Reference documentation

Reference documentation

3.0

Copyright © 2008-2011

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

Table of Contents

List of Tables

Part I. Introduction to DataCleaner

Table of Contents

Chapter 1. Background and concepts

Abstract

In this chapter we will try to define how we see the concepts and terms surrounding the environment(s) around DataCleaner.

Although these terms have no strict definitions, you can use this chapter as a guide, at least for the scope of how to use and what to expect from DataCleaner in relation to the described topics.

As a lot of the statements in this chapter are in deed subjective or based upon personal experience, we encourage everyone to provide their feedback and to contribute corrections/improvements to it.

What is Data Quality (DQ)?

Data Quality (DQ) is a concept and a business term covering the quality of the data used for a particular purpose. Often times the DQ term is applied to the quality of data used in business decisions but it may also refer to the quality of data used in research, campaigns, processes and more.

Working with Data Quality typically varies a lot from project to project, just as the issues in the quality of data vary a lot. Examples of data quality issues include:

1. Completeness of data
2. Correctness of data
3. Duplication of data
4. Uniformedness/standardization of data

A less technical definition of high-quality data is, that data are of high quality "if they are fit for their intended uses in operations, decision making and planning" (J. M. Juran).

What is Master Data Management (MDM)?

Master Data Management (MDM) is a very broad term and is seen materialized in a variety of ways. For the scope of this document it serves more as a context of data quality than an activity that we actually target per-se.

The overall goals of MDM is to manage the important data of an organization. By "master data" we refer to "a single version of the truth", ie. not the data of a particular system, but for example all the customer data of a company.

Obviously one of the very important issues to handle in MDM is the quality of data. If you simply gather eg. "all customer data" from all systems in an organization, you will most likely see a lot of data quality issues. There will be a lot of duplicate entries, there will be variances in the way that customer data is filled out, there will be different identifiers and even different levels of granularity for defining "what is a customer?".

What is Data Quality Analysis (DQA)?

Data Quality Analysis (DQA) is the (human) process of examining the quality of data for a particular process or organization. The DQA includes both technical and non-technical elements. For example, to

do a good DQA you will probably need to talk to users, business people, partner organizations and maybe customers. This is needed to asses what the goal of the DQA should be.

From a technical viewpoint the main task in a DQA is the data profiling activity, which will help you discover and measure the current state of affairs in the data.

What is Data Profiling?

Data Profiling is the activity of investigating a datastore to create a 'profile' of it. With a profile of your datastore you will be a lot better equipped to actually improve it.

The way you do profiling often depends on whether you already have some ideas about the quality of the data or if you're not experienced with the datastore at hand. Either way we recommend an *explorative* approach, because even though you think there are only a certain amount of issues you need to look for, it is our experience (and reasoning behind a lot of the features of DataCleaner) that it is just as important to check those items in the data that you think are correct! Typically it's cheap to include a bit more data into your analysis and the results just might surprise you and save you time!

What is Data Quality Monitoring?

We've argued that Data Profiling is ideally an explorative activity. Data Quality Monitoring typically isn't! The measurements that you do when profiling often times needs to be continuously checked so that your improvements are enforced through time. This is what Data Quality Monitoring is about.

As of version 3, DataCleaner now also includes a monitoring web application.

Data Quality Monitoring comes in different shapes and sizes. You can set up your own bulk of scheduled jobs that run every night. You can build alerts around it that send you emails if a particular measure goes beyond it's allowed thresholds, or in some cases you can attempt ruling out the issue entirely by applying First-Time-Right (FTR) principles that validate data at entry-time. eg. at data registration forms and more. With the DataCleaner monitoring application, these scenarios are provided in a convenient way for users of the DataCleaner profiler.

What is Open Source software?

That software is Open Source software basically means that the software is licensed under a license that gives you the right to use it (and it's source code) freely. But there are of course some limitations to these rights: First and foremost you cannot sell the software to a third party for money. In the case of DataCleaner it also means that if you modify or enhance the product, then you have to contribute back to the community your modifications. The idea here is that we should cooperate on building great software and that we should make this great software available to anyone for everybody's benefit.

So what *can* you do with it? You can use it for any purpose you'd like. You can even include it in a software bundle that you sell for money - but you can only sell the bundle, not DataCleaner as such.

For more information on Open Source software and the LGPL license which DataCleaner is distributed under, visit the Free Software Foundation:

http://www.gnu.org/copyleft/lesser.html

Chapter 2. Getting started

Installing the desktop application

The desktop version of DataCleaner requires practically no installation. The application is not dependent on any particular operating system and does not need to "register" in any registration database or something like that.

The only two requirements of DataCleaner are:

1. A computer (with a graphical display, except if run in command-line mode).

2. A Java Runtime Environment [http://www.java.com] (JRE), version 6 or higher.

You have two options in terms of installation.

1. Download DataCleaner as a distributable package (.zip or .tar.gz) from the release list on our downloads page [http://datacleaner.eobjects.org/downloads] .

   Unpackage the distributable in a directory of your own choice. DataCleaner will save it's configuration within this same directory.

2. Run DataCleaner as a Java WebStart application. You can do this by clicking the WebStart link on our downloads page [http://datacleaner.eobjects.org/downloads] .

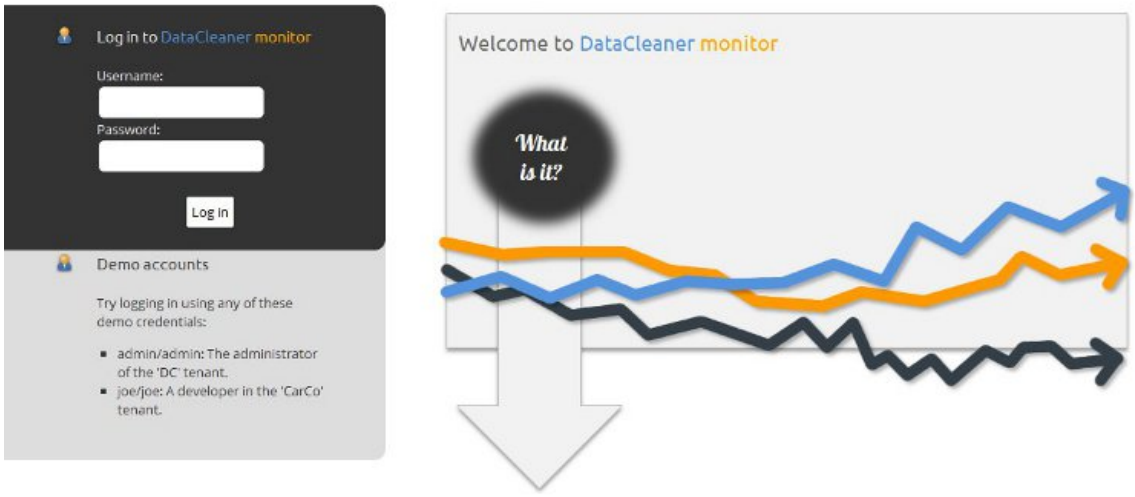
   In WebStart the application will be installed in your JRE's application cache. Each time you start the application it will automatically check for updates to the application. DataCleaner's configuration will be stored in your "home"-directory in a folder called .datacleaner/[version].

Installing the monitoring web application

In addition to (and in some cases, even as a replacement for) the desktop version of DataCleaner, we also provide a web application for monitoring, scheduling and sharing analysis jobs and results.

A Java servlet container and web server is required to run the monitoring web application. An example of this is Apache Tomcat 7.x [http://tomcat.apache.org/download-70.cgi] , which is often used and tested by the DataCleaner development team.

To install the monitoring web application, download [http://datacleaner.eobjects.org/downloads] the Web Archive (.war) distribution of DataCleaner. Install the .war file in your container. If you're using Apache Tomcat, this is done by copying the .war file to the "webapps" folder within your tomcat directory. Afterwards, start the container and go to http://localhost:8080/DataCleaner-monitor to see the welcome/login screen:



In the community edition of DataCleaner, you will find suggestions for login credentials directly on the screen, to get you started quickly.

Connecting to your datastore

Below is a screenshot of the initial screen that will be presented when launching DataCleaner (desktop edition), containing a list of datastores. Above the list you will see a set of icons, each representing a type of datastore. Click either of the icons to register your own datastore.

Once you've registered ('created') your own datastore, you can select it from the list and click 'Analyze!' to start working with it!

Tip

You can also configure your datastore by means of the configuration file (conf.xml), which has both some pros and some cons. For more information, read the configuration file chapter .

Chapter 3. Building jobs

Adding components to the job

There are a few different kinds of components that you can add to your job:

1. *Analyzers* , which are the most important components. Actually, without at least one analyzer the job will not run. An analyzer is a component that inspects the data that it receives and generates a result or a report. The majority of the data profiling cruft is created as analyzers.

2. *Transformers* are components used to modify the data before analyzing it. Sometimes it's nescesary to extract parts of a value or combine two values to correctly get an idea about a particular measure. In other scenarios, transformers can be used to perform reference data lookups or other similar tasks and place the results of an operation into the stream of data in the job.

   The result of a transformer is a set of output columns. These columns work excactly like regular columns in your job, except that they have a preceeding step in the flow before they become materialized.

3. *Filters* are components that split the flow of processing in a job. A filter will have a number of possible outcomes and depending on the outcome of a filter, a particular row might be processed by different sub-flows. Filters are often used simply to disregard certain rows from the analysis, eg. null values or values outside the range of interest.

Each of these components will get their own tab, from where you can configure them.

Transformers and filters are added to your job using the "Transform" button. Please refer to the reference chapter Transformations for more information on specific transformers and filters.

Analyzers are added to your job using the "Analyze" button. Please refer to the reference chapter Analyzers for more information on specific analyzers.

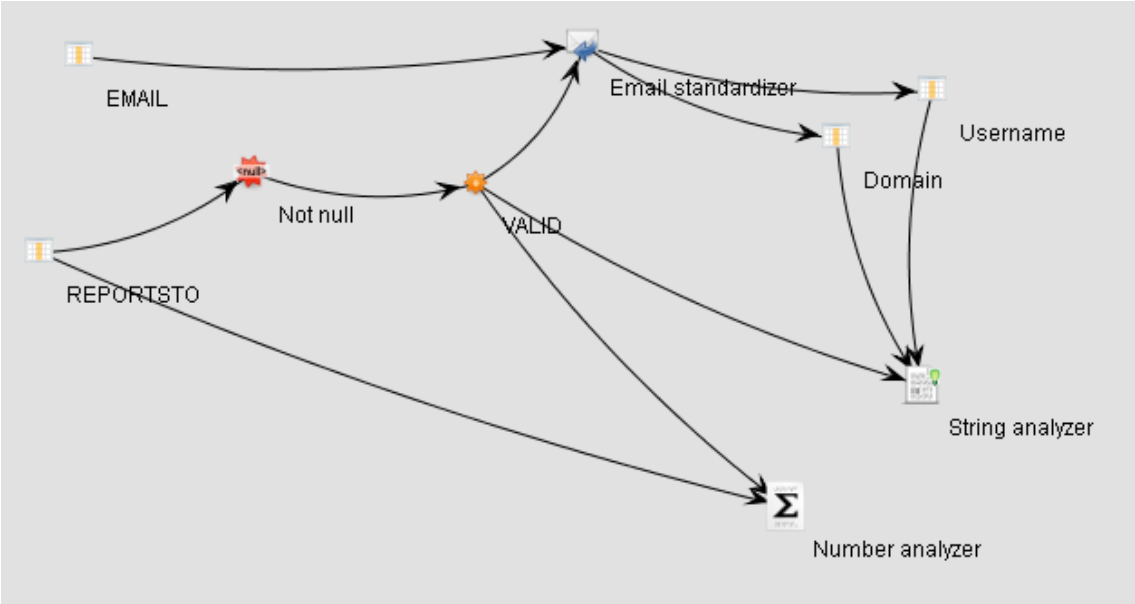
Wiring components together

Simply adding a transformer or filter actually doesn't change your job as such! This is because these components only have an impact if you wire them together somehow.

To wire a transformer you simply need to use it's output column. DataCleaner will automatically build the flow so that a transformer is executed before components that depend on it's output columns.

To wire a filter you need to set up a dependency on either of it's outcomes. All components have a button for selecting filter outcomes in their top-right corners. Click this button to select a filter outcome to depend on. If you have multiple filters you can chain these simply by having dependent outcomes of the individual filters.

To get an overview of your current job flow, you can click the "Visualize" button, which will present the job's contents in an interactive flow diagram:



Executing jobs

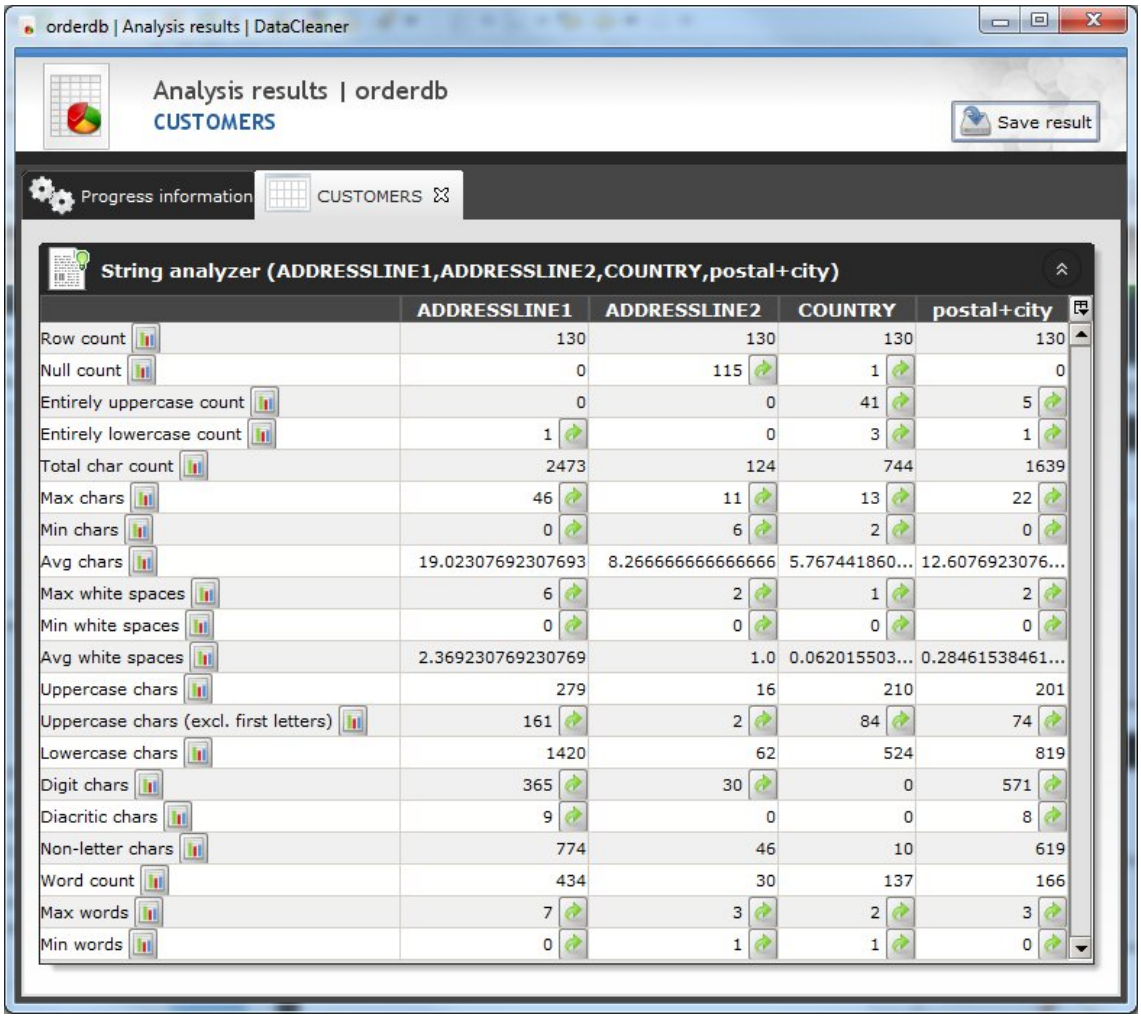
When a job has been built you can execute it. To check whether your job is correctly configured and ready to execute, check the status bar in the bottom of the job building window.

To execute the job, simply click the "Run analysis" button in the top-right corner of the window. This will bring up the result window, which contains:

1. The *Progress information* tab which contains useful information and progress indications while the job is executing.

2. Additional tabs for each table that is being processed in the job. Results for the individual analyzers will be shown in these tabs.

Here's an example of an analysis result window:



	ADDRESSLINE1	ADDRESSLINE2	COUNTRY	postal+city
Row count	130	130	130	130
Null count	0	115	1	0
Entirely uppercase count	0	0	41	5
Entirely lowercase count	1	0	3	1
Total char count	2473	124	744	1639
Max chars	46	11	13	22
Min chars	0	6	2	0
Avg chars	19.02307692307693	8.266666666666666	5.767441860...	12.6076923076...
Max white spaces	6	2	1	2
Min white spaces	0	0	0	0
Avg white spaces	2.369230769230769	1.0	0.062015503...	0.28461538461...
Uppercase chars	279	16	210	201
Uppercase chars (excl. first letters)	161	2	84	74
Lowercase chars	1420	62	524	819
Digit chars	365	30	0	571
Diacritic chars	9	0	0	8
Non-letter chars	774	46	10	619
Word count	434	30	137	166
Max words	7	3	2	3
Min words	0	1	1	0

Saving and opening jobs

You can save your jobs in order to reuse them at a later time. Saving a job is simple: Simply click the "Save analysis job" button in the top-left corner of the window.

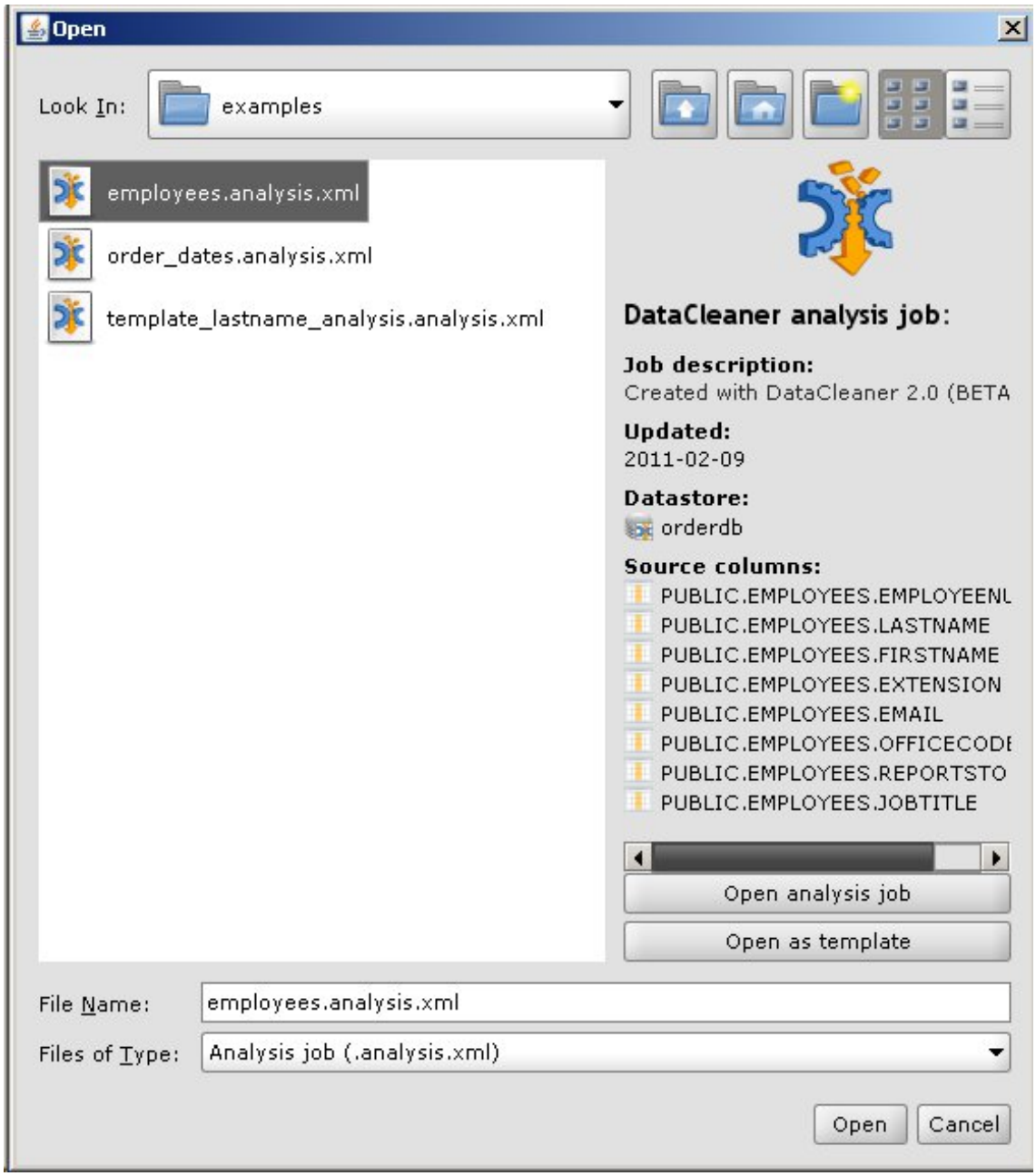
Analysis jobs are saved in files with the ".analysis.xml" extension. These files are XML files that are readable and editable using any XML editor.

Opening jobs can be done using the "File -> Open analysis job..." menu item. Opening a job will restore a job building window from where you can edit and run the job.

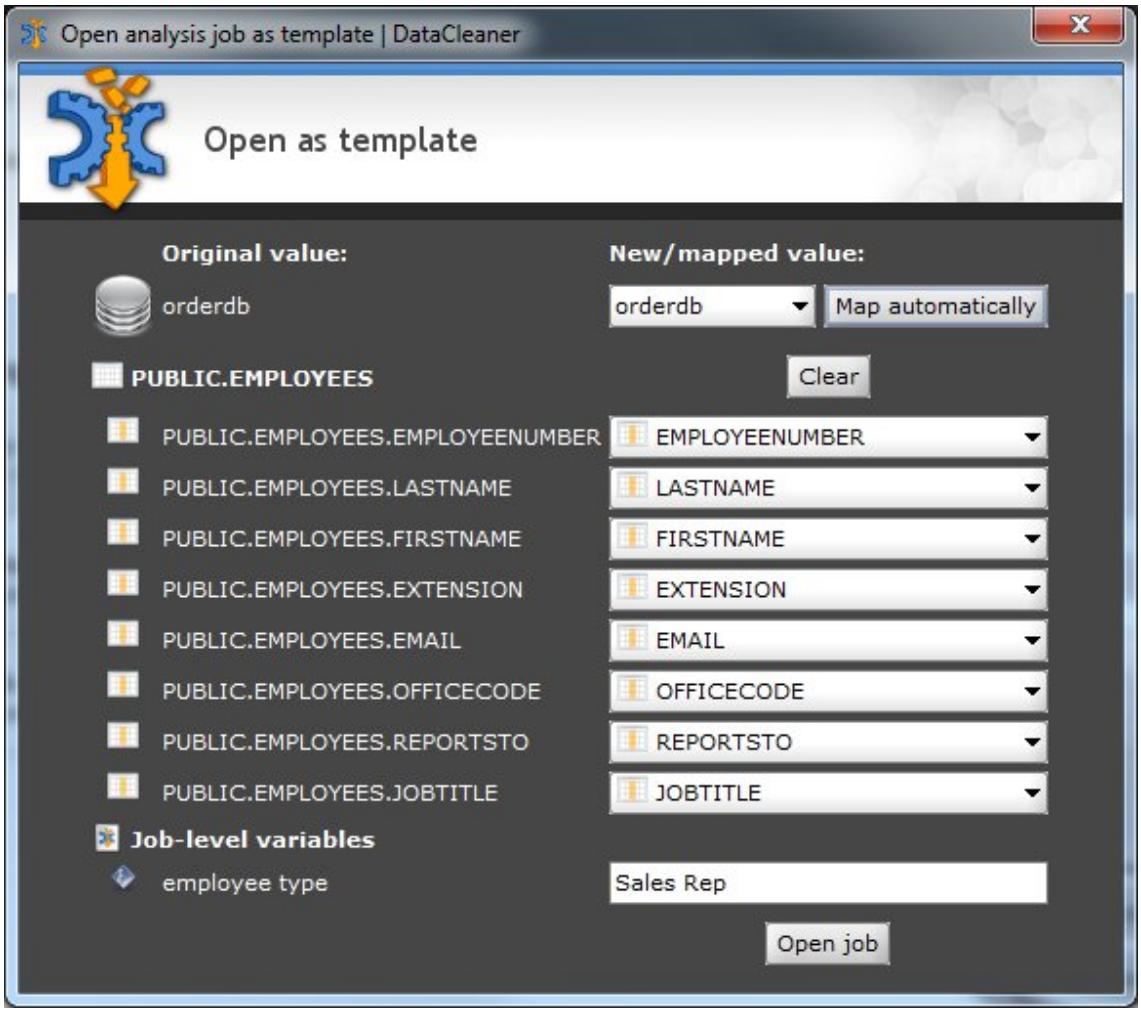
Template jobs

DataCleaner contains a feature where you can reuse a job for multiple datastores or just multiple columns in the same datastore. We call this feature 'template jobs'.

When opening a job you are presented with a file chooser. When you select a job file a panel will appear, containing some information about the job as well as available actions:



If you click the 'Open as template' button you will be presented with a dialog where you can map the job's original columns to a new set of columns:



First you need to specify the datastore to use. On the left side you see the name of the original datastore, but the job is not restricted to use only this datastore. Select a datastore from the list and the fields below for the columns will become active.

Then you need to map individual columns. If you have two datastore that have the same column names, you can click the "Map automatically" button and they will be automatically assigned. Otherwise you need to map the columns from the new datastore's availabel columns.

Finally your job may contain 'Job-level variables'. These are configurable properties of the job that you might also want to fill out.

Once these 2-3 steps have been completed, click the "Open job" button, and DataCleaner will be ready for executing the job on a new set of columns!

Writing cleansed data to files

Although the primary focus of DataCleaner is analysis, often during such analysis you will find yourself actually improving data by means of applying transformers and filters on it. When this is the case, obviously you will want to export the improved/cleansed data so you can utilize it in other situations than the analysis.

Please refer to the reference chapter Writers for more information on writing cleansed data.

Part II. Analysis component reference

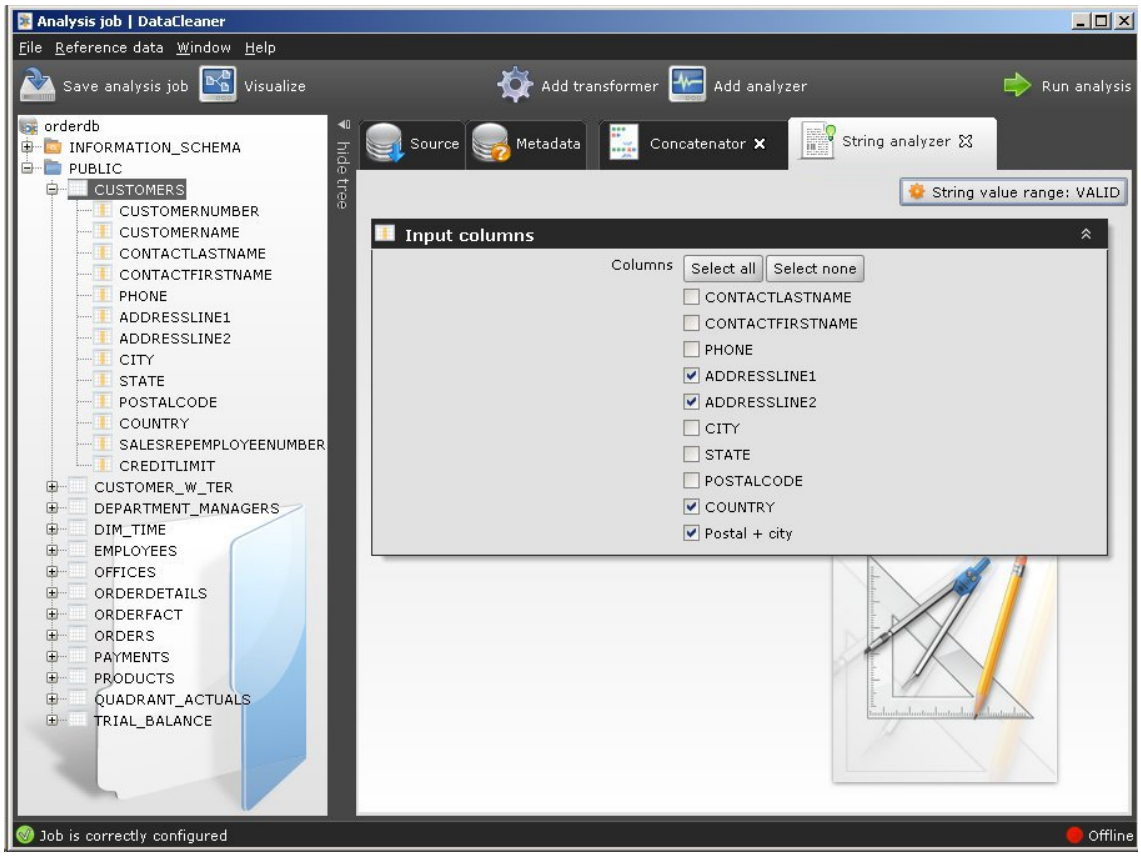
Table of Contents

Chapter 4. Analyzers

This chapter deals with one of the most important concepts in DataCleaner: Analyzers. Analyzers are the endpoints of any analysis job, meaning that a job requires at least on analyzer.

An analyzer consumes a (set of) column(s) and generates an analysis result based on the values in the consumed columns.

Here is an example of a configuration panel pertaining to an analyzer:



In the panel there will always be one or more selections of columns. The configuration panel may also contain additional properties for configuration.

Completeness analyzer

The completeness analyzer provides a really simple way to check that all required fields in your records have been filled. Think of it like a big "not null" check across multiple fields. In combination with the monitoring application, this analyzer makes it easy to track which records needs additional information.

EasyDQ matching and deduplication

EasyDataQuality [http://www.easydq.com] is an on-demand service for data quality functions. DataCleaner provides access to the EasyDQ services, including the deduplication service which is used to provide *Duplicate detection* and *Inter-Dataset matching* , but the core functionality is provided by Human Inference.

Please refer to the EasyDQ for DataCleaner documentation [http://help.easydq.com/datacleaner] for detailed information about the services provided through EasyDQ.

Boolean analyzer

Boolean analyzer is an analyzer targeted at boolean values. For a single boolean column it is quite simple: It will show the distribution of true/false (and optionally null) values in a column. For several columns it will also show the value combinations and the frequencies of the combinations. The combination matrix makes the Boolean analyzer a handy analyzer for use with combinations of matching transformers and other transformers that yield boolean values.

Boolean analyzer has no configuration parameters, except for the input columns.

Character set distribution

The Character set distribution analyzer inspects and maps text characters according to character set affinity, such as Latin, Hebrew, Cyrillic, Chinese and more.

Such analysis is convenient for getting insight into the international aspects of your data. Are you able to read and understand all your data? Will it work in your non-internationalized systems?

Date gap analyzer

The Date gap analyzer is used to identify gaps in recorded time series. This analyzer is useful for example if you have employee time registration systems which record FROM and TO dates. It will allow you to identify if there are unexpected gaps in the data.

Date/time analyzer

The Date/time analyzer provides general purpose profiling metrics for temporal column types such as DATE, TIME and TIMESTAMP columns.

Matching analyzer

The matching analyzer provides an easy means to match several columns against several dictionaries and/or several string patterns. The result is a matrix of match information for all columns and all matched resources.

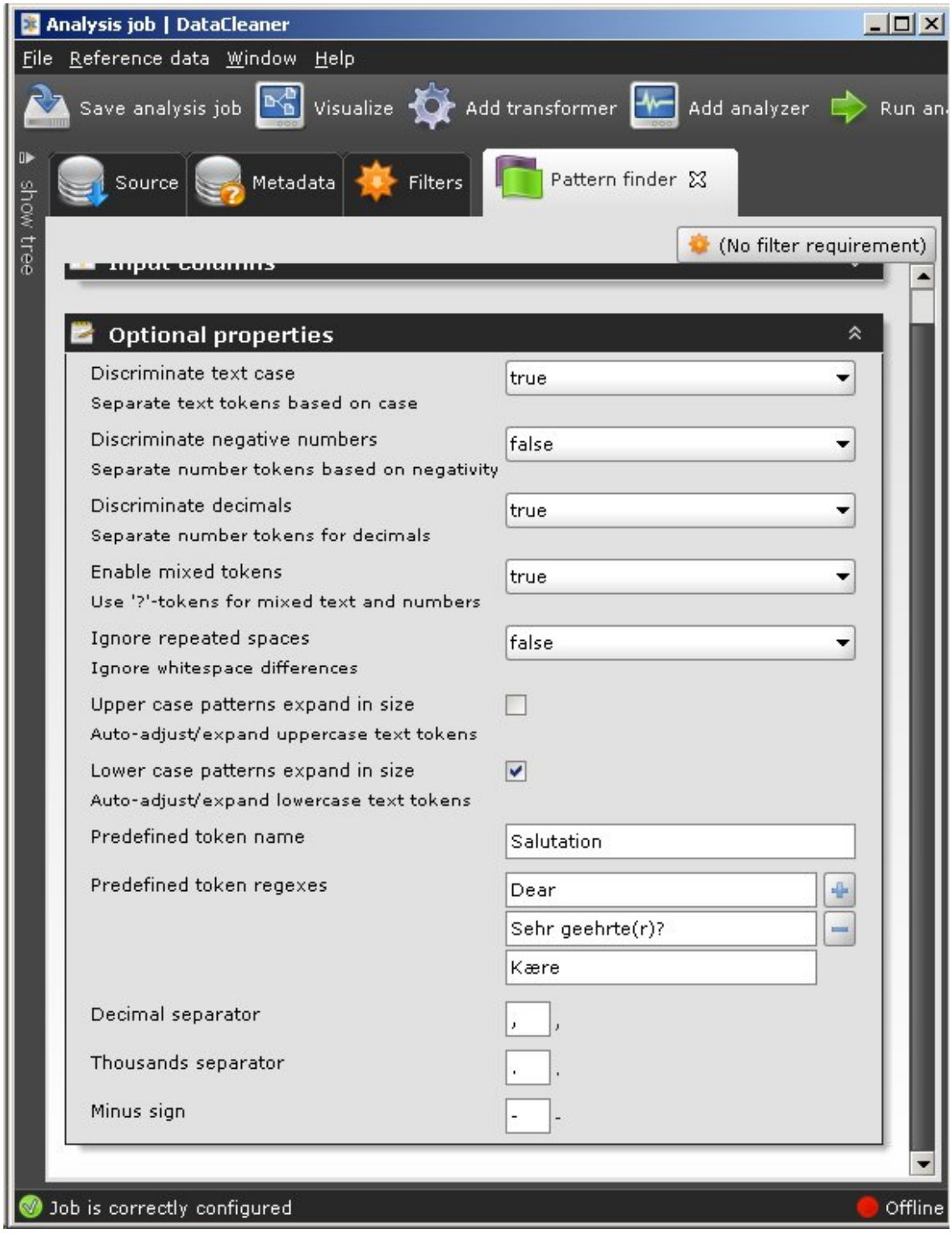
Number analyzer

The number analyzer provides general purpose profiling metrics for numerical column types.

Pattern finder

The pattern finder is one of the more advanced, but also very popular analyzers of DataCleaner.

Here is a screenshot of the configuration panel of the Pattern finder:



From the screenshot we can see that the Pattern finder has these configuration properties:

Table 4.1. Pattern finder properties

Property	Description
Group column	Allows you to define a pattern group column. With a pattern group column you can separate

Property	Description
	the identified patterns into separate buckets/groups. Imagine for example that you want to check if the phone numbers of your customers are consistent. If you have an international customer based, you should then group by a country column to make sure that phone patterns identified are not matched with phone patterns from different countries.
Discriminate text case	Defines whether or not to discriminate (ie. consider as different pattern parts) based on text case. If true "DataCleaner" and "datacleaner" will be considered instances of different patterns, if false they will be matched within same pattern.
Discriminate negative numbers	When parsing numbers, this property defines if negative numbers should be discriminated from positive numbers.
Discriminate decimals	When parsing numbers, this property defines if decimal numbers should be discriminated from integers.
Enable mixed tokens	Defines whether or not to categorize tokens that contain both letters and digits as "mixed", or alternatively as two separate tokens. Mixed tokens are represented using questionmark ('?') symbols. This is one of the more important configuration properties. For example if mixed tokens are enabled (default), all these values will be matched against the same pattern: foo123, 123foo, foobar123, foo123bar. If mixed tokens are NOT enabled only foo123 and foobar123 will be matched (because 123foo and foo123bar represent different combinations of letter and digit tokens).
Ignore repeated spaces	Defines whether or not to discriminate based on amount of whitespaces.
Upper case patterns expand in size	Defines whether or not upper case tokens automatically "expand" in size. Expandability refers to whether or not the found patterns will include matches if a candidate has the same type of token, but with a different size. The default configuration for upper case characters is false (ie. ABC is not matched with ABCD).
Lower case patterns expand in size	Defines whether or not lower case tokens automatically "expand" in size. As with upper case expandability, this property refers to whether or not the found patterns will include matches if a candidate has the same type of token, but with a different size. The default configuration for lower case characters is true (ie. 'abc' is not matched with 'abc').

Property	Description
	The defaults in the two "expandability" configuration properties mean that eg. name pattern recognition is meaningful: 'James' and 'John' both pertain to the same pattern ('Aaaaa'), while 'McDonald' pertain to a different pattern ('AaAaaaaa').
Predefined token name	Predefined tokens make it possible to define a token to look for and classify using either just a fixed list of values or regular expressions. Typically this is used if the values contain some additional parts which you want to manually define a matching category for. The 'Predefined token name' property defines the name of such a category.
Predefined token regexes	Defines a number of string values and/or regular expressions which are used to match values against the (pre)defined token category.
Decimal separator	The decimal separator character, used when parsing numbers
Thousand separator	The thousand separator character, used when parsing numbers
Minus sign	The minus sign character, used when parsing numbers

String analyzer

The string analyzer provides general purpose profiling metrics for string column types. Of special concern to the string analyzer is the amount of words, characters, special signs, diacritics and other metrics that are virtal to understanding what kind of string values occur in the data.

Value distribution

The value distribution (often also referred to as 'Frequency analysis') allows you to identify all the values of a particular column. Furthermore you can investigate which rows pertain to specific values.

Here are the configuration properties for the value distribution analyzer:

Table 4.2. Value distribution properties

Property	Description
Group column	Allows you to define a column for grouping the result. With a group column you can separate the identified value distributions into separate buckets/ groups. Imagine for example that you want to check if the postal codes and city names correspond or if you just want to segment your value distribution on eg. country or gender or ...
Record unique values	By default all unique values will be included in the result of the value distribution. This can potentially

Property	Description
	cause memory issues if your analyzed columns contains a LOT of unique values (eg. if it's a unique key). If the actual unique values are not of interest, then uncheck this checkbox to only count (but not save for inspection) the unique values.
Top n most frequent vales	An optional number used if the analysis should only display eg. the "top 5 most frequent values". The result of the analysis will only contain top/bottom n most frequent values, if this property is supplied.
Bottom n most frequent values	An optional number used if the analysis should only display eg. the "bottom 5 most frequent values". The result of the analysis will only contain top/bottom n most frequent values, if this property is supplied.

Weekday distribution

The weekday distribution provides a frequency analysis for date columns, where you can easily identify which weekdays a date field represents.

Chapter 5. Transformations

With transformations (accessible through the 'Transform' button in the main window) you can pre- and postprocess your data as part of your DQ project.

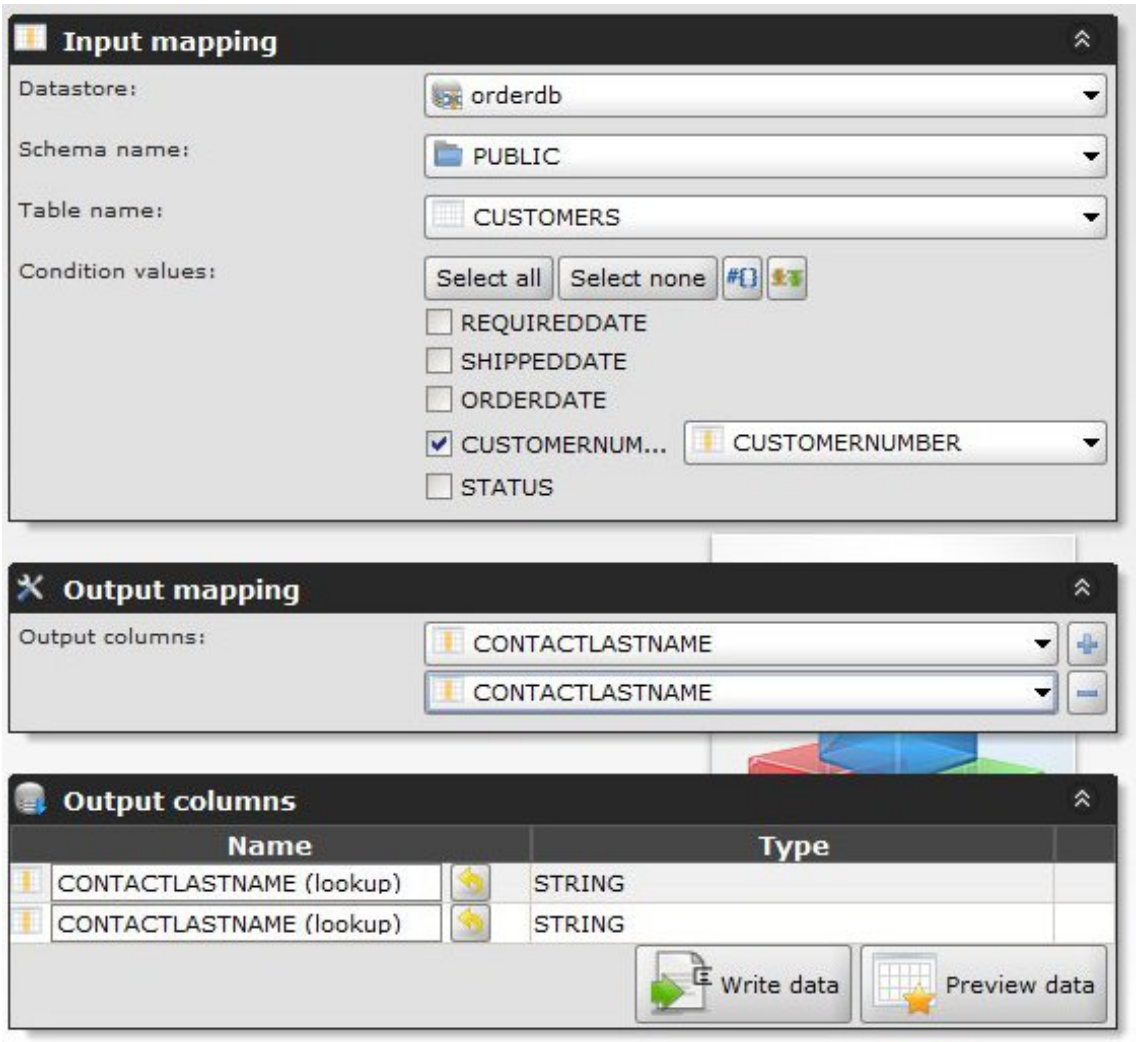
Technically speaking there are two kinds of transformations: Transformers and Filters. Transformers are used to extract, generate or refine data (new columns and sometimes also new rows), whereas filters are used to limit the dataset. In previous version (2.0 - 2.4) of DataCleaner filters and transformers were completely separated concepts, also in the user interface.

There's quite a lot of transformations available in DataCleaner, more than will be feasible to describe all in detail. This chapter provides a documentation for some of the essential ones.

Table lookup

The table lookup transformer allows you to look up values in a different table. Any amount of columns can be used for mapping (lookup conditioning) and for outputting (the outcome of the lookup).

The configuration screen for the table lookup transformer looks like this:



To make the mapping you need to select the target datastore, schema and table names. Once selected you will be able to select which columns to use for condition setting when looking up values.

The semantics of the Table lookup are close to the semantics of a LEFT JOIN. If no lookup value is found, nulls will be returned. However, if multiple records are found to match the conditions, only the first will be returned.

Note that the Table lookup will use a cache for looking up values, to avoid querying the target table for every incoming value.

JavaScript transformer

The JavaScript transformer allows the user to define his/her own script which can perform rather intricate things like conditioning, looping. It can also be used as a useful way to express small business rules.

For this documentation, a complete reference of JavaScript is out of scope. But we will show a few examples and more importantly talk about the available variables and their types.

The JavaScript transformer returns a single string. The entered script should provide this string as the last line of the script. This is why the template script is as follows (so you can just implement the eval() function):

```
function eval() {
   return \"hello \" + values[0];
}
eval();
```

Variables:

Table 5.1. JavaScript variables

Variable	Description

values	An array of all values in the row (as mapped by the "Columns" property).
	
	Using "values" you can reference eg. the first and third values like this:
	
	`var first = values[0];`
	`var third = values[2];`
	
	Note that JavaScript arrays are 0-based.
	
	Instead of indexes you can also reference by column name, like this:
	
	`var idValue = values["id"];`

column_name *	Any column name that is also a valid JavaScript and *not* a reserved variable name will also be added

Variable	Description
	directly to the scope of the script as a variable. For example, if you have two columns, FIRST_NAME and LAST_NAME, you can concatenate them easily, like this: `var fullname = FIRST_NAME + " " + LAST_NAME;`
out	A reference to the system console's "out" stream. If running DataCleaner with the console visible, you can print messages to the console, like this: `out.println("Value: " + values[0]);`

log	A reference to the logging subsystem. Logging can be configured and log messages are stored in files, which makes it a bit more flexible than simply using "out". Here's how you write a few log messages with varying severities:
	`log.debug("This is a DEBUG message, it`
	`log.info("This is a INFO message, it wi`
	`log.warn("This is a WARN message, it wi`
	`log.error("This is a ERROR message, it`

Data types:

Table 5.2. JavaScript data types

Data type	Description

STRING	String values are represented as JavaScript strings, which means that they have (among others) methods like:
	`var str = values[0];`
	`// get the length of a string`
	`var len = str.length();`
	`// uppercase variant of a string`
	`var up = str.toUpperCase();`
	`// lowercase variant of a string`
	`var lw = str.toLowerCase();`
	For more information, we recommend W3 schools JavaScript string reference [http://www.w3schools.com/jsref/jsref_obj_string.asp] .

Data type	Description

NUMBER	Numbers are treated as regular JavaScript numbers, which means that they have (among others) methods and operators like:
	
	`var num = values[0];`
	
	`// format with 2 decimals`
	`var formattedNumber = num.toFixed(2);`
	
	`// add, subtract, multiply or divide`
	`var m = (4 + num * 2 - 1) / 2;`
	
	For more information, we recommend W3 schools JavaScript number reference [http://www.w3schools.com/jsref/jsref_obj_number.asp] and also check out the Math function [http://www.w3schools.com/jsref/jsref_obj_math.asp] reference.

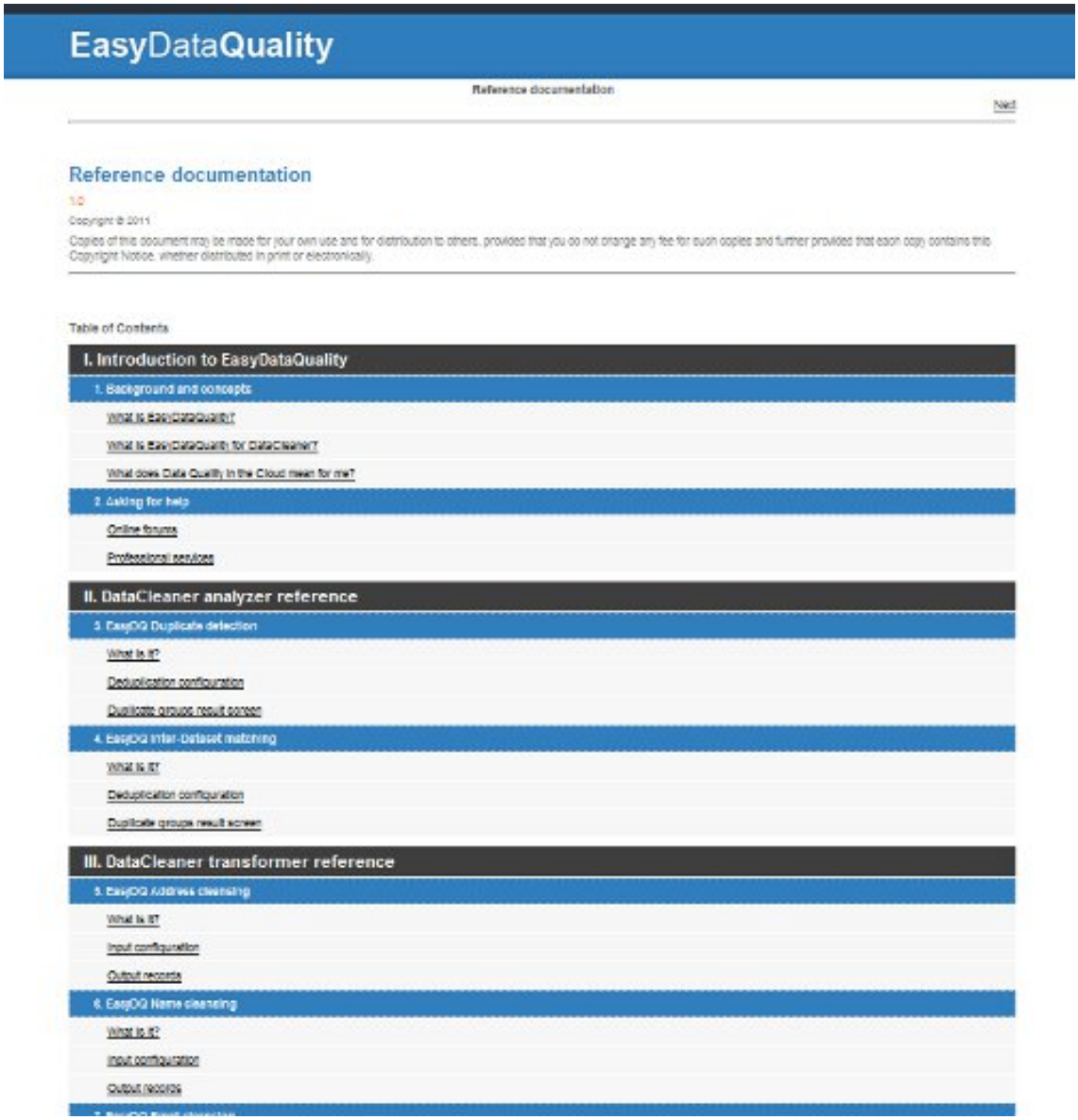
DATE	Date values are treated as Java dates, which is a bit unusual, but leaves you with almost an identical interface as a regular JavaScript date. Here's a summary of typical methods:
	
	`var d = values[0];`
	
	`var year = d.getYear();`
	`var month = d.getMonth();`
	`var date = d.getDate();`
	`var hour = d.getHour();`
	`var minutes = d.getMinutes();`
	`var seconds = d.getSeconds();`
	
	`// milliseconds since 1970-01-01`
	`var timestamp = d.getTime();`
	
	For a full reference, please look at the Java Date class reference [http://download.oracle.com/javase/6/docs/api/java/util/Date.html] .

BOOLEAN	Boolean (true/false) values are simply booleans, no sugar coating added :)

EasyDQ services

EasyDataQuality [http://www.easydq.com] is an on-demand service for data quality functions. DataCleaner provides access to the EasyDQ services, but the core functionality is provided by Human Inference.

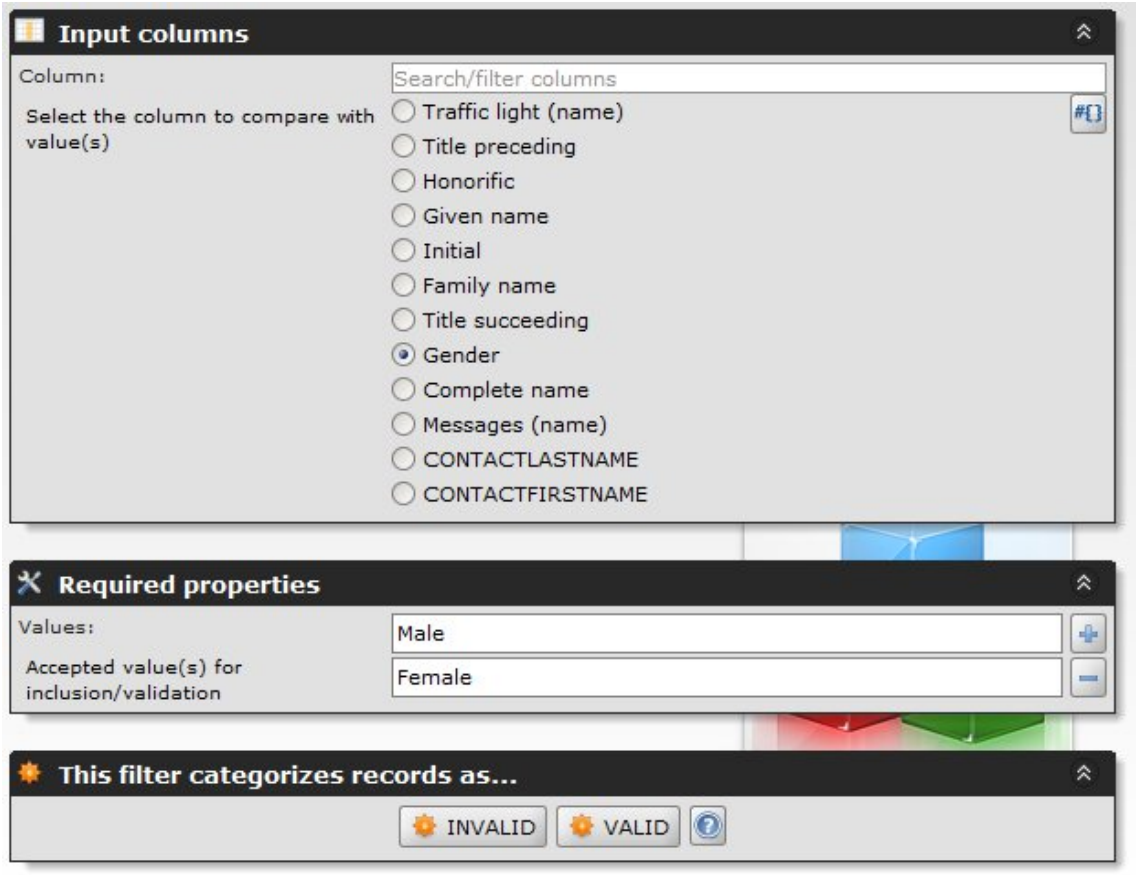
Please refer to the EasyDQ for DataCleaner documentation [http://help.easydq.com/datacleaner] for detailed information about the services provided through EasyDQ.



Equals

The *Equals* filter provides a way to make a simple filtering condition based on a white list / valid list of values. Simply enter a list of values that you accept for a given column, and then you can map your flow to the VALID outcome of the filter.

Here's an example of an Equals filter configuration where valid Gender values are being checked.



Use the plus/minus buttons to grow or shrink the list of values you want to accept.

If placed as the first component in a flow, the Equals filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.

Max rows

The *Max rows* filter is used to limit the amount of records that are passed further on in the job's flow.

If placed as the first component in a flow, the Max rows filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.

Not null

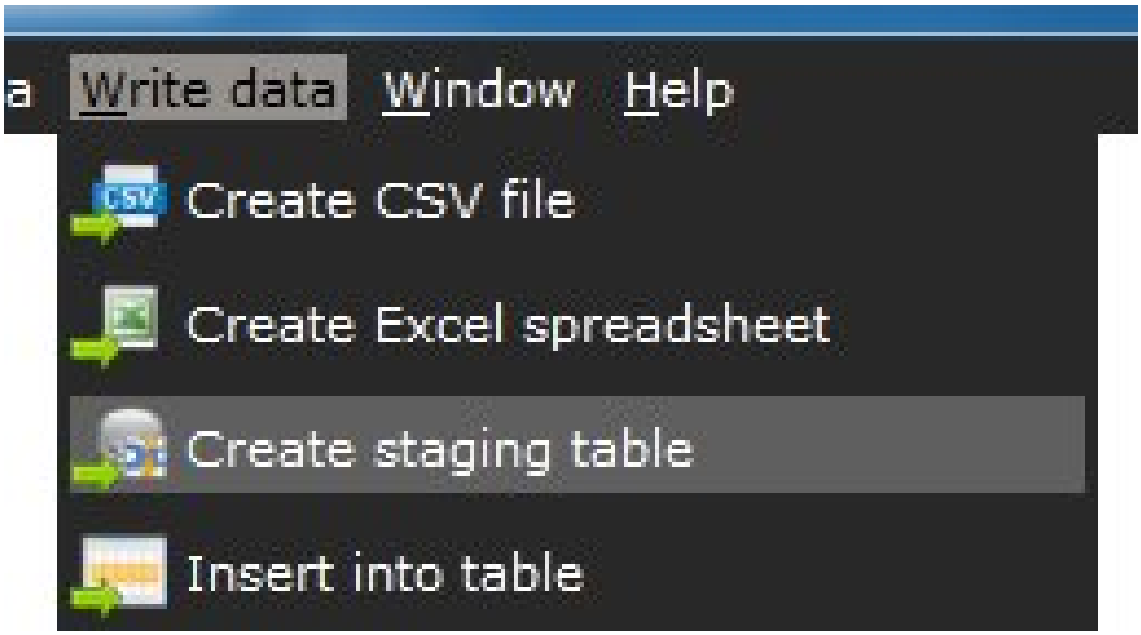
The *Not null* filter is a simple filter that can be used to exclude null values from your flow. Additionally you can select whether or not you want to accept empty strings ("") or not.

If placed as the first component in a flow, the Not null filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.

Chapter 6. Writers

Although the primary focus of DataCleaner is analysis, often during such analysis you will find yourself actually improving data by means of applying transformers and filters on it. When this is the case, obviously you will want to export the improved/cleansed data so you can utilize it in other situations than the analysis.

Writing cleansed data is archieved by wiring the job together with an *output writer* instead of (or in addition to) an analyzer. Adding an output writer is done by selecting it from the "Write data" menu item. Alternatively by clicking either your filter's outcome buttons or your transformer's "Save transformed data" button. When writing data there's a couple of available output formats:



In the following sections each output format option will be described:

Create CSV file

Writes a data set to an Comma Separated Values file. CSV files are a popular choise for interoperability with other systems and loading of data into databases.

Create Excel spreadsheet

Writes a data set to an Excel spreadsheet. An advantage of this approach is that a single file can contain multiple sheets, and that it is easily navigatable in Microsoft Excel. A disadvantage is that for very large data sets it is less performant.

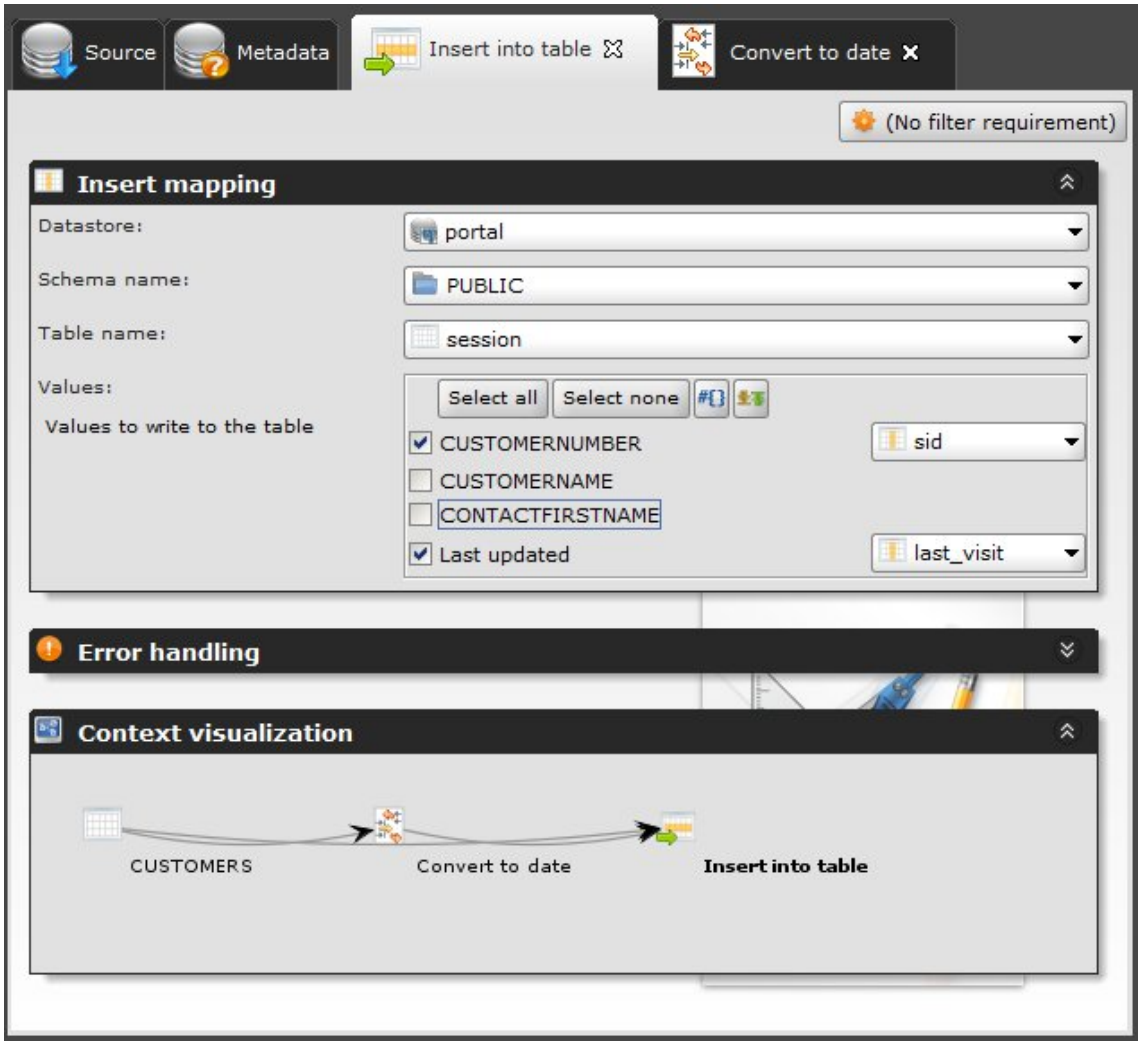
Create staging table

Writes a data set to an embedded relational database, which DataCleaner manages. This option is primarily used for staging data for further analysis. The advantage of using the feature is that it retains column type information, it can handle a lot of data and multiple data sets can be written to the same datastore. A disadvantage is that the data is not easily readable by third party applications (unless exported again).

Insert into table

Using this writer you can insert your data into a table of an existing datastore. If you already have a table layout ready or if you want to append to eg. a database table, then this writing option is the right one for you.

Optionally, you can make the 'Insert into table' component truncate your table before insertion. This will delete all existing records in the table, useful for *initial load* situations.



Currently target tables can be from any of the following datastore types:

1. *CSV file* . In this case data will be appended to the file.
2. *Excel spreadsheet* . In this case data will be appended to the file.
3. *Relational database* . In this case data will be inserted to the table using an INSERT statement.
4. *MongoDB database* . In this case data will be inserted into the MongoDB collection.
5. *CouchDB database* . In this case data will be inserted into the CouchDB database.

Part III. Reference data

Table of Contents

Chapter 7. Dictionaries

Dictionaries are reference data lists used for verifying or categorizing values against certain black- or whitelists. Dictionaries are generally enumerable and finite, whereas eg. string patterns are dynamic and evaluated each time.

Examples of meaningful dictionaries are:

1. A dictionary of product types like "jewelry", "menswear", "sportswear" etc.
2. A dictionary of gender symbols like "M", "F" and maybe "UNKNOWN".
3. A dictionary of age group names (eg. infant, child, young, mature, senior)
4. Two dictionaries for male and female given names (in order to determine gender of persons)

Chapter 8. Synonyms (aka. Synonym catalogs)

Synonym catalogs are used to replace and standardize values to their master terms, in order to avoid multiple terms for the same real world thing.

There are many real life examples of synonyms that make for messy data, for example:

1. Company and brand names, like "Coca-Cola", "Coca cola" and "Coke".

2. Titles, like "Doctor", "Dr." and "Doc"

In the following sections we will describe how to set up synonym catalogs that can be used in a variety of ways to standardize your database.

Text file synonym catalog

A text file synonym catalog is the easiest and often also the fastest way to perform synonym replacement. Simply create a text file with content in a format, where the master term is succeeded with a comma-separated list of synonyms, like this:

```
M,Male,Man,Guy,Boy
F,Female,Woman,Girl
```

In the above example, most typical gender tokens will be replaced with either "M" or "F".

Datastore synonym catalog

If your synonyms are located in a database or another type of datastore, then you can also create synonym catalogs based on this.

Datastore synonym catalogs allow you to specify a single master term column and multiple synonym columns. The synonym catalog will look then find synonym matches by searching/querying the datastore.

Chapter 9. String patterns

String patterns define a "template" for string values which they may or may not conform to.

DataCleaner currently supports two type of popular string formats:

1. *Regular expressions* , which is a general purpose string pattern matching language popular in computer science. Regular expressions does take a bit of time to learn, but are very powerful once harnessed.

   Explaining the syntax of regular expressions is definately outside the scope of the DataCleaner documentation. We recommend the Java Regular Expressions lesson [http://docs.oracle.com/javase/tutorial/essential/regex/] if you are looking for a resource on this.

2. *Simple string patterns* , which use the same syntax as the Pattern finder analyzer. Patterns such as "aaaa@aaaa.aaa" could for example be used to match typical email addresses.

Part IV. Configuration reference

Table of Contents

Chapter 10. Configuration file

XML schema

The configuration file (conf.xml) is an XML file pertaining to the XML namespace "http://eobjects.org/analyzerbeans/configuration/1.0".

For XML-savvy readers, who prefer to use XML schema aware editors to edit their XML files, you can find the XML schema for this namespace here: http://eobjects.org/svn/AnalyzerBeans/trunk/src/main/resources/configuration.xsd [http://eobjects.org/svn/AnalyzerBeans/trunk/src/main/resources/configuration.xsd] .

Datastores

Datastores can be configured in the configuration file under the element <datastore-catalog>. The following sections will go into further details with particular types of datastores.

Database (JDBC) connections

Here are a few examples of common database types.

Tip

The DataCleaner User Interface makes it a lot easier to figure out the url (connection string) and driver class part of the connection properties. It's a good place to start if you don't know these properties already.

MySQL

```
<jdbc-datastore name="MySQL datastore">
 <url>jdbc:mysql://hostname:3306/database?defaultFetchSize=-2147483648</url>
 <driver>com.mysql.jdbc.Driver</driver>
 <username>username</username>
 <password>password</password>
</jdbc-datastore>
```

Oracle

```
<jdbc-datastore name="Oracle datastore">
 <url>jdbc:oracle:thin:@hostname:1521:sid</url>
 <driver>oracle.jdbc.OracleDriver</driver>
 <username>username</username>
 <password>password</password>
</jdbc-datastore>
```

Microsoft SQL Server

A typical connection to Microsoft SQL server will look like this:

```
<jdbc-datastore name="MS SQL Server datastore">
 <url>jdbc:jtds:sqlserver://hostname/database;useUnicode=true;characterEncoding
 <driver>net.sourceforge.jtds.jdbc.Driver</driver>
 <username>username</username>
 <password>password</password>
</jdbc-datastore>
```

However, if you want to use an instance name based connection, then the SQL Server Browser service MUST BE RUNNING and then you can include the instance parameter: Here's an example for connecting to a SQLEXPRESS instance:

```
 <url>jdbc:jtds:sqlserver://hostname/database;instance=SQLEXPRESS;useUnicode=tr
```

Comma-Separated Values (CSV) files

This is an example of a CSV file datastore

```
<csv-datastore name="my_csv_file">
 <filename>/path/to/file.csv</filename>
 <quote-char>"</quote-char>
 <separator-char>;</separator-char>
 <encoding>UTF-8</encoding>
 <fail-on-inconsistencies>true</fail-on-inconsistencies>
 <header-line-number>0</header-line-number>
</csv-datastore>
```

Excel spreadsheets

This is an example of a Excel spreadsheet datastore

```
<excel-datastore name="my_excel_spreadsheet">
 <filename>/path/to/file.xls</filename>
</excel-datastore>
```

XML file datastores

Defining XML datastores can be done in both a simple (automatically mapped) way, or an advanced (and more performant and memory effective way).

The simple way is just to define a xml-datastore with a filename, like this:

```
<xml-datastore name="my_xml_datastore">
 <filename>/path/to/file.xml</filename>
</xml-datastore>
```

This kind of XML datastore works find when the file size is small and the hierarchy is not too complex. The downside to it is that it tries to automatically detect a table structure that is fitting to represent the XML contents (which is a tree structure, not really a table).

To get around this problem you can also define your own table structure in which you specify the XPaths that make up your rows and the values within your rows. Here's an example:

```
<xml-datastore name="my_xml_datastore">
 <filename>/path/to/file.xml</filename>
 <table-def>
   <rowXpath>/greetings/greeting</rowXpath>
   <valueXpath>/greetings/greeting/how</valueXpath>
   <valueXpath>/greetings/greeting/what</valueXpath>
 </table-def>
</xml-datastore>
```

The datastore defines a single table, where each record is defined as the element which matches the XPath "/greetings/greeting". The table has two columns, which are represented by the "how" and "what" elements that are child elements to the row's path.

For more details on the XPath expressions that define the table model of XML datastores, please refer to MetaModel's tutorial on the topic [http://metamodel.eobjects.org/example_xml_mapping.html] (MetaModel is the data access library used to read data in DataCleaner).

MongoDB databases

This is an example of a fully specified MongoDB datastore, with an example table structure based on two collections.

```
<mongodb-datastore name="my_mongodb_datastore">
 <hostname>localhost</hostname>
 <port>27017</port>
 <database-name>my_database</database-name>
 <username>user</username>
 <password>pass</password>
 <table-def>
   <collection>company_collection</collection>
   <property>
    <name>company_name</name>
    <type>VARCHAR</type>
   </property>
   <property>
    <name>customer</name>
    <type>BOOLEAN</type>
   </property>
   <property>
    <name>num_employees</name>
    <type>INTEGER</type>
   </property>
   <property>
    <name>address_details</name>
    <type>MAP</type>
   </property>
 </table-def>
 <table-def>
   <collection>person_collection</collection>
```

```
    <property>
     <name>person_name</name>
     <type>VARCHAR</type>
    </property>
    <property>
     <name>birthdate</name>
     <type>DATE</type>
    </property>
    <property>
     <name>emails</name>
     <type>LIST</type>
    </property>
  </table-def>
</mongodb-datastore>
```

If the hostname and port elements are left out, localhost:27017 will be assumed.

If the username and password elements are left out, an anonymous connection will be made.

If there are no table-def elements, the database will be inspected and table definitions will be auto-detected based on the first 1000 documents of each collection.

CouchDB databases

This is an example of a fully specified CouchDB datastore, with an example table structure based on two CouchDB databases.

```
<couchdb-datastore name="my_couchdb_datastore">
 <hostname>localhost</hostname>
 <port>5984</port>
 <username>user</username>
 <password>pass</password>
 <ssl>true</ssl>
 <table-def>
   <database>company_collection</database>
   <field>
    <name>company_name</name>
    <type>VARCHAR</type>
   </field>
   <field>
    <name>customer</name>
    <type>BOOLEAN</type>
   </field>
   <field>
    <name>num_employees</name>
    <type>INTEGER</type>
   </field>
   <field>
    <name>address_details</name>
    <type>MAP</type>
   </field>
 </table-def>
 <table-def>
```

```
      <database>person_collection</database>
      <field>
       <name>person_name</name>
       <type>VARCHAR</type>
      </field>
      <field>
       <name>birthdate</name>
       <type>DATE</type>
      </field>
      <field>
       <name>emails</name>
       <type>LIST</type>
      </field>
    </table-def>
  </couchdb-datastore>
```

If the hostname and port elements are left out, localhost:5984 will be assumed.

If the username and password elements are left out, an anonymous connection will be made.

If the "ssl" element is false or left out, a regular HTTP connection will be used.

If there are no table-def elements, the database will be inspected and table definitions will be auto-detected based on the first 1000 documents of each database.

Reference data

Reference data are defined in the configuration file in the element <reference-data-catalog>.

Task runner

The task runner defines how DataCleaner's will engine will execute the tasks of an analysis job. Typically you shouldn't edit this element. However, here are the two options:

```
<multithreaded-taskrunner max-threads="30" />
```

Defines a multi threaded task runner with a thread pool of 30 available threads. Beware that although 30 might seem like a high number that too small a pool of threads might cause issues because some tasks schedule additional tasks and thus there's a risk of dead lock when thread count is very low.

```
<singlethreaded-taskrunner />
```

Defines a single threaded task runner. On legacy hardware or operating systems this setting will be better, but it will not take advantage of the multi threading capabilities of modern architecture.

Storage provider

The storage provider is used for storing temporary data used while executing an analysis job. There are two types of storage: Large collections of (single) values and "annotated rows", ie. rows that have been marked with a specific category which will be of interest to the user.

To explain the storage provider configuration let's look at the default element:

```
<storage-provider>
 <combined>
  <collections-storage>
   <berkeley-db/>
  </collections-storage>
  <row-annotation-storage>
   <in-memory max-rows-threshold="1000"/>
  </row-annotation-storage>
 </combined>
</storage-provider>
```

The element defines a combined storage strategy.

Collections are stored using berkeley-db, an embedded database by Oracle. This is the recommended strategy for collections.

Row annotations are stored in memory. There's a threshold of 1000 rows which means that if more than 1000 records are annotated with the same category then additional records will not be saved (and thus is not viewable by the user). Most user scenarios will not require more than max. 1000 annotated records for inspection, but if this is really nescesary a different strategy can be pursued:

Using MongoDB for annotated rows

If you have a local MongoDB [http://www.mongodb.org/] instance, you can use this as a store for annotated rows. This is how the configuration looks like:

```
<row-annotation-storage>
 <custom-storage-provider class-name="org.eobjects.analyzer.storage.MongoDbS
</row-annotation-storage>
```

The MongoDB storage provider solution has shown very good performance metrics, but does add more complexity to the installation, which is why it is still considered experimental and only for savvy users.

Chapter 11. Analysis job files

XML schema

Analysis job files are written in an XML format pertaining to the XML namespace "http://eobjects.org/analyzerbeans/job/1.0".

For XML-savvy readers, who prefer to use XML schema aware editors to edit their XML files, you can find the XML schema for this namespace here: http://eobjects.org/svn/AnalyzerBeans/trunk/src/main/resources/job.xsd [http://eobjects.org/svn/AnalyzerBeans/trunk/src/main/resources/job.xsd] .

Chapter 12. Logging

Logging configuration file

Logging in DataCleaner is based on Log4j, an open source logging framework by the Apache foundation. With log4j you can configure logging at a very detailed level, while at the same time keeping a centralized configuration.

There are three approaches to configuring logging in DataCleaner:

1. *The default logging configuration* . This requires no changes to the standard distribution of DataCleaner. Log files will be generated in the log/datacleaner.log file.

2. *Specifying your own XML log configuration* . This requires you to put a file named log4j.xml in the root directory of DataCleaner.

3. *Specifying your own property file log configuration* . This requires you to put a file named log4j.properties in the root directory of DataCleaner.

The recommended way of doing custom configuration of DataCleaner logging is using the XML format. In the following sections we will explain this approach using examples. For more detailed documentation on Log4j configuration, please refer to the Log4j website [http://logging.apache.org/log4j/] .

Default logging configuration

Here's a listing of the default logging configuration, in XML format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

 <appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
  <param name="Target" value="System.out"/>
  <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%-5p %d{HH:mm:ss} %c{1} - %m%n"/>
  </layout>
 </appender>

 <appender name="fileAppender" class="org.apache.log4j.DailyRollingFileAppender
  <param name="File" value="log/datacleaner.log"/>
  <param name="DatePattern" value="'.'yyyy-MM-dd'.log'"/>
  <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%-5p %d{HH:mm:ss.SSS} %c{1} - %m%n"/>
  </layout>
 </appender>

 <logger name="org.eobjects.metamodel">
  <level value="info" />
 </logger>

 <logger name="org.eobjects.analyzer">
```

```
 <level value="info" />
</logger>

<logger name="org.eobjects.analyzer.job.runner">
 <level value="info" />
</logger>

<logger name="org.eobjects.analyzer.storage">
 <level value="info" />
 </logger>

<logger name="org.eobjects.analyzer.descriptors.ClasspathScanDescriptorProvide
 <level value="info" />
</logger>

<logger name="org.eobjects.datacleaner">
 <level value="info" />
</logger>

<root>
 <priority value="info" />
 <appender-ref ref="consoleAppender" />
 <appender-ref ref="fileAppender" />
</root>

</log4j:configuration>
```

This logging configuration specifies the INFO level as the default level of logging. It appends (outputs) log messages to the console (if available) and to a file with the path: log/datacleaner.log

We recommend using this default configuration as a template for custom log configurations. Next we will explore how to modify the configuration and create new logging outputs.

Modifying logging levels

These are the logging levels available in DataCleaner and Log4j, order by priority (highest priority first):

1. error
2. warn
3. info
4. debug
5. trace

Typically the bottom-two logging levels (debug and trace) are not used unless unexpected situations has to be investigated by developers.

Modifying the logging levels can be done either globally or in a hierarical manner:

1. If you change the <priority> element's value attribute, you change the global threshold for logging messages.

2. If you change the <logger> element's level, you change the logging priority logging messages that pertain to a particular hierachy of loggers.

Alternative logging outputs

Log messages are printed to an output, typically a file or the console. In the configuration file this is configured in the <appender> elements. Here's a few examples of alternative appenders you can use. For more examples and documentation, please refer to the Log4j website [http://logging.apache.org/log4j/] .

Logging in a PostgreSQL database:

```
<appender name="jdbcAppender" class="org.apache.log4j.jdbc.JDBCAppender">
 <param name="URL" value="jdbc:postgresql:db"/>
 <param name="Driver" value="org.postgresql.Driver"/>
 <param name="User" value="user"/>
 <param name="Password" value="password"/>
 <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern"
     value="INSERT INTO log4j (log_date,log_level,log_location,log_message) VAL
 </layout>
</appender>
```

Part V. DataCleaner monitor repository

Table of Contents

Chapter 13. Repository location

Abstract

In this chapter we will explain configuration of the repository of the DataCleaner monitor web application. By default the repository and other artifacts are bundled with the application, but for production deployments this configuration may not be sufficient. Learn about how to deploy a repository that is located independently of the web application code.

Configure repository location

By default DataCleaner monitor web application uses a file-based repository location which is relative to the deployed web archive. This makes it easy to deploy and test-drive, but it might not be the best production deployment choice.

To change the repository location, find the file *WEB-INF/classes/context/repository-context.xml* within the deployed web archive folder. Find the <bean> element which initially looks like this:

```
<bean name="repository" class="org.eobjects.datacleaner.repository.file.FileRep
  <constructor-arg type="java.io.File" value="repository"/>
</bean>
```

You can modify the value attribute of the inner <constructor-arg> element to point to a directory location of choice. The location can be absolute or relative to the web archive folder. For instance you might want to use a configuration like this in a unix environment:

```
<bean name="repository" class="org.eobjects.datacleaner.repository.file.FileRep
  <constructor-arg type="java.io.File" value="/var/datacleaner/repository"/>
</bean>
```

Providing signed Java WebStart client files

The DataCleaner monitor web application features an option to let the user launch the desktop application for editing and testing jobs deployed on the monitor server. To enable this special mode of interoperability, signed JAR files needs to be provided, since otherwise the desktop application will not be allowed to launch by most Java runtime configurations.

Signed JAR files can be downloaded separately [http://datacleaner.eobjects.org/downloads] and should be extracted into a directory of choice on the server. Once extracted you need to configure the application with the directory path. Find the file *WEB-INF/classes/context/repository-context.xml* within the deployed web archive folder. Remove or comment the <bean> element which initially looks like this:

```
<bean name="launchArtifactProvider" lazy-init="true"
   class="org.eobjects.datacleaner.monitor.server.DevModeLaunchArtifactProvider
```

It needs to be replaced with a new <bean> element which will look like this:

```
<bean name="launchArtifactProvider"
```

```
    lazy-init="true" class="org.eobjects.datacleaner.monitor.server.FileFolderL
  <constructor-arg type="java.io.File" value="/var/datacleaner/signed_jars"/>
</bean>
```

Chapter 14. Repository layout

Abstract

In this chapter we look at the file and folder layout of a DataCleaner monitor repository. Beginning with the multi-tenant layout, and then proceeding with a typical tenant's repository layout.

Multi-tenant layout

The DataCleaner repository layout, used by the monitoring web application, is built to support multi-tenant deployments. Therefore, on the root level of the repository, folders are located which each represent a tenant's separate home folder. Users from one tenant are not able to access files or folders from other tenant's home folders.

Tenant home layout

To function properly, each tenant home folder requires these files and folders:

1. conf.xml (file)
2. jobs (folder)
3. results (folder)
4. timelines (folder)

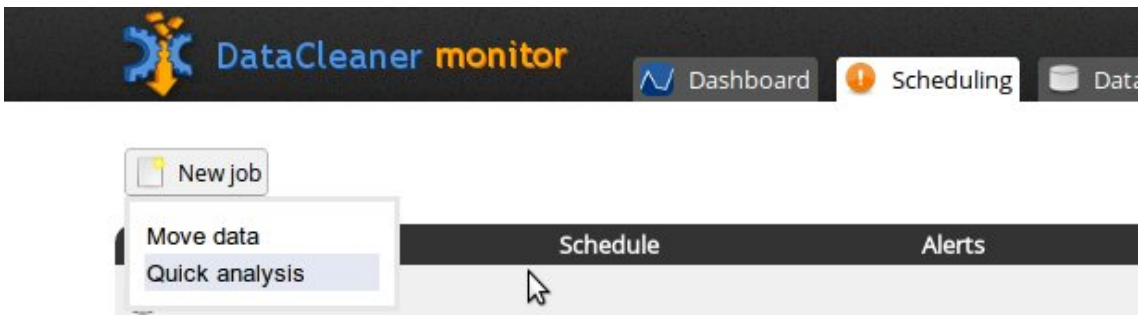
The *conf.xml* file represents the DataCleaner configuration for the particular tenant. The file format is the same as described in the Configuration file chapter. It is recommended to use the supplied example conf.xml file (for the 'DC' tenant) as a template for further customization. Specifically the custom elements for task-runner, descriptor-provider and storage-provider in this template conf.xml file is recommended for optimal performance.

The folders are all *managed* by the DataCleaner monitoring web application, so only in rare cases should you manually interact with them.

It is allowed to add more files and folders to the tenant home. These will not be managed by the monitor application, but can be referenced eg. as the filename paths of datastores defined in conf.xml.

Adding a new job to the repository

There are multiple ways to add a new job to the repository. The most fail-safe way is to use one of the job wizards, provided by the DataCleaner monitoring web application. These are found on the 'Scheduling' page, using the "New job" button.



Alternatively, you can create the job using the desktop application. In that case, make sure that the *name of any involved datastore in the desktop application matches the name of the datastore in the repository!* If so, you can safely drop the .analysis.xml job file in the *jobs* folder on the repository. It will be immediately available in the web UI of the monitor application.

Part VI. DataCleaner monitor web services

Table of Contents

Chapter 15. Job triggering

Chapter 16. Repository navigation

Abstract

In this chapter we will learn how to use web service URLs to navigate the DataCleaner repository. The repository is available at the root level of the DataCleaner monitoring web application. Access is of course restricted to the tenant's home folder. Therefore, all these web services are located with URLs starting with the form:

/DataCleaner-monitor/repository/DC/...

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

Job files

Job files are available in the reserved folder name 'jobs'. To get a listing of all job files (in JSON format), go to:

```
/DataCleaner-monitor/repository/DC/jobs
```

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

The result will resemble this example result:

```
[
  {"repository_path":"/DC/jobs/Contributor name cleansing.analysis.xml",
    "name":"Contributor name cleansing","filename":"Contributor name cleansing.
  {"repository_path":"/DC/jobs/Customer completeness.analysis.xml",
    "name":"Customer completeness","filename":"Customer completeness.analysis.x
  {"repository_path":"/DC/jobs/Customer duplicates.analysis.xml",
    "name":"Customer duplicates","filename":"Customer duplicates.analysis.xml"}
  {"repository_path":"/DC/jobs/product_profiling.analysis.xml",
    "name":"product_profiling","filename":"product_profiling.analysis.xml"}
]
```

Further navigating to one of these repository paths, you will be able to read the full XML description of the job, as described in the Analysis Job files chapter:

```
/DataCleaner-monitor/repository/DC/jobs/product_profiling.analysis.xml
```

Where 'DC' is the tenant identifier, 'product_profiling' is the name of the job and 'DataCleaner-monitor' is the web application archive name.

Result files

Result files are available in the reserved folder name 'results'. To get a listing of all job files (in JSON format), go to:

```
/DataCleaner-monitor/repository/DC/results
```

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

The result will resemble this example result:

```
[
  {"repository_path":"/DC/results/Customer completeness-1345128427583.analysis.
    "filename":"Customer completeness-1345128427583.analysis.result.dat"},
  {"repository_path":"/DC/results/Customer completeness-1345200106074.analysis.
    "filename":"Customer completeness-1345200106074.analysis.result.dat"}
]
```

Further navigating to one of these repository paths, you will be able to inspect the analysis result, rendered in HTML format.

```
/DataCleaner-monitor/repository/DC/results/Customer completeness-1345200106074.
```

Where 'DC' is the tenant identifier, 'Customer completeness' is the name of the job, '1345200106074' is the timestamp of the result and 'DataCleaner-monitor' is the web application archive name.

Chapter 17. Atomic transformations (data cleaning as a service)

Abstract

In this chapter we will learn how to use the DataCleaner monitor as a web service for invoking DataCleaner jobs as on-demand / on-the-fly data cleansing operations. Like in the case of repository navigation, these web services are located with URLs starting with the form:

/DataCleaner-monitor/repository/DC/...

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

What are atomic transformation services?

DataCleaner jobs normally operate on batches of records, for example all records in a CSV file. However, any DataCleaner job can also be used to process single records on the fly. This feature is called "atomic transformations". Such transformations could be used as a part of a data processing pipeline in a SOA architecture or in a web application with flexible data processing needs.

Creating an atomic transformation is easy. Any DataCleaner job can be used for performing atomic transformations. Simply create a job that does some kind of transformation (anything from a simple 'String length' operation to a complete chain of 'Address cleansing' and regex parsing), and place the job in the DataCleaner monitor repository. A specialized web service endpoint will be automatically exposed to use the job as an atomic transformation. The endpoint URL will be:

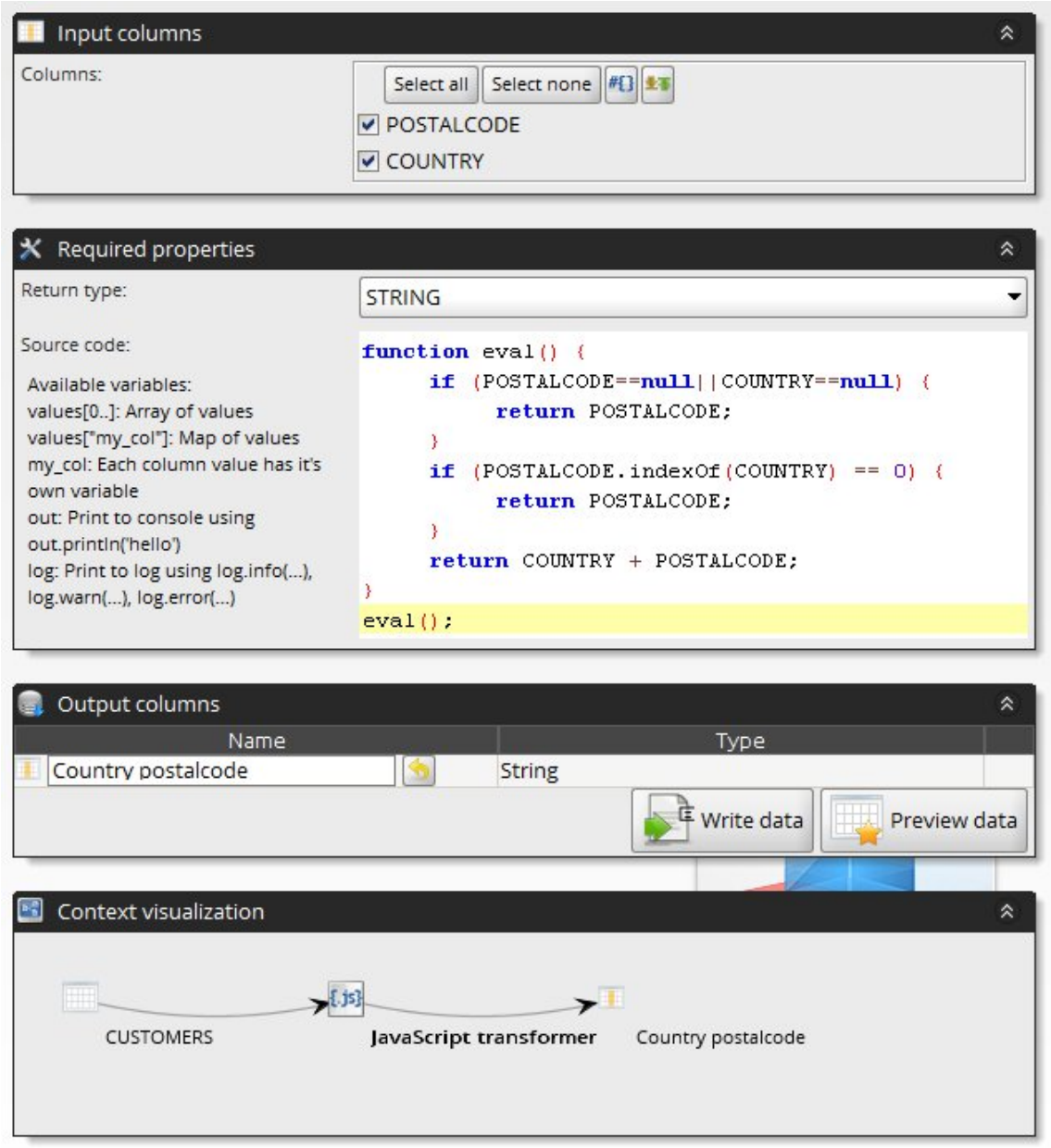
```
/DataCleaner-monitor/repository/DC/jobs/[jobname].invoke
```

Invoking atomic transformations

The contract of the atomic transformation invocation service is dynamic, based on the contract defined by source data and transformed data in the job.

The web service is based on JSON data. You need to provide the JSON data corresponding to the source record format of the job.

For instance, let's say a job defines the following JavaScript transformation with two source columns (POSTALCODE and COUNTRY):



The web service will therefore expect that each JSON record contains two values. The service accepts one or multiple records in it's payload. Use a HTTP POST with a body like this (3 records and 2 columns):

```
{"rows":[
  {"values":["2200","DK"]},
  {"values":["2200",""]},
  {"values":["DK2200","DK"]}
]}
```

And the resulting response body will look like this:

```
{
  "rows":[
    {"values":["DK2200"]},
    {"values":["2200"]},
    {"values":["DK2200"]}
  ],
  "columns":[
    "Country postalcode"
  ]
}
```

Note that the 'Content-Type' header of the request is assumed to be 'application/json'.

Part VII. Invoking DataCleaner jobs from the command-line

Table of Contents

Chapter 18. Command-line interface

DataCleaner offers a Command-Line Interface (CLI) for performing various tasks, including executing analysis jobs, via simple commands that can be invoked eg. as a scheduled tasks.

Executables

Depending on your distribution of DataCleaner, you will have one of these CLI executables included:

1. *datacleaner-console.exe* , which is a Windows-only executable.
2. *datacleaner.cmd* , which is a script to start DataCleaner in Windows.
3. *datacleaner.sh* , which is a script to start DataCleaner in Unix-like systems, like Linux and Mac OS.
4. If you're running DataCleaner in Java Webstart mode, then there is no Command-Line Interface!

Usage scenarios

The usage scenarios of DataCleaner's CLI are:

1. Executing an analysis job
2. List registered datastores
3. List schemas in a datastore
4. List tables in a schema
5. List columns in a table
6. List available analyzers, transformers or filters

How these scenarios are attained is revealed by invoking your executable with the *-usage* argument:

```
> datacleaner-console.exe -usage
-conf
(-configuration, --configuration-file) FILE
        : XML file describing the configuration of AnalyzerBeans
-ds (-datastore, --datastore-name) VAL
        : Name of datastore when printing a list of schemas, tables or columns
-job (--job-file) FILE
        : An analysis job XML file to execute
-list [ANALYZERS | TRANSFORMERS | FILTERS | DATASTORES | SCHEMAS |
TABLES | COLUMNS]
        : Used to print a list of various elements available in the configurati
-s (-schema, --schema-name) VAL
        : Name of schema when printing a list of tables or columns
-t (-table, --table-name) VAL
        : Name of table when printing a list of columns
```

Executing an analysis job

Here's how to execute an analysis job - we'll use the bundled example job "employees.analysis.xml":

```
> datacleaner-console.exe -job examples/employees.analysis.xml
SUCCESS!

...

RESULT:
Value distribution for column: REPORTSTO
Top values:
 - 1102: 6
 - 1143: 6
 - 1088: 5
Null count: 0
Unique values: 0


RESULT:
         Match count Sample
Aaaaaaa           22 William
Aaaa Aaa           1 Foon Yue


RESULT:
           Match count Sample
aaaaaaaaaa          23 jfirrelli


RESULT:
                            Match count Sample
Aaaaa Aaa                             17 Sales Rep
AA Aaaaaaaaa                           2 VP Marketing
Aaaa Aaaaaaa (AAAA)                    1 Sale Manager (EMEA)
Aaaaa Aaaaaaa (AA)                     1 Sales Manager (NA)
Aaaaa Aaaaaaa (AAAAA, AAAA)            1 Sales Manager (JAPAN, APAC)
Aaaaaaaaa                              1 President

...
```

As you can see from the listing, the results of the analysis will be printed directly to the command-line output. If you want to save the results to a file, simply use your operating systems built-in functionality to pipe command-line output to a file, typically using the '>' operator.

Listing datastore contents and available components

The Command-Line Interface allows for listing of datastore contents and available components. The intended usage for this is to aid in hand-editing an analysis file, if this is desired. By using the *-list*

arguments you can get the metadata of your datastore and the DataCleaner components that will allow you to manually compose an analysis file.

Listing the contents of a datastore is pretty self-explanatory, if you look at the output of the *-usage* command. Here's a few examples, using the example database 'orderdb':

```
> datacleaner-console.exe -list datastores
Datastores:
-----------
Country codes
orderdb

> datacleaner-console.exe -list tables -ds orderdb
Tables:
-------
CUSTOMERS
CUSTOMER_W_TER
DEPARTMENT_MANAGERS
DIM_TIME
EMPLOYEES
OFFICES
ORDERDETAILS
ORDERFACT
ORDERS
PAYMENTS
PRODUCTS
QUADRANT_ACTUALS
TRIAL_BALANCE

> datacleaner-console.exe -list columns -ds orderdb -table employees
Columns:
--------
EMPLOYEENUMBER
LASTNAME
FIRSTNAME
EXTENSION
EMAIL
OFFICECODE
REPORTSTO
JOBTITLE
```

Listing DataCleaner's components is done by setting the *-list* argument to one of the three component types: ANALYZER, TRANSFORMER or FILTER:

```
> datacleaner-console.exe -list analyzers

...

name: Matching analyzer
- Consumes multiple input columns (type: UNDEFINED)
- Property: name=Dictionaries, type=Dictionary, required=false
- Property: name=String patterns, type=StringPattern, required=false
```

```
name: Pattern finder
- Consumes 2 named inputs
   Input column: Column (type: STRING)
   Input column: Group column (type: STRING)
- Property: name=Discriminate text case, type=Boolean, required=false
- Property: name=Discriminate negative numbers, type=Boolean, required=false
- Property: name=Discriminate decimals, type=Boolean, required=false
- Property: name=Enable mixed tokens, type=Boolean, required=false
- Property: name=Ignore repeated spaces, type=Boolean, required=false
- Property: name=Upper case patterns expand in size, type=boolean, required=fal
- Property: name=Lower case patterns expand in size, type=boolean, required=fal
- Property: name=Predefined token name, type=String, required=false
- Property: name=Predefined token regexes, type=String, required=false
- Property: name=Decimal separator, type=Character, required=false
- Property: name=Thousands separator, type=Character, required=false
- Property: name=Minus sign, type=Character, required=false

...

> datacleaner-console.exe -list transformers

...

name: Tokenizer
- Consumes a single input column (type: STRING)
- Property: name=Delimiters, type=char, required=true
- Property: name=Number of tokens, type=Integer, required=true
- Output type is: STRING
name: Whitespace trimmer
- Consumes multiple input columns (type: STRING)
- Property: name=Trim left, type=boolean, required=true
- Property: name=Trim right, type=boolean, required=true
- Property: name=Trim multiple to single space, type=boolean, required=true
- Output type is: STRING

...
```

Parameterizable jobs

If you want to make a part of a job parameterizable/variable, then it is possible to do so. Currently this is a feature only supported by means of editing the .analysis.xml files though, since the DataCleaner graphical user interface does not store job variables when saving jobs.

In the source section of your job, you can add variables which are key/value pairs that will be referenced throughout your job. Each variable can have a default value which will be used in case the variable value is not specified. Here's a simple example:

```
...

<source>
  <data-context ref="my_datastore" />
  <columns>
```

```
    <column path="column1" id="col_1" />
    <column path="column2" id="col_2" />
  </columns>
  <variables>
    <variable id="filename" value="/output/dc_output.csv" />
    <variable id="separator" value="," />
  </variables>
</source>

...
```

In the example we've defined two variables: *filename* and *separator* . These we can refer to for specific property values, further down in our job:

```
...

<analyzer>
  <descriptor ref="Write to CSV file"/>
  <properties>
    <property name="File" ref="filename" />
    <property name="Quote char" value="&amp;quot;" />
    <property name="Separator char" ref="separator" />
  </properties>
  <input ref="col_1" />
  <input ref="col_2" />
</analyzer>

...
```

Now the property values of the *File* and *Separator char* properties in the *Write to CSV file* have been made parameterizable. To execute the job with new variable values, use *-var* parameters from the command line, like this:

```
DataCleaner-console.exe -job my_job.analysis.xml -var filename=/output/my_file.
```

Dynamically overriding configuration elements

Since version 2.5 of DataCleaner it is possible to override elements in the configuration file dynamically from the command line. This is a feature which can be useful in scenarios where you want the to invoke the same job but with slightly different configuration details.

For example, you might want to reuse the same job to be executed on several similar CSV files, or similar database environments. Let us assume that you have a CSV datastore that is defined like this:

```
</datastore-catalog>
  <csv-datastore name="My csv file">
    <filename>/path/to/file.csv</filename>
  </csv-datastore>
```

```
</datastore-catalog>
```

To override the filename dynamically, you have to specify the property path (datastore catalog, then datastore name, then property name) with a '-D' parameter on the command line. Furthermore any spaces or dashes are removed and the succeeding character is uppercased. In the end it will look like "camelCase" strings, like this:

```
DataCleaner-console.exe ... -DdatastoreCatalog.myCsvFile.filename=anotherfile.csv
```

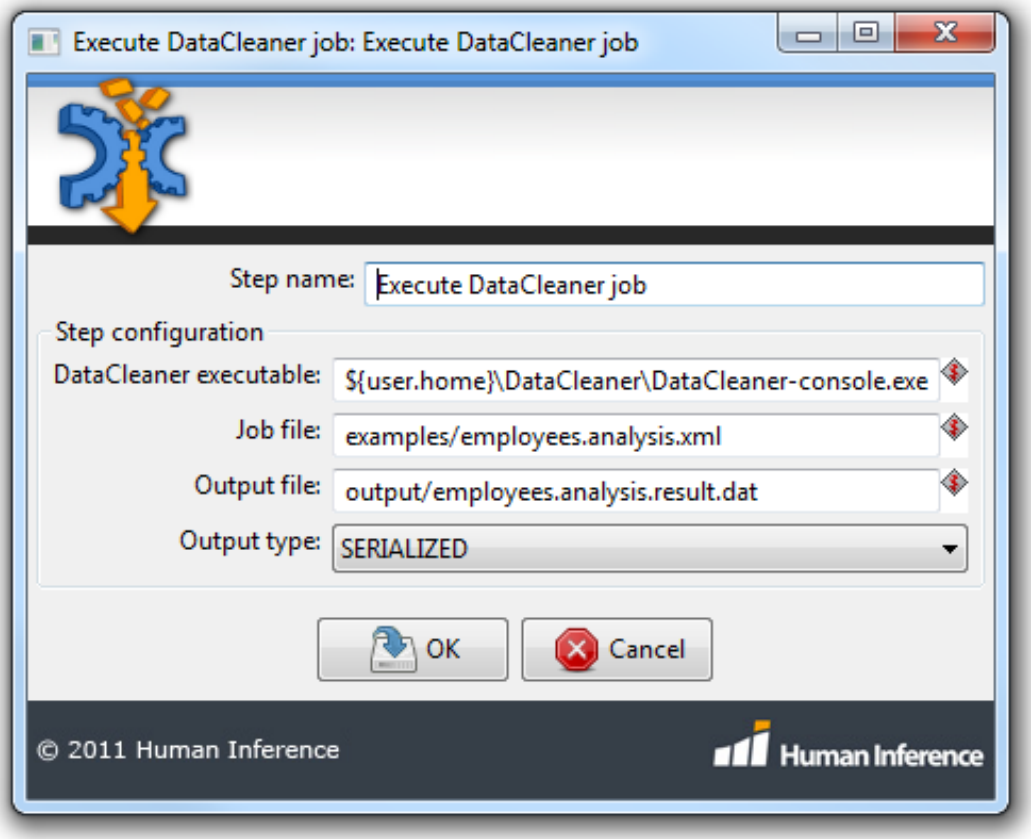
This mechanism can be used for any configuration property within the datastore catalog and reference data catalog.

Chapter 19. Scheduling jobs

Scheduling DataCleaner jobs can be done via any third party scheduler in combination with the command line interface . Typical options are:

1. *Pentaho Data Integration job entry* . If you want to have DataCleaner scheduled and integrated into an environment where you can eg. iterate over files in a folder etc., then you can use Pentaho Data Integration (PDI), which is an open source ETL tool that includes a scheduler.

   Construct a PDI "job" (ie. not a "transformation") and add the DataCleaner job entry. The configuration dialog will look like this:



   The most tricky part is to fill out the executable and the job filename. Note that all configuration options can contain PDI variables, like it is the case with ${user.home} in the screenshot above. This is useful if you want to eg. timestamp your resulting files etc.

2. *Cron jobs* . Typically on Linux and other Unix variants. Simply configure a cron entry which invokes datacleaner.sh with the desired parameters.

3. *Scheduled tasks* . On Windows based machines, using "Scheduled tasks" (aka "Task Scheduler" on newer versions of Windows) can be used similarly to cron jobs.

Part VIII. Developer's guide

Table of Contents

Chapter 20. Architecture

The architecture of DataCleaner can be described from different angles depending on the topic of interest. In the following sections we cover different aspects of the DataCleaner architecture.

Data access

In DataCleaner all sources of data are called 'datastores'. This concept covers both sources that are read/parsed locally and those that are 'connected to', eg. databases and applications. Some datastores can also be written to, for instance relational databases.

DataCleaner uses the MetaModel framework [http://metamodel.eobjects.org] for data access. From DataCleaner's perspective, MetaModel provides a number of features:

1. A common way of interacting with different datastores.
2. A programmatic query syntax that abstracts away database-specific SQL dialects, and that is usable also for non-SQL oriented datastores (files etc.).
3. Out-of-the-box connectivity to a lot of sources, eg. CSV files, relational databases, Excel spreadsheets and a lot more.
4. A framework for modelling new sources using the same common model.

DataCleaners datastore model is also extensible in the way that you can yourself implement new datastores in order to hook up DataCleaner to legacy systems, application interfaces and more. For more information refer to the Developer resources chapter.

Processing framework

For processing data, DataCleaner builds upon the AnalyzerBeans framework [http://analyzerbeans.eobjects.org] framework, which is a closely related, but independent project. The goal of AnalyzerBeans is to provide a performant and extensible framework for batch-analyzing data.

AnalyzerBeans is different from most conventional data processing frameworks in that you do not specify a specific order of events, but the framework *infers* it. If a step (aka. a component) in the flow requires input from another step, then the flow is automatically layed out so that the dependent step is executed before the other. Please refer to the section on wiring components together for an walkthrough on how this works in practice.

One of the key characteristics of AnalyzerBeans is that it is built for massively parallel processing. Whereas most data processing frameworks archieve parallelism by the use of parallel execution of *each step* in a flow, the AnalyzerBeans framework executes the processing of *each record* in parallel. This means that any CPU utilization is drastically improved since unequal processing time of the steps in a processing chain does not cause in bottlenecks! The downside to this approach is that the order of the processed records cannot be guaranteed, but this is only very rarely required in the domain of data profiling and analysis, and if it is required there are technical workarounds to apply.

To further optimize performance, the AnalyzerBeans framework also allows certain components to modify the source query of the processing chain! This is a mechanism known as "push down query optimization", and is only applicable in scenarios where all succeeding components depend on a specific condition. But in those cases (for instance a filter specifying that a field is NOT NULL), then it can drastically improve the time needed to process all records. For more information on this principle, please read the

blog entry ' Push down query optimization in DataCleaner [http://kasper.eobjects.org/2011/12/push-down-query-optimization-in.html] ' by Kasper Sørensen.

Chapter 21. Developer resources

Extension development tutorials

There are many useful resources for those who engage in developing extensions (aka. plugins / add-ons) to DataCleaner. To help you on your way, here's a list of useful links. If you think this list is missing a link, please let us know:

1. Webcast: Developing DataCleaner extensions [http://datacleaner.eobjects.org/media]
2. Tutorial: Developing a transformer [http://kasper.eobjects.org/2010/09/developing-value-transformer-using.html]
3. Tutorial: Developing an analyzer [http://kasper.eobjects.org/2010/09/developing-analyzer-using-analyzerbeans.html]
4. Tutorial: Implementing a custom datastore [http://kasper.eobjects.org/2012/04/implementing-custom-datastore-in.html]
5. Forum: How to install plugins in DataCleaner [http://datacleaner.eobjects.org/topic/154/How-to-install-plugins-in-DataCleaner]
6. Javadoc: DataCleaner-core [http://eobjects.org/datacleaner/apidocs/current/]
7. Javadoc: AnalyzerBeans [http://analyzerbeans.eobjects.org/apidocs/]
8. Javadoc: MetaModel [metamodel.eobjects.org/apidocs/]

Issue tracking

The development of DataCleaner is driven primarily by issues registered in our issue tracking system, *trac* . In trac every member of the community has the possibility to register, monitor and update issues ('tickets') that he/she prioritizes.

http://eobjects.org/trac

When submitting code in Subversion, it is recommended to prefix the commit remarks with the ticket id, so that we can track issues and their corresponding patches.

Building DataCleaner

Check out the source code for AnalyzerBeans and DataCleaner from subversion:

```
> svn checkout http://eobjects.org/svn/AnalyzerBeans/trunk/ AnalyzerBeans
> svn checkout http://eobjects.org/svn/DataCleaner/trunk/ DataCleaner
```

Build the projects:

```
> cd AnalyzerBeans
```

```
> mvn clean install
> cd ../DataCleaner
> mvn clean install
```

Run DataCleaner

```
> cd ../DataCleaner/DataCleaner-packaging/target
> java -jar DataCleaner.jar
```

Chapter 22. Extension packaging

DataCleaner extensions are packages of added functionality, written in Java. To correctly package an extension, this chapter will walk through the details.

Annotated component

The main principle behind extension discovery in DataCleaner is annotated classes. These are the annotations that will work to activate components within your extension:

1. @AnalyzerBean
2. @TransformerBean
3. @FilterBean
4. @RendererBean

Please refer to the javadoc documentation of these annotations/classes for details on usage.

Single JAR file

The extension must consist of a single JAR file. If you have dependencies other than the libraries provided by the DataCleaner distribution, you need to package these inside your own JAR file. If you're using Maven for your build, the Maven Assembly Plugin can provide this functionality easily using this snippet in your POM:

```
<build>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-assembly-plugin</artifactId>
   <version>2.2.1</version>
   <configuration>
    <descriptorRefs>
     <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
   </configuration>
  </plugin>
 </plugins>
</build>
```

Component icons

If you wish to add a custom icon for your components (eg. a transformer or analyzer), you need to place the icon as a PNG image with the same name as the fully classified class name of the component.

An example: If your component class name is "com.company.ext.MyAnalyzer", then the icon for this component should be located at "/com/company/ext/MyAnalyzer.png" in the extension JAR file.

Similarly, if you bundle your own ComponentCategory implementations (which define the menu groups in DataCleaner), you can define icons for these by adding a PNG file with a fully classified filename corresponding to the ComponentCategory class name.

Chapter 23. Embedding DataCleaner

It is possible to embed DataCleaner into other Java applications. This allows a simple way to add Data Quality Analysis (DQA) and Data Profiling functionality as an addition to the applications that you are building.

The simples way to embed DataCleaner is simply by doing what DataCleaner's main executable does - instantiate the *Bootstrap* class with default arguments:

```
BootstrapOptions bootstrapOptions = new DefaultBootstrapOptions(args);
Bootstrap bootstrap = new Bootstrap(bootstrapOptions);
bootstrap.run();
```

To customize further, add your own implementation of the *BootstrapOptions* class. The main scenario for embedding DataCleaner is to run the application in the so-called "single datastore mode". This can be archieved by implementing the BootstrapOptions and providing a non-null value for the getSingleDatastore() method.