the validation message.
Usage	This component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Limitation	n/a	

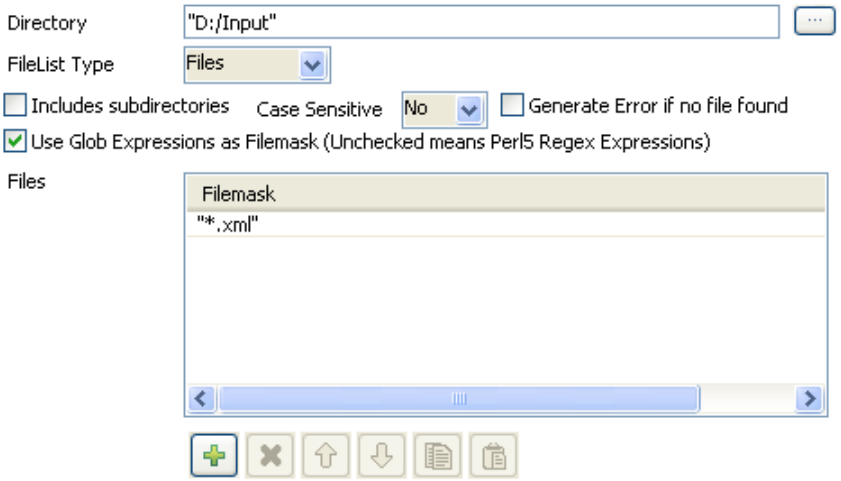
Scenario: Validating XML files

This scenario describes a Job that validates the specified type of files from a folder, displays the validation result on the Run tab console, and outputs the log information for the invalid files into a delimited file.

1. Drop the following components from the **Palette** to the design workspace: **tFileList**, **tDTDValidator**, **tMap**, **tFileOutputDelimited**.

2. Connect the **tFileList** to the **tDTDValidator** with an **Iterate** link and the remaining component using a **main** row.

3. Set the **tFileList** component properties, to fetch an XML file from a folder.

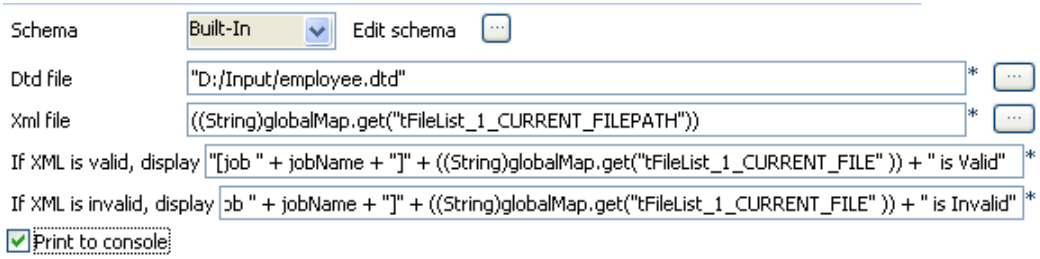


Click the plus button to add a filemask line and enter the filemask: *.xml. Remember Java code requires double quotes.

Set the path of the XML files to be verified.

Select **No** from the Case Sensitive drop-down list.

4. In the **tDTDValidate Component** view, the schema is read-only as it contains standard log information related to the validation process.



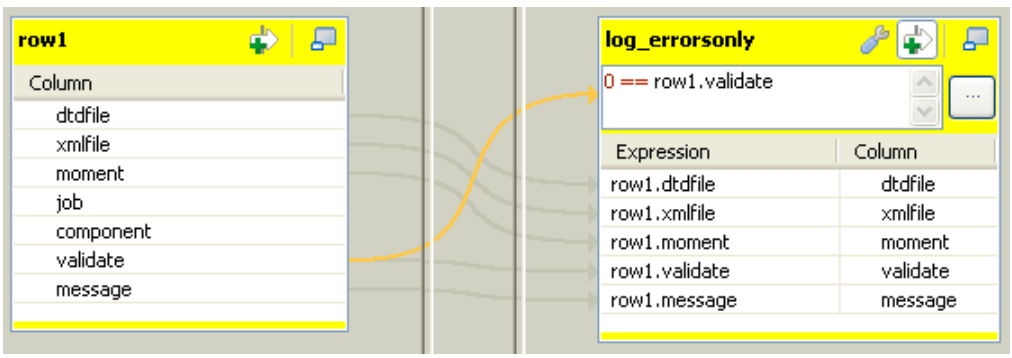
In the **Dtd file** field, browse to the DTD file to be used as reference.

5. Click in the XML file field, press **Ctrl+Space bar** to access the variable list, and double-click the current filepath global variable: `tFileList.CURRENT_FILEPATH`.

6. In the various messages to display in the **Run** tab console, use the *jobName* variable to recall the job name tag. Recall the filename using the relevant global variable: `((String)globalMap.get("tFileList_1_CURRENT_FILE"))`. Remember Java code requires double quotes.

Select the **Print to Console** check box.

7. In the **tMap** component, drag and drop the information data from the standard schema that you want to pass on to the output file.



8. Once the Output schema is defined as required, add a filter condition to only select the log information data when the XML file is invalid.

   Follow the best practice by typing first the wanted value for the variable, then the operator based on the type of data filtered then the variable that should meet the requirement. In this case: `0 == row1.validate`.

9. Then connect (if not already done) the **tMap** to the **tFileOutputDelimited** component using a **Row** > **Main** connection. Name it as relevant, in this example: *log_errorsOnly*.

10. In the **tFileOutputDelimited Basic settings**, Define the destination filepath, the field delimiters and the encoding.

11. Save your Job and press **F6** to run it.

```
Starting job tDTDValidator_oj at 14:28 14/10/2010.

[statistics] connecting to socket on port 3420
[job tDTDValidator_oj]employee_1.xml is Valid
[job tDTDValidator_oj]employee_2.xml is Invalid
[job tDTDValidator_oj]employee_3.xml is Valid
[job tDTDValidator_oj]employee_4.xml is Invalid
[job tDTDValidator_oj]employee_5.xml is Invalid
[statistics] disconnected
Job tDTDValidator_oj ended at 14:28 14/10/2010. [exit
code=0]
```

   On the **Run** console the messages defined display for each of the files. At the same time the output file is filled with the log data for invalid files.

tExtractXMLField

tExtractXMLField properties

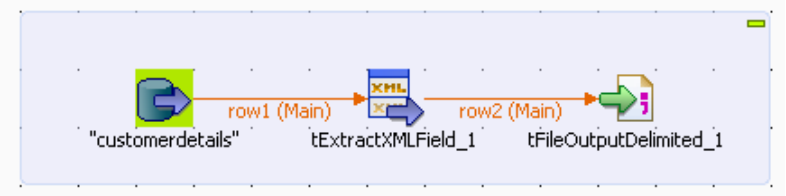
Component family	XML	
Function	**tExtractXMLField** reads an input XML field of a file or a database table and extracts desired data.	
Purpose	**tExtractXMLField** opens an input XML field, reads the XML structured data directly without having first to write it out to a temporary file, and finally sends data as defined in the schema to the following component via a **Row** link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data is stored centrally.
		Repository: Properties are stored in a repository file. When this file is selected, the fields that follow are pre-filled in using fetched data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* User Guide
		Repository: You already created the schema and stored it in the Repository, hence can be reused in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* User Guide.
	XML field	Name of the XML field to be processed. Related topic: see *Talend Open Studio for Big Data* User Guide
	Loop XPath query	Node of the XML tree, which the loop is based on.
	Mapping	**Column**: reflects the schema as defined by the Schema type field. **XPath Query**: Enter the fields to be extracted from the structured input. **Get nodes**: Select this check box to recuperate the XML content of all current nodes specified in the **Xpath query** list or select the check box next to specific XML nodes to recuperate only the content of the selected nodes.
	Limit	Maximum number of rows to be processed. If Limit is 0, no rows are read or processed.
	Die on error	This check box is selected by default. Clear the check box to skip the row on error and complete the process for error-free rows. If needed, you can retrieve the rows on error via a **Row** > **Reject** link.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component is an intermediate component. It needs an input and an output components.	
Limitation	n/a	

Scenario 1: Extracting XML data from a field in a database table

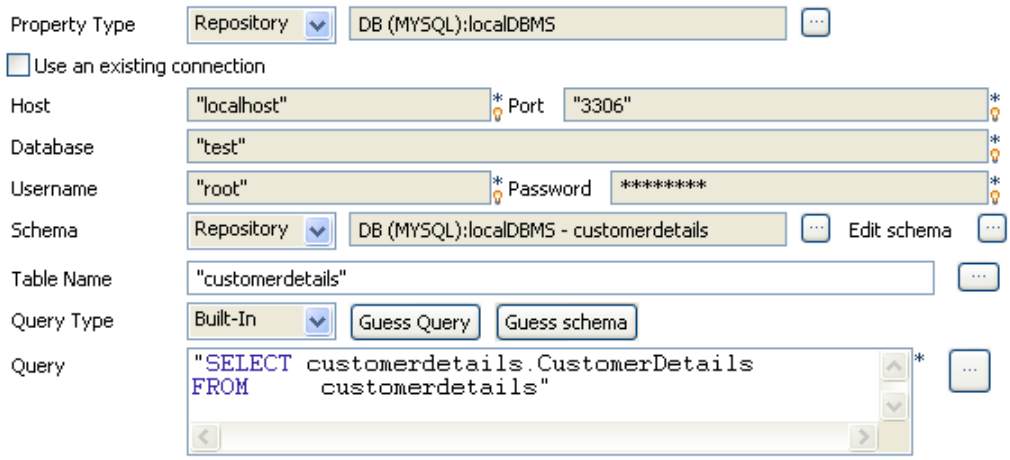
This three-component Java scenario allows to read the XML structure included in the fields of a database table and then extracts the data.

1. Drop the following components from the **Palette** onto the design workspace: **tMysqlInput**, **tExtractXMLField**, and **tFileOutputDelimited**.

   Connect the three components using **Main** links.



2. Double-click **tMysqlInput** to display its **Basic settings** view and define its properties.



3. If you have already stored the input schema in the **Repository** tree view, select **Repository** first from the **Property Type** list and then from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.

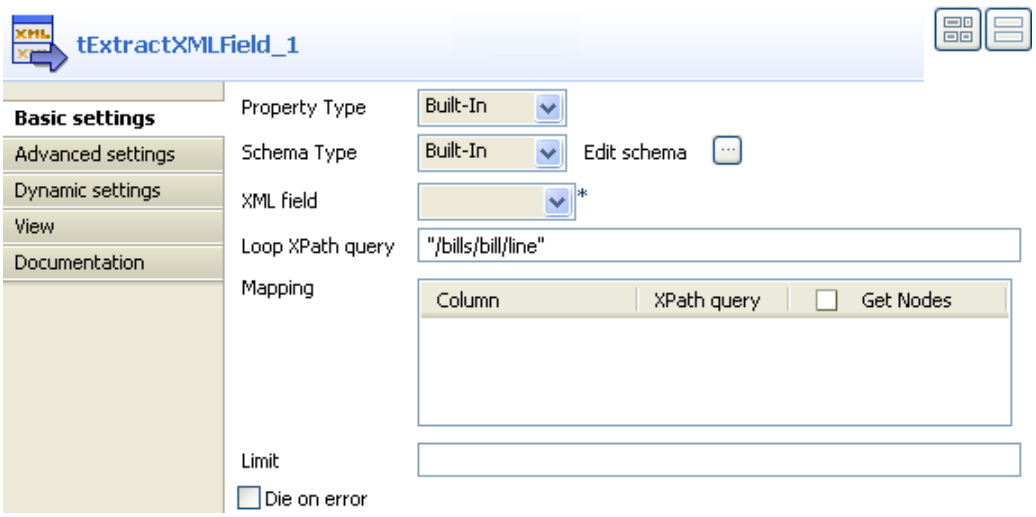
   For more information about storing schema metadata in the **Repository** tree view, see *Talend Open Studio for Big Data* **User Guide**.

4. If you have not stored the input schema locally, select **Built-in** in the **Property Type** and **Schema** fields and enter the database connection and the data structure information manually. For more information about **tMysqlInput** properties, see section *tMysqlInput*.

5. In the **Table Name** field, enter the name of the table holding the XML data, *customerdetails* in this example.

   Click **Guess Query** to display the query corresponding to your schema.

6. Double-click **tExtractXMLField** to display its **Basic settings** view and define its properties.



7. In the **Property type** list, select **Repository** if you have already stored the description of your file in the **Repository** tree view. The fields that follow are filled in automatically with the stored data.

   If not, select **Built-in** and fill in the fields that follow manually.

8. Click **Sync columns** to retrieve the schema from the preceding component. You can click the three-dot button next to **Edit schema** to view/modify the schema.

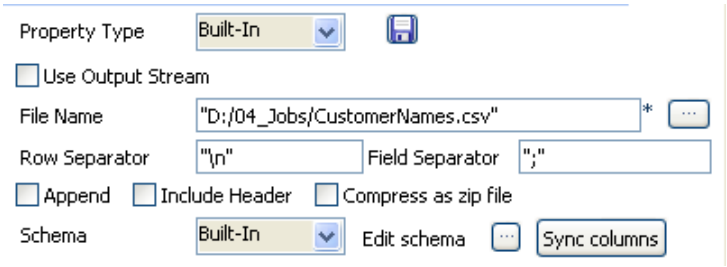
   Column in the **Mapping** table will be automatically populated with the defined schema.

9. In the **Xml field** list, select the column from which you want to extract the XML data. In this example, the filed holding the XML data is called *CustomerDetails*.

   In the **Loop XPath query** field, enter the node of the XML tree on which to loop to retrieve data.

   In the **Xpath query** column, enter between inverted commas the node of the XML field holding the data you want to extract, *CustomerName* in this example.

10. Double-click **tFileOutputDelimited** to display its **Basic settings** view and define its properties.



11. In the **File Name** field, define or browse to the path of the output file you want to write the extracted data in.

    Click **Sync columns** to retrieve the schema from the preceding component. If needed, click the three-dot button next to **Edit schema** to view the schema.

12. Save your Job and click **F6** to execute it.

```
1  Griffith Paving and Sealcoatin
2  Bill's Dive Shop
3  Childress Child Day Care
4  Facelift Kitchen and Bath
5  Terrinni & Son Auto and Truck
6  Kermit the Pet Shop
7  Tub's Furniture Store
8  Toggle & Myerson Ltd
9  Childress Child Day Care
10 Elle Hypnosis and Therapy Cent
11 Lennox Air Pollution Control
```

tExtractXMLField read and extracted the clients names under the node `CustomerName` of the *CustomerDetails* field of the defined database table.

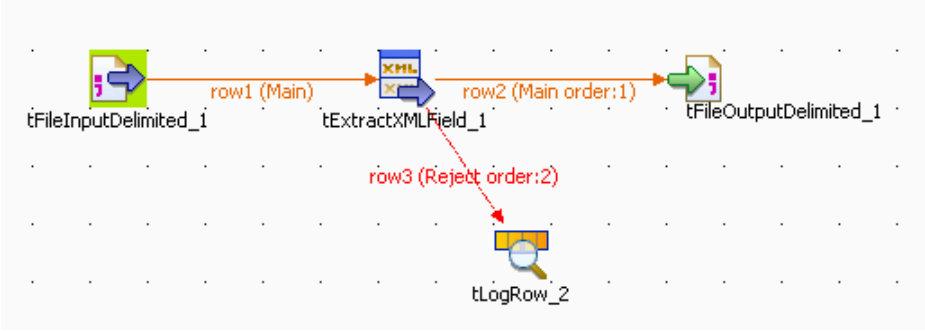
Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file

This Java scenario describes a four-component Job that reads an XML structure from a delimited file, outputs the main data and rejects the erroneous data.

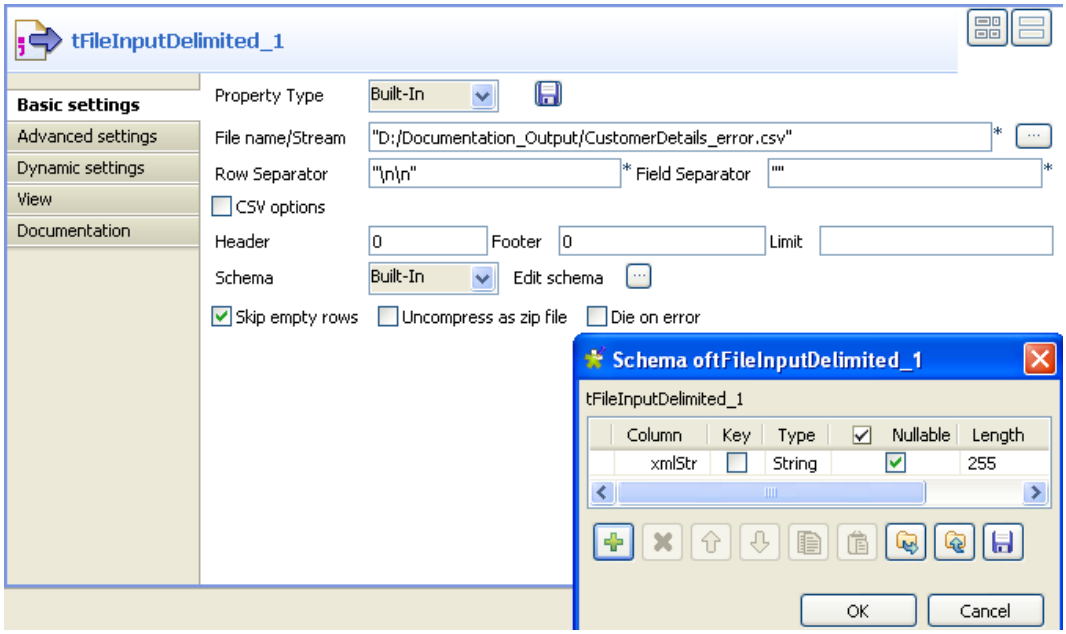
1. Drop the following components from the **Palette** to the design workspace: **tFileInputDelimited**, **tExtractXMLField**, **tFileOutputDelimited** and **tLogRow**.

   Connect the first three components using **Row Main** links.

   Connect **tExtractXMLField** to **tLogRow** using a **Row Reject** link.



2. Double-click **tFileInputDelimited** to open its **Basic settings** view and define the component properties.



3. Select **Built-in** in the **Schema** list and fill in the file metadata manually in the corresponding fields.

   Click the three-dot button next to **Edit schema** to display a dialog box where you can define the structure of your data.

   Click the plus button to add as many columns as needed to your data structure. In this example, we have one column in the schema: *xmlStr*.

   Click **OK** to validate your changes and close the dialog box.

   If you have already stored the schema in the **Metadata** folder under **File delimited**, select **Repository** from the **Schema** list and click the three-dot button next to the field to display the **[Repository Content]** dialog box where you can select the relevant schema from the list. Click **Ok** to close the dialog box and have the fields automatically filled in with the schema metadata.

   For more information about storing schema metadata in the Repository tree view, see *Talend Open Studio for Big Data* User Guide.

4. In the **Property type** list, select:

   -**Repository** if you have already stored the metadata of your input file in the Repository, the fields that follow are automatically filled in with the stored information, or

   -select **Built-in** and fill in the fields that follow manually.

   For this example, we use the **Built-in** mode.

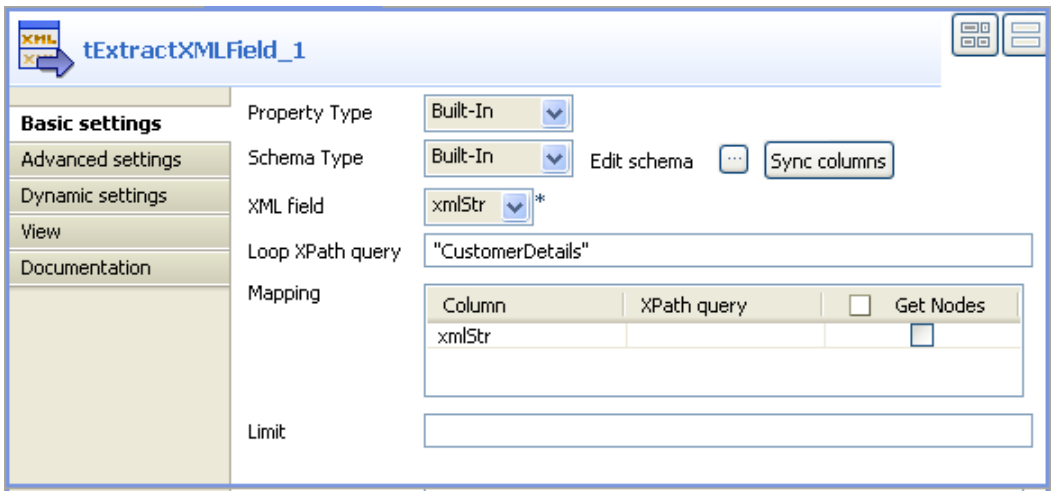
5. In the **File Name** field, click the three-dot button and browse to the input delimited file you want to process, *CustomerDetails_Error* in this example.

   This delimited file holds a number of simple XML lines separated by double carriage return.

   Set the row and field separators used in the input file in the corresponding fields, double carriage return for the first and nothing for the second in this example.

   If needed, set **Header**, **Footer** and **Limit**. None is used in this example.

6. In the design workspace, double-click **tExtractXMLField** to display its **Basic settings** view and define the component properties.



7. In the **Property type** list, select:

   -**Repository** if you have already stored the metadata of your file in the Repository, the fields that follow are automatically filled in with the stored information, or

   -select **Built-in** and fill in the fields that follow manually.

   For this example, we use the **Built-in** mode.

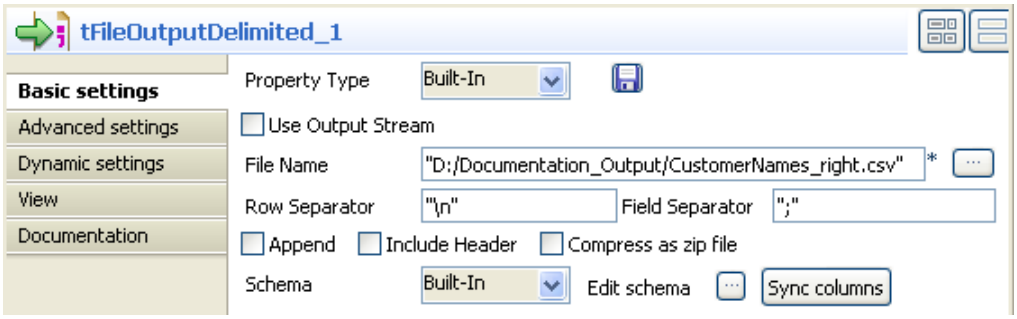
8. Click **Sync columns** to retrieve the schema from the preceding component. You can click the three-dot button next to **Edit schema** to view/modify the schema.

   Column in the **Mapping** table will be automatically populated with the defined schema.

9. In the **Xml field** list, select the column from which you want to extract the XML data. In this example, the filed holding the XML data is called *xmlStr*.

   In the **Loop XPath query** field, enter the node of the XML tree on which to loop to retrieve data.

10. In the design workspace, double-click **tFileOutputDelimited** to open its **Basic settings** view and display the component properties.



11. Set **Property Type** to **Built-in**.

    In the **File Name** field, define or browse to the output file you want to write the correct data in, *CustomerNames_right.csv* in this example.

    Click **Sync columns** to retrieve the schema of the preceding component. You can click the three-dot button next to **Edit schema** to view/modify the schema.

12. In the design workspace, double-click **tLogRow** to display its **Basic settings** view and define the component properties.

Click **Sync Columns** to retrieve the schema of the preceding component. For more information on this component, see section *tLogRow*.

13. Save your Job and press **F6** to execute it.

```
Starting job A at 14:22 05/11/2009.

[statistics] connecting to socket on port 3845
[statistics] connected
|
<CustomerDetails>
  <CustomerName>Childress Child Day Care</Customername>
  <id>9</id>
</CustomerDetails>||Error on line 3 of document  : The element type
"CustomerName" must be terminated by the matching end-tag "</CustomerName>".
Nested exception: The element type "CustomerName" must be terminated by the
matching end-tag "</CustomerName>". - Line: 8
[statistics] disconnected
Job A ended at 14:22 05/11/2009. [exit code=0]
```

tExtractXMLField reads and extracts in the output delimited file, *CustomerNames_right*, the client information for which the XML structure is correct, and displays as well erroneous data on the console of the **Run** view.

tFileInputXML

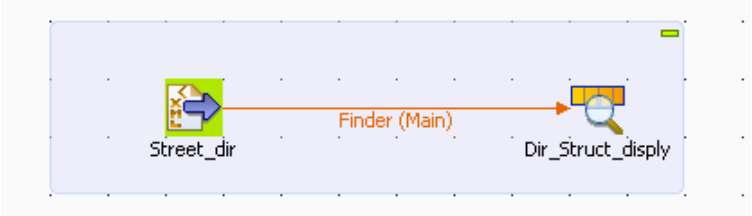
tFileInputXML Properties

Component family	XML or File/Input	
Function	**tFileInputXML** reads an XML structured file and extracts data row by row.	
Purpose	Opens an XML structured file and reads it row by row to split them up into fields then sends fields as defined in the Schema to the next component, via a **Row** link.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data* User Guide
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* User Guide.
	File Name/Stream	**File name**: Name and path of the file to be processed. **Stream:** The data flow to be processed. The data must be added to the flow in order for **tFileInputXML** to fetch these data via the corresponding representative variable. This variable could be already pre-defined in your Studio or provided by the context or the components you are using along with this component, for example, the *INPUT_STREAM* variable of **tFileFetch**; otherwise, you could define it manually and use it according to the design of your Job, for example, using **tJava** or **tJavaFlex**. In order to avoid the inconvenience of hand writing, you could select the variable of interest from the auto-completion list (**Ctrl +Space**) to fill the current field on condition that this variable has been properly defined. Related topic to the available variables: see *Talend Open Studio for Big Data* **User Guide**. Related scenario to the input stream, see section *Scenario 2: Reading data from a remote file in streaming mode*.
	Loop XPath query	Node of the tree, which the loop is based on.
	Mapping	**Column:** Columns to map. They reflect the schema as defined in the Schema type field. **XPath Query**: Enter the fields to be extracted from the structured input. **Get nodes**: Select this check box to recuperate the XML content of all current nodes specified in the **Xpath query** list, or select the check box next to specific XML nodes to recuperate only the content of the selected nodes. These nodes are important when the

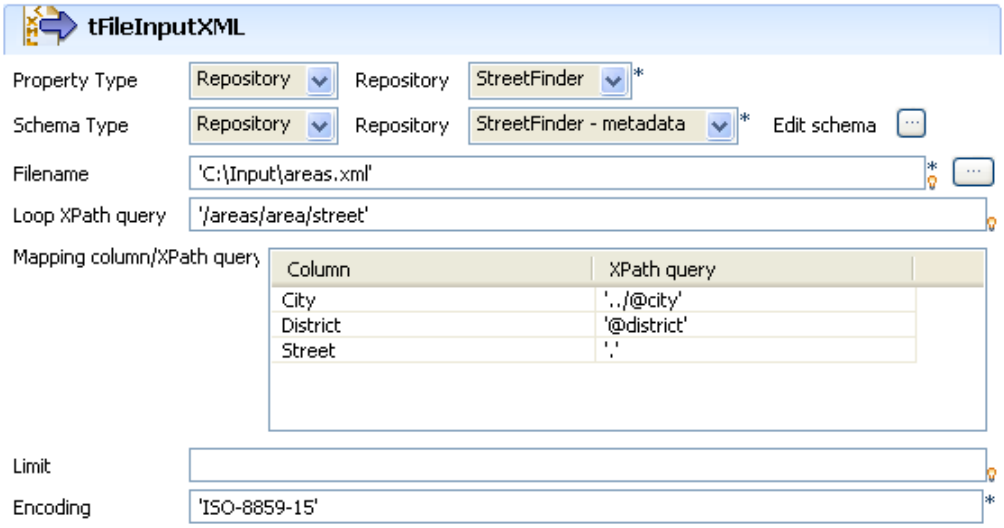
		output flow from this component needs to use the XML structure, for example, the **Document** data type. For further information about the Document type, see *Talend Open Studio for Big Data* **User Guide**. The **Get Nodes** option functions in the **DOM4j** and **SAX** modes, although in **SAX** mode namespaces are not supported. For further information concerning the **DOM4j** and **SAX** modes, please see the properties noted in the **Generation mode** list of the **Advanced Settings** tab..
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read nor processed. If -1, all rows are read or processed.
	Die on error	This check box is selected by default and stops the job in the event of error. Clear the check box to skip erroneous rows. The process will still be completed for error-free rows. If needed, you can retrieve the erroneous rows using a **Row** > **Reject** link.
Advanced settings	*Ignore DTD file*	Select this check box to ignore the DTD file indicated in the XML file being processed.
	Advanced separator (for number)	Select this check box to change data separator for numbers: **Thousands separator**: define the separators to use for thousands. **Decimal separator**: define the separators to use for decimals.
	Ignore the namespaces	Select this check box to ignore name spaces. **Generate a temporary file**: click the three-dot button to browse to the XML temporary file and set its path in the field.
	Use Separator for mode Xerces	Select this check box if you want to separate concatenated children node values. This field can only be used if the selected **Generation mode** is Xerces. The following field displays: **Field separator**: Define the delimiter to be used to separate the children node values.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Generation mode	From the drop-down list select the generation mode for the XML file, according to the memory available and the desired speed: • **Slow and memory-consuming (Dom4j)** This option allows you to use dom4j to process the XML files of high complexity. • **Memory-consuming (Xerces)**. • **Fast with low memory consumption (SAX)**
	Validate date	Select this check box to check the date format strictly against the input schema.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	**tFileInputXML** is for use as an entry componant. It allows you to create a flow of XML data using a **Row > Main** link. You can also create a rejection flow using a **Row > Reject** link to filter the data which doesn't correspond to the type defined. For an example of how to use these two links, see section *Scenario 2: Extracting correct and erroneous data from an XML field in a delimited file*.	
Limitation	n/a	

Scenario 1: Reading and extracting data from an XML structure

This scenario describes a basic Job that reads a defined XML directory and extracts specific information and outputs it on the **Run** console via a **tLogRow** component.



1. Drop **tFileInputXML** and **tLogRow** from the **Palette** to the design workspace.
2. Connect both components together using a **Main Row** link.
3. Double-click **tFileInputXML** to open its **Basic settings** view and define the component properties.



4. As the street dir file used as input file has been previously defined in the Metadata area, select **Repository** as **Property type**. This way, the properties are automatically leveraged and the rest of the properties fields are filled in (apart from Schema). For more information regarding the metadata creation wizards, see *Talend Open Studio for Big Data* User Guide.
5. Select the same way the relevant schema in the Repository metadata list. **Edit schema** if you want to make any change to the schema loaded.
6. The **Filename** shows the structured file to be used as input
7. In **Loop XPath query**, change if needed the node of the structure where the loop is based.
8. On the **Mapping** table, fill the fields to be extracted and displayed in the output.
9. If the file size is consequent, fill in a **Limit** of rows to be read.
10. Enter the encoding if needed then double-click on **tLogRow** to define the separator character.
11. Save your Job and press **F6** to execute it.

```
Starting job XMLStreetFinder at 12:42 05/01/2007.
Paris|2eme arrondissement|Rue de la Paix
Paris|8eme arrondissement|Champs Elysees
New York City|Manhattan|Madison avenue
New York City|Brooklyn|Washington heights
Job XMLStreetFinder ended at 12:42 05/01/2007. [exit code
```

The fields defined in the input properties are extracted from the XML structure and displayed on the console.

Scenario 2: Extracting erroneous XML data via a reject flow

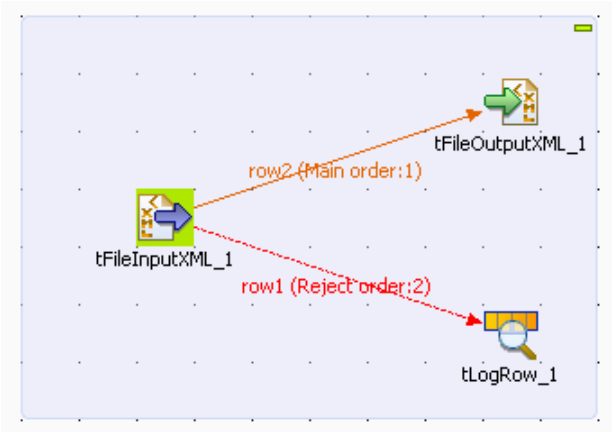
This Java scenario describes a three-component Job that reads an XML file and:

1. first, returns correct XML data in an output XML file,
2. and second, displays on the console erroneous XML data which type does not correspond to the defined one in the schema.

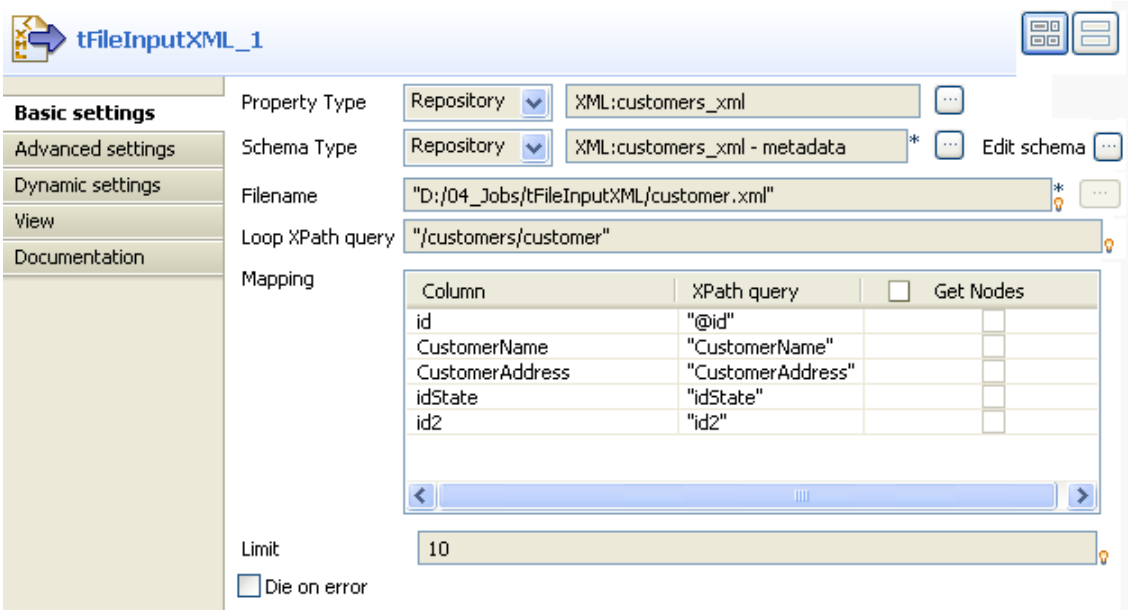
1. Drop the following components from the **Palette** to the design workspace: **tFileInputXML**, **tFileOutputXML** and **tLogRow**.

   Right-click **tFileInputXML** and select **Row** > **Main** in the contextual menu and then click **tFileOutputXML** to connect the components together.

   Right-click **tFileInputXML** and select **Row** > **Reject** in the contextual menu and then click **tLogRow** to connect the components together using a reject link.



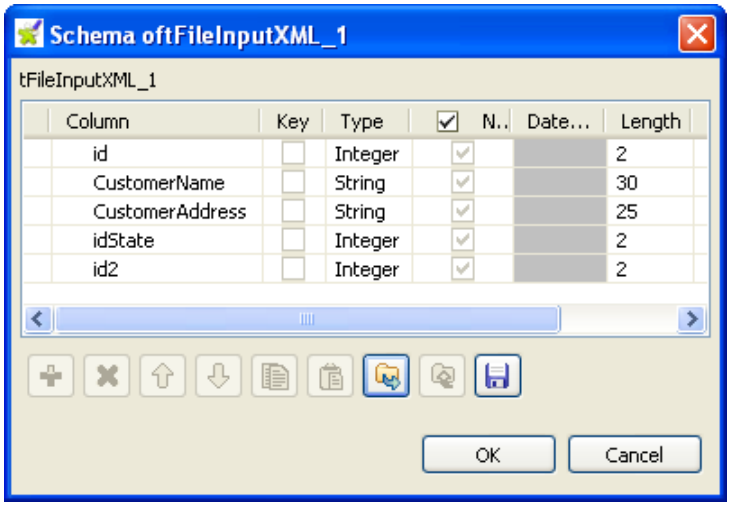
2. Double-click **tFileInputXML** to display the **Basic settings** view and define the component properties.



3. In the **Property Type** list, select **Repository** and click the three-dot button next to the field to display the **[Repository Content]** dialog box where you can select the metadata relative to the input file if you have already stored it in the **File xml** node under the **Metadata** folder of the **Repository** tree view. The fields that follow are automatically filled with the fetched data. If not, select **Built-in** and fill in the fields that follow manually.

   For more information about storing schema metadat in the Repository tree view, see *Talend Open Studio for Big Data* User Guide.

4. In the **Schema Type** list, select **Repository** and click the three-dot button to open the dialog box where you can select the schema that describe the structure of the input file if you have already stored it in the **Repository** tree view. If not, select **Built-in** and click the three-dot button next to **Edit schema** to open a dialog box where you can define the schema manually.



   The schema in this example consists of five columns: *id*, *CustomerName*, *CustomerAddress*, *idState* and *id2*.

5. Click the three-dot button next to the **Filename** field and browse to the XML file you want to process.

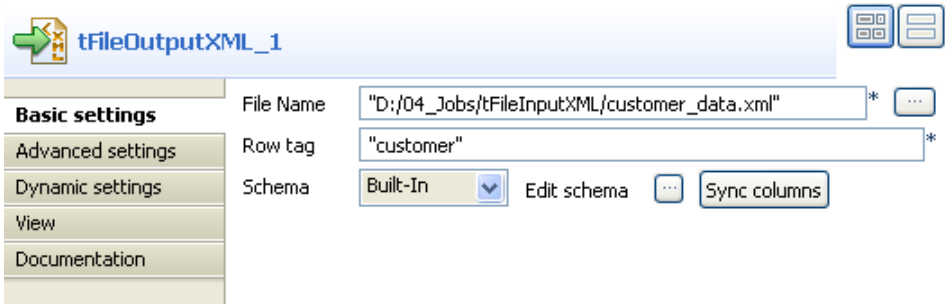
6. In the **Loop XPath query**, enter between inverted commas the path of the XML node on which to loop in order to retrieve data.

   In the **Mapping** table, **Column** is automatically populated with the defined schema.

In the **XPath query** column, enter between inverted commas the node of the XML file that holds the data you want to extract from the corresponding column.

7. In the **Limit** field, enter the number of lines to be processed, the first 10 lines in this example.

8. Double-click **tFileOutputXML** to display its **Basic settings** view and define the component properties.



9. Click the three-dot button next to the **File Name** field and browse to the output XML file you want to collect data in, *customer_data.xml* in this example.

   In the **Row tag** field, enter between inverted commas the name you want to give to the tag that will hold the recuperated data.

   Click **Edit schema** to display the schema dialog box and make sure that the schema matches that of the preceding component. If not, click **Sync columns** to retrieve the schema from the preceding component.

10. Double-click **tLogRow** to display its **Basic settings** view and define the component properties.

    Click **Edit schema** to open the schema dialog box and make sure that the schema matches that of the preceding component. If not, click **Sync columns** to retrieve the schema of the preceding component.

    In the **Mode** area, select the **Vertical** option.

11. Save your Job and press **F6** to execute it.

```
Starting job tFileInputXML at 14:34 06/11/2009.

[statistics] connecting to socket on port 3996
[statistics] connected
.-------------------------------------------------.
|                    #1. tLogRow_1                |
+-----------------+-------------------------------+
| key             | value                         |
+-----------------+-------------------------------+
| id              | null                          |
| CustomerName    | null                          |
| CustomerAddress | null                          |
| idState         | null                          |
| id2             | null                          |
| errorCode       | null                          |
| errorMessage    | For input string: "ab" - Line: 1 |
+-----------------+-------------------------------+
.-------------------------------------------------.
|                    #2. tLogRow_1                |
+-----------------+-------------------------------+
| key             | value                         |
+-----------------+-------------------------------+
| id              | 9                             |
| CustomerName    | Childress Child Day Care      |
| CustomerAddress | 788 Tennyson Ave.             |
| idState         | 12                            |
| id2             | null                          |
| errorCode       | null                          |
| errorMessage    | For input string: "cd" - Line: 8 |
+-----------------+-------------------------------+
[statistics] disconnected
Job tFileInputXML ended at 14:35 06/11/2009. [exit code=0]
```

The output file *customer_data.xml* holding the correct XML data is created in the defined path and erroneous XML data is displayed on the console of the **Run** view.

tFileOutputXML

tFileOutputXML properties

Component family	XML or File/Output	
Function	**tFileOutputXML** outputs data to an XML type of file.	
Purpose	**tFileOutputXML** writes an XML file with separated data value according to a defined schema.	
Basic settings	*File name*	Name or path to the output file and/or the variable to be used. Related topic: see Defining variables from the Component view section in *Talend Open Studio for Big Data User Guide*
	Incoming record is a document	Select this check box if the data from the preceding component is in XML format. When this check box is selected, a **Column list** appears allowing you to select a **Document** type column of the schema that holds the data, and the **Row tag** field disappears.
	Row tag	Specify the tag that will wrap data and structure per row.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Open Studio for Big Data User Guide*.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: see *Talend Open Studio for Big Data*.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
Advanced settings	*Split output in several files*	If the output is big, you can split the output into several files, each containing the specified number of rows. **Rows in each output file**: Specify the number of rows in each output file.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output XML files if required.
	Root tags	Specify one or more root tags to wrap the whole output file structure and data. The default root tag is *root*.
	Output format	Define the output format. **Column**: The columns retrieved from the input schema. **As attribute**: select check box for the column(s) you want to use as attribute(s) of the parent element in the XML output. If the same column is selected in both the **Output format** table as an attribute and in the **Use dynamic grouping** setting as the criterion for dynamic grouping, only the dynamic group setting will take effect for that column. **Use schema column name**: By default, this check box is selected for all columns so that the column labels from the input schema

		are used as data wrapping tags. If you want to use a different tag than from the input schema for any column, clear this check box for that column and specify a tag label between quotation marks in the **Label** field.
	Use dynamic grouping	Select this check box if you want to dynamically group the output columns. Click the plus button to add one ore more grouping criteria in the **Group by** table. **Column**: Select a column you want to use as a wrapping element for the grouped output rows. **Attribute label**: Enter an attribute label for the group wrapping element, between quotation marks.
	Custom the flush buffer size	Select this check box to define the number of rows to buffer before the data is written into the target file and the buffer is emptied. **Row Number**: Specify the number of rows to buffer.
	Advanced separator (for numbers)	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Don't generate empty file	Select the check box to avoid the generation of an empty file.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	Use this component to write an XML file with data passed on from other components using a **Row** link.	
Limitation	n/a	

Related scenarios

For related scenarios using **tFileOutputXML**, see section *Scenario 1: From Positional to XML file* and section *Scenario 2: Using a SOAP message from an XML file to get weather information and saving the information to an XML file*.

tWriteXMLField

tWriteXMLField properties

Component family	XML	
Function	**tWriteXMLField** outputs data to defined fields of the output XML file.	
Purpose	**tWriteXMLField** reads an input XML file and extracts the structure to insert it in defined fields of the output file.	
Basic settings	*Output Column*	Select the destination field in the output component where you want to write the XML structure.
	Configure Xml Tree	Opens the interface that supports the creation of the XML structure you want to write in a field. For more information about the interface, see section *Defining the XML tree*.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create the schema and store it locally for this component only. Related topic: see *Talend Open Studio for Big Data* User Guide.
		Repository: You already created the schema and stored it in the Repository, hence can be reused in various projects and job flowcharts. Related topic: see *Talend Open Studio for Big Data* User Guide.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the output component.
	Group by	Define the aggregation set, the columns you want to use to regroup the data.
Advanced settings	*Remove the xml declaration*	Select this check box if you do not want to include the XML header.
	Create empty element if needed	This check box is selected by default. If the **Related Column** in the interface that supports the creation of the XML structure has null values, or if no column is associated with the XML node, this option creates an open/close tag in the expected place.
	Create associated XSD file	If one of the XML elements is defined as a Namespace element, this option will create the corresponding XSD file. To use this option, you must select the **Dom4J** generation mode.
	Advanced separator (for number)	Select this check box if you want to modify the separators used by default for numbers. **Thousands separator**: enter between brackets the separators to use for thousands. **Decimal separator**: enter between brackets the separators to use for decimals.
	Generation mode	Select the appropriate generation mode according to your memory availability. The available modes are: • **Slow and memory-consuming (Dom4j)** This option allows you to use dom4j to process the XML files of high complexity.

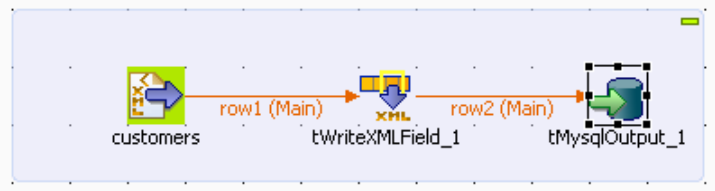
		• **Fast with low memory consumption**
	Encoding Type	Select the encoding type in the list or select **Custom** and define it manually. This field is compulsory when working with databases.
	tStatCatcher Statistics	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	n/a	

Scenario: Extracting the structure of an XML file and inserting it into the fields of a database table

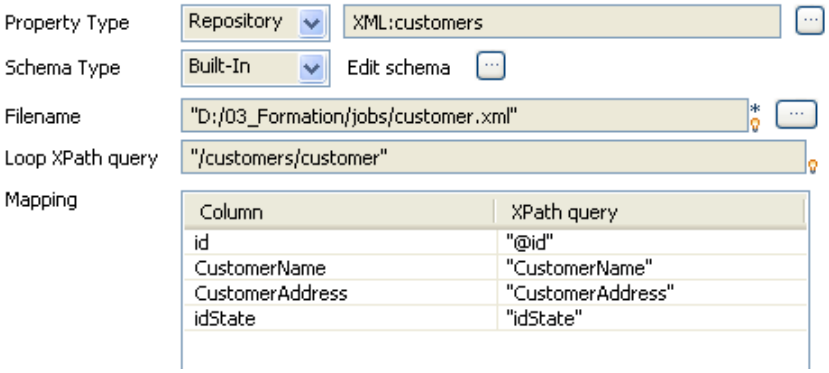
This three-component scenario allows to read an XML file, extract the XML structure, and finally outputs the structure to the fields of a database table.

1. Drop the following components from the **Palette** onto the design workspace: **tFileInputXml**, **tWriteXMLField**, and **tMysqlOutput**.

   Connect the three components using **Main** links.



2. Double-click **tFileInputXml** to open its **Basic settings** view and define its properties.



3. If you have already stored the input schema in the **Repository** tree view, select **Repository** first from the **Property Type** list and then from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.

   For more information about storing schema metadata in the **Repository** tree view, see *Talend Open Studio for Big Data* **User Guide**.

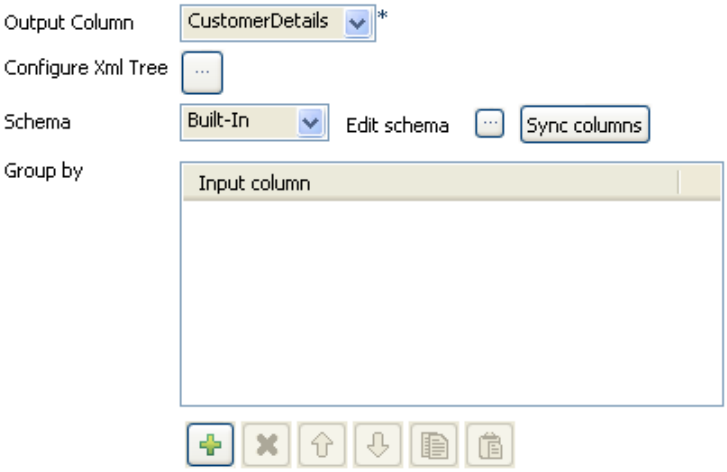
4. If you have not stored the input schema locally, select **Built-in** in the **Property Type** and **Schema** fields and fill in the fields that follow manually. For more information about **tFileInputXML** properties, see section *tFileInputXML*.

   If you have selected **Built-in**, click the three-dot button next to the **Edit schema** field to open a dialog box where you can manually define the structure of your file.

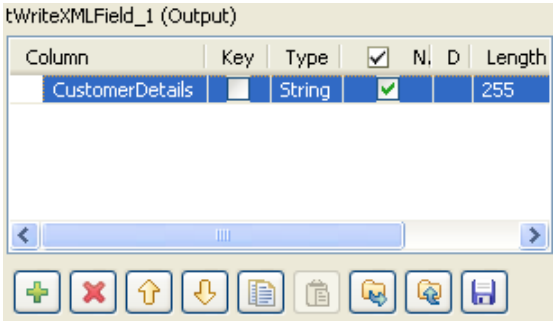
5. In the **Look Xpath query** field, enter the node of the structure where the loop is based. In this example, the loop is based on the *customer* node. **Column** in the **Mapping** table will be automatically populated with the defined file content.

In the **Xpath query** column, enter between inverted commas the node of the XML file that holds the data corresponding to each of the **Column** fields.

6. In the design workspace, click **tWriteXMLField** and then in the **Component** view, click **Basic settings** to open the relevant view where you can define the component properties.



7. Click the three-dot button next to the **Edit schema** field to open a dialog box where you can add a line by clicking the plus button.



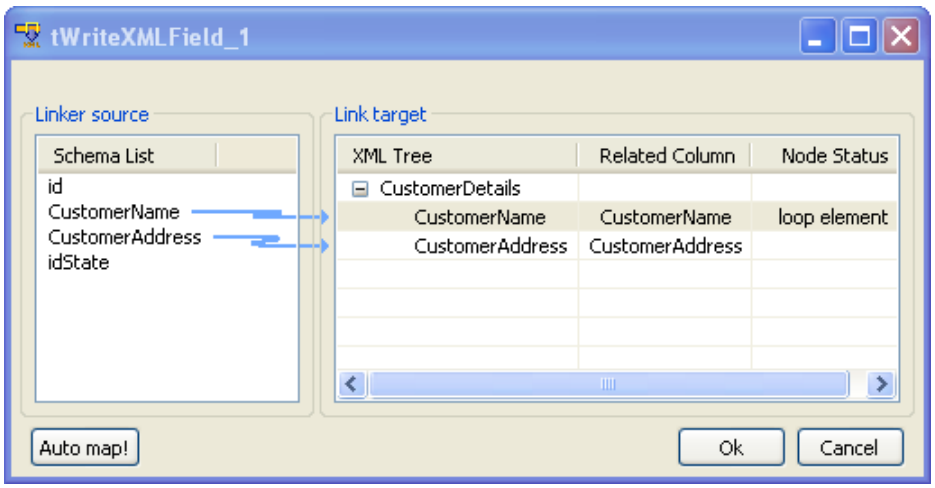
8. Click in the line and enter the name of the output column where you want to write the XML content, *CustomerDetails* in this example.

   Define the type and length in the corresponding fields, *String* and *255*in this example.

   Click **Ok** to validate your output schema and close the dialog box.

   In the **Basic settings** view and from the **Output Column** list, select the column you already defined where you want to write the XML content.

9. Click the three-dot button next to **Configure Xml Tree** to open the interface that helps to create the XML structure.



10. In the **Link Target** area, click *rootTag* and rename it as *CustomerDetails*.

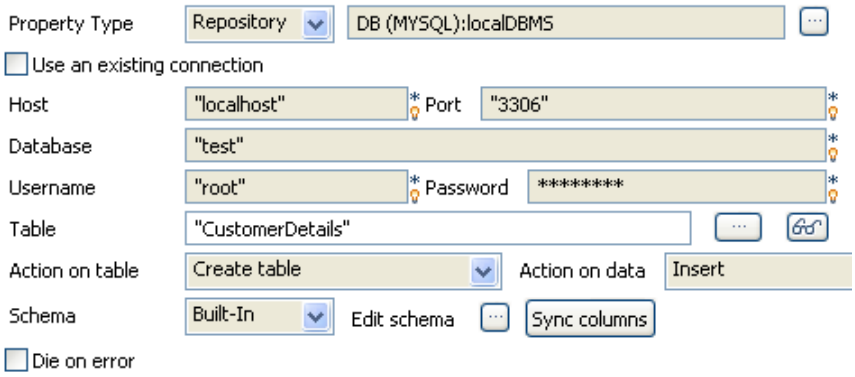
    In the **Linker source** area, drop *CustomerName* and *CustomerAddress* to *CustomerDetails*. A dialog box displays asking what type of operation you want to do.

    Select **Create as sub-element of target node** to create a sub-element of the *CustomerDetails node*.

    Right-click *CustomerName* and select from the contextual menu **Set As Loop Element**.

    Click **OK** to validate the XML structure you defined.

11. Double-click **tMysqlOutput** to open its **Basic settings** view and define its properties.



12. If you have already stored the schema in the **DB Connection** node in the **Repository** tree view, select **Repository** from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.

    For more information about storing schema metadata in the **Repository** tree view, see *Talend Open Studio for Big Data* **User Guide**.

    If you have not stored the schema locally, select **Built-in** in the **Property Type** and **Schema** fields and enter the database connection and data structure information manually. For more information about **tMysqlOutput** properties, see section *tMysqlOutput*.

    In the **Table** field, enter the name of the database table to be created, where you want to write the extracted XML data.

    From the **Action on table** list, select **Create table** to create the defined table.

    From the **Action on data** list, select Insert to write the data.

Click **Sync columns** to retrieve the schema from the preceding component. You can click the three-dot button next to **Edit schema** to view the schema.

13. Save your Job and click **F6** to execute it.

CustomerDetails
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>talend@apres91</CustomerAddress>...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>511 Maple Ave. Apt. 1B</CustomerA...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>662 Lyons Circle</CustomerAddress>...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>unknown</CustomerAddress>· <Cust...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>770 Exmoor Rd.</CustomerAddress>·...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>1860 Parkside Ln.</CustomerAddress...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>807 Old Trail Rd.</CustomerAddress>...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>618 Sheriden rd.</CustomerAddress>...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>788 Tennyson Ave.</CustomerAddre...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>2032 Northbrook Ct.</CustomerAddre...
<?xml version="1.0" encoding="ISO-8859-15"?>··<CustomerDetails>· <CustomerAddress>4522 N. Greenview Apt. 1B</Custom...

tWriteXMLField fills every field of the *CustomerDetails* column with the XML structure of the input file: the XML processing instruction `<?xml version=""1.0"" encoding=""ISO-8859-15""?>`, the first node that separates each client `<CustomerDetails>` and finally customer information `<CustomerAddress>` and `<CustomerName>`.

tXMLMap

tXMLMap belongs to two component families: Processing and XML. For more information on it, see section *tXMLMap*.

tXSDValidator

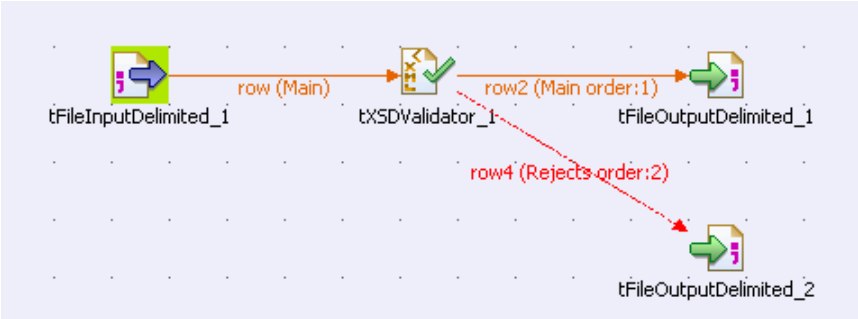
tXSDValidator Properties

Component family	XML	
Function	Validates an input XML file or an input XML flow against an XSD file and sends the validation log to the defined output.	
Purpose	Helps at controlling data and structure quality of the file or flow to be processed	
Basic settings	*Mode*	From this dropdown list, select: - **File**, to validate an input file - **Flow**, to validate an input flow
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but in this case, the schema is read-only. It contains standard information regarding the file validation.
File mode only	*XSD file*	Filepath to the reference XSD file. HTTP URL also supported, e.g. *http://localhost:8080/book.xsd*.
File mode only	*XML file*	Filepath to the XML file to be validated.
File mode only	*If XML is valid, display* *If XML is invalid, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
File mode only	*Print to console*	Select this check box to display the validation message.
Flow mode only	*Allocate*	Specify the column or columns to be validated and the path to the reference XSD file.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the Job processing metadata at a Job level as well as at each component level.
Usage	When used in **File** mode, this component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Limitation	n/a	

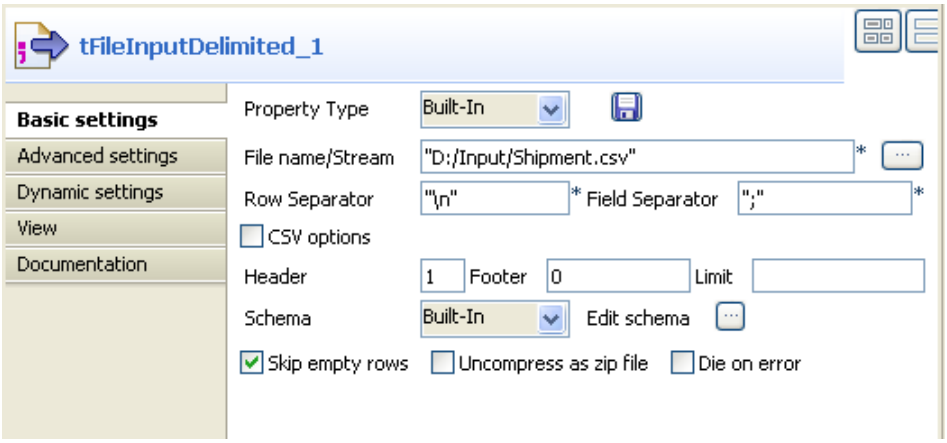
Scenario: Validating data flows against an XSD file

This scenario describes a Job that validates an XML column in an input file against a reference XSD file and outputs the log information for the invalid rows of the column into a delimited file. For the **tXSDValidator** use case that validates an XML file, see section *Scenario: Validating XML files*.

1. Drop a **tFileInputDelimited** component, a **tXSDValidator** component, and two **FileOutputDelimited** components from the **Palette** to the design workspace.



2. Double-click the **tFileInputDelimited** to open its **Component** view and set its properties:

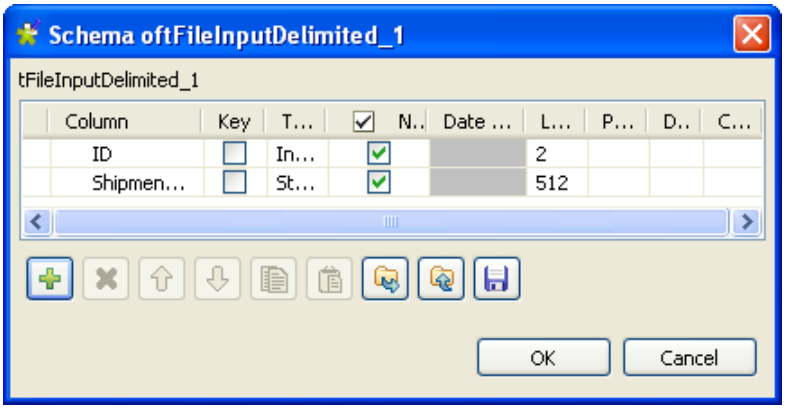


3. Use the **Built-In** property type for this scenario.

   Browse to the input file, and define the number of rows to be skipped in the beginning of the file.

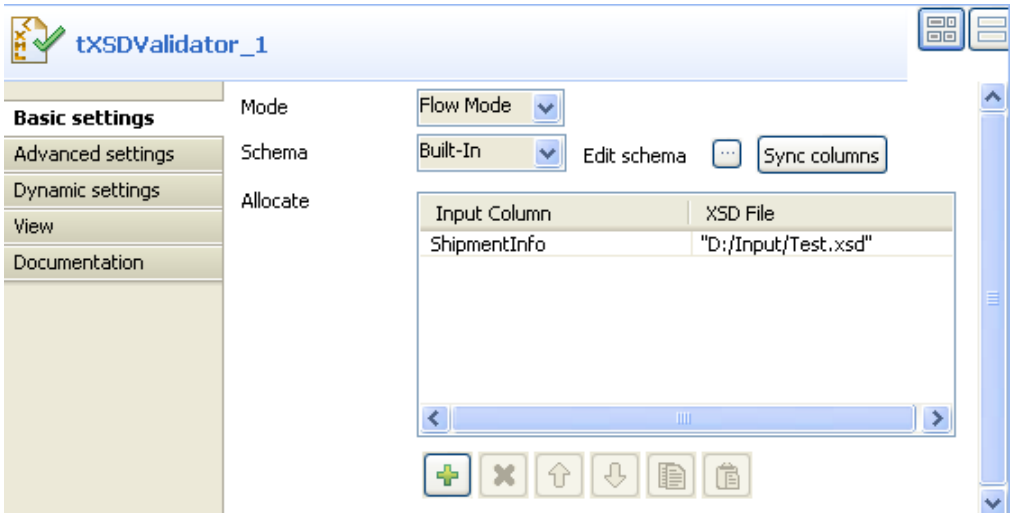
   Use a **Built-In** schema for this scenario. This means that it is available for this Job only.

   Click Edit schema and edit the schema according to the input file. In this scenario, the input file has only two columns: *ID* and *ShipmentInfo*. The *ShipmentInfo* column is an XML column and needs to be validated.



4. On your design workspace, connect the **tFileInputDelimited** component to the **tXSDValidator** component using a **Row** > **Main** link.

5. Double-click the **tXSDValidator** component, and set its properties:



6. From the **Mode** dropdown list, select **Flow Mode**.

   Use a **Built-In** schema for this scenario. Click **Sync columns** to retrieve the schema from the preceding component. To view or modify the schema, click the three-dot button next to **Edit schema**.

   Add a line in the **Allocate** table by clicking the plus button. The name of the first column of the input file automatically appears in the **Input Column** field. Click in the field and select the column you want to validate.

   In the **XSD File** field, fill in the path to your reference XSD file.

7. On your design workspace, connect the **tXSDValidator** component to one **tFileOutputDelimited** component using a **Row** > **Main** link to output the information about valid XML rows.

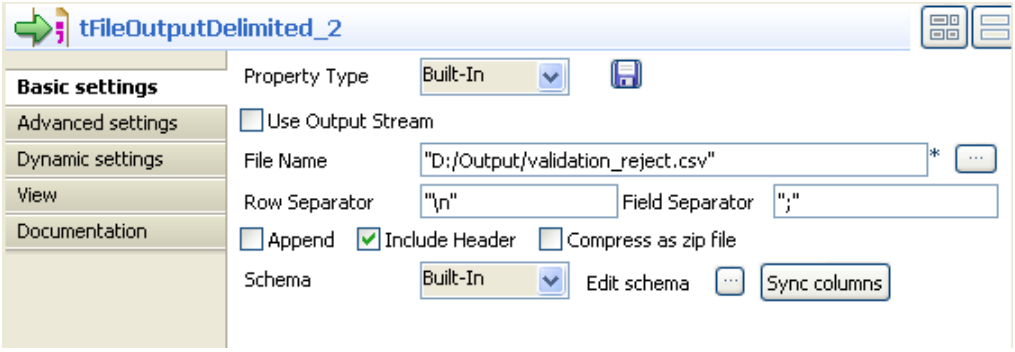
8. Connect the **tXSDValidator** component to the other **tFileOutputDelimited** component using a **Row** > **Rejects** link to output the information about invalid XML rows.

9. Double-click each of the two **tFileOutputDelimited** components and configure the component properties.

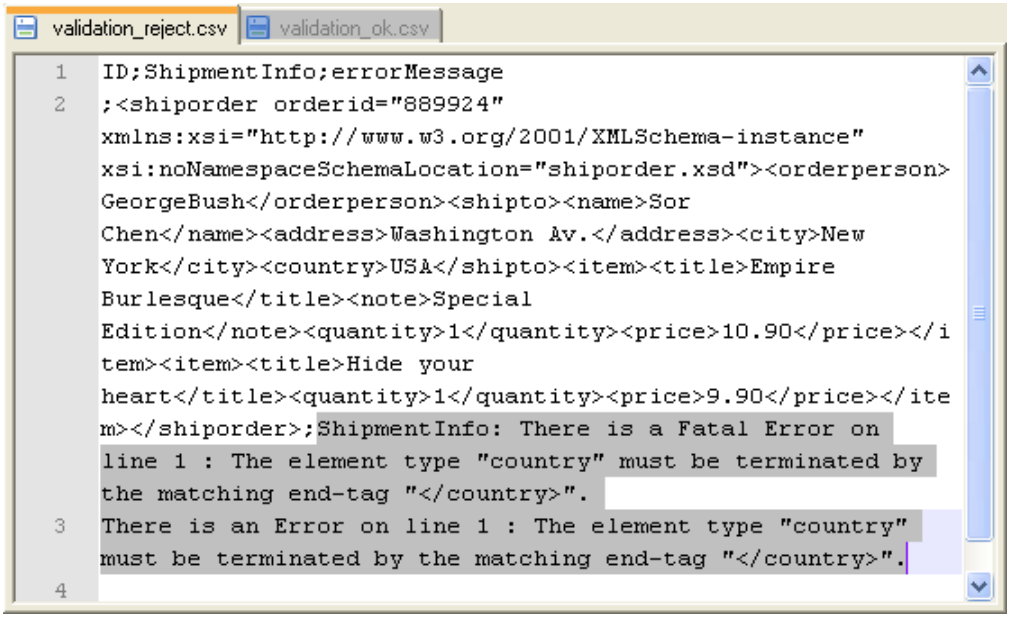
   In the **Property Type** field, select Built-In.

   In the **File Name** field, enter or, if you want to use an existing output file, browse to the output file path.

10. Select **Built-In** from the **Schema** list and click **Sync columns** to retrieve the schema from the preceding component.



11. Save your Job and press **F6** to run it.



The output files contain the validation information about the valid and invalid XML rows of the specified column respectively.

tXSLT

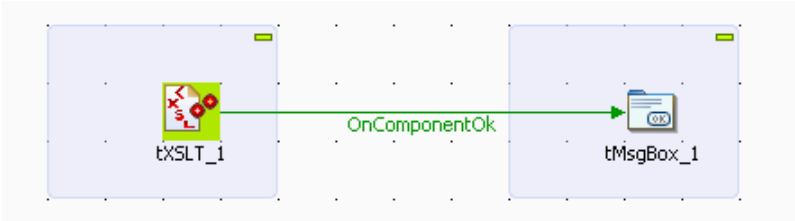
tXSLT Properties

Component family	XML	
Function	Refers to an XSL stylesheet, to transform an XML source file into a defined output file.	
Purpose	Helps to transform data structure to another structure.	
Basic settings	*XML file*	File path to the XML file to be validated.
	XSL file	File path to the reference XSL transformation file.
	Output file	File path to the output file. If the file does not exist, it will be created. The output file can be any structured or unstructured file such as html, xml, txt or even pdf or edifact depending on your xsl.
	Parameters	Click the plus button to add new lines in the **Parameters** list and define the transformation parameters of the XSLT file. Click in each line and enter the key in the *name* list and its associated value in the *value* list.
Usage	This component can be used as standalone component.	
Limitation	Due to license incompatibility, one or more JARs required to use this component are not provided. You can easily find out and add such JARs in the **Integration** perspective of your studio. For details, see *How to install external modules* in *Talend Open Studio for Big Data User Guide*.	

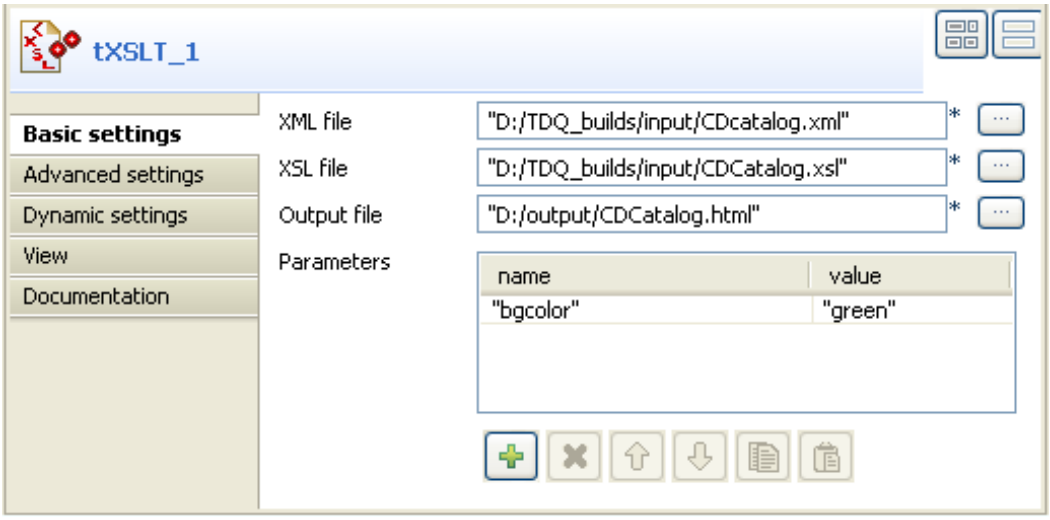
Scenario: Transforming XML to html using an XSL stylesheet

This scenario describes a two-component Job that converts xml data into an html document using an xsl stylesheet. It as well defines a transformation parameter of the xsl stylesheet to change the background color of the header of the created html document.

1. Drop the **tXSLT** and **tMsBox** components from the **Palette** to the design workspace.



2. Double-click **tXSLT** to open its **Basic settings** view where you can define the component properties.



3. In the **XML file** field, set the path or browse to the xml file to be transformed. In this example, the xml file holds a list of MP3 song titles and related information including artist names, company etc.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <catalog>
3   <cd>
4      <title>Empire Burlesque</title>
5      <artist>Bob Dylan</artist>
6      <country>USA</country>
7      <company>Columbia</company>
8      <price>10.90</price>
9      <year>1985</year>
10    </cd>
11    .
12    .
13
14 </catalog>
```

4. In the **XSL file** field in the **Basic settings** view, set the path or browse to the relevant xsl file.

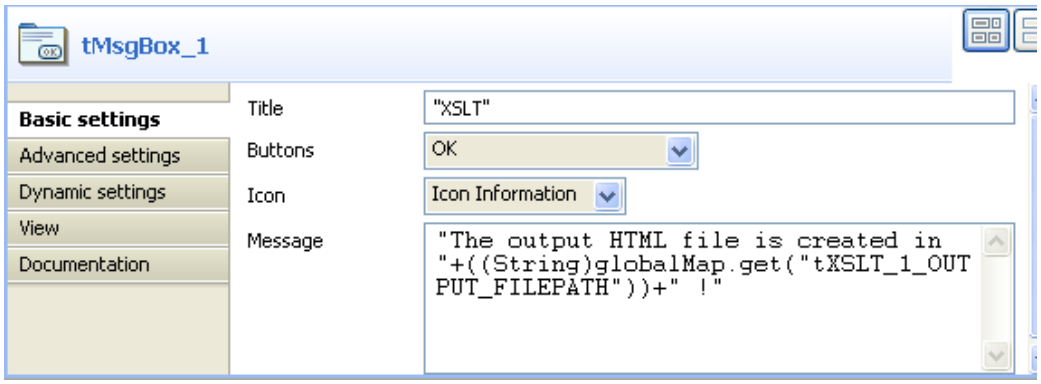
5. In the **Output file** field, set the path or browse to the output html file.

   In this example, we want to convert the xml data into an html file holding a table heading followed by a table listing artists' names next to song titles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:param name="bgcolor" />

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="{$bgcolor}">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

6. In the **Parameters** area of the **Basic settings** view, click the plus button to add a line where you can define the name and value of the transformation parameter of the xsl file. In this example, the name of the transformation parameter we want to use is *bgcolor* and the value is *green.*

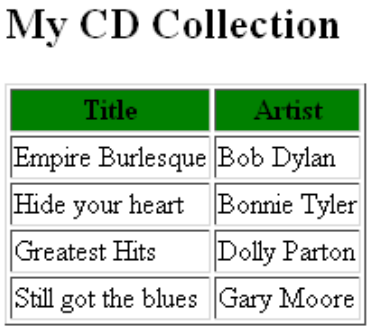
7. Double-click the **tMsgBox** to display its **Basic settings** view and define its display properties as needed.



8. Save the Job and press **F6** to execute it. The message box displays confirming that the output html file is created and stored in the defined path.

9. Click **OK** to close the message box.

You can now open the output html file to check the transformation of the xml data and that of the background color of the table heading.



My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore