Còmpilo

Compiere Developer Documentation

from an idea of Marco LOMBARDO

Table of Contents

1 Name and License

The name came from the word Compiere. In italian *Compilo* means *do it* and *I compile*, it depends on how you stress the word.

2 Introduction

The word *Compilo*

The name came from the word Compiere. In italian *Compilo* means *do it* and *I compile*, it depends on how you stress the word.

The initial idea

There is an online italian GNU/Linux manual called manualinux, you can find it at `http://www.manualinux.it/`, that began the same way: as an howto beginning with a few points. Adding topic after topic it is now quite usefull. Some parts of this document might be *highly condensed*, I apologise for that. You can query me or add topics yourself. Or write a better version than mine. I reserve the first 10 points for myself but the section "Coming soon" is not an Invariant Section so everyone is free to add topics here.

Conventions

You can find Tabs references by the Window name and the Tab name. For example: "Sales Order.Order" is the Tab Order in the Sales Order Window.

Contact: marco.lombardo@enneenne.com

Marco LOMBARDO, 2004-03-14, Lucca, Italy.

Version

Versions are numbered based on how many points the document contains. Version 1.0 will have 1000 points/topics. This is the **0.1.0** version.

Note

This is my first document under the GNU Free Documentation License. It's dedicated to RMS. The Master.

3 First 10 historical topics

000, Howto add a PL/SQL procedure and execute it.

1. Summary

   Let's consider you have a new or an existing Window/Tab and you want to add a button that executes the procedure myproc if pressed. Obviously there is a GUI part, adding the button and a PL/SQL part, creating the right PL/SQL procedure.

2. GUI part

   Open the Report & Process Window and create a new Process. Decide:

   - The name, description and help. Name is not important but I usually choose something that I think will not conflict with Compiere in the future. Description and Help are displayed when the user uses the mouse button so they are relatively important.
   - "User can start process" can be unchecked.
   - In the Procedure Name field write myproc.
   - Obviously "Is Report" field should be unchecked.
   - Parameters: you have a Tab for setup Parameters. The only thing to keep in mind is that what you write in the Column Name field is used for the column ParameterName of AD_PInstance_Para were java code pass parameters to PL/SQL code, see the example below.

   After this, alter the table referenced by the Tab in which you want to add the button:

   ```
   alter table mytable add(
     NewButton char(1)
   );
   ```

   Modify the Compiere AD, adding this Column to mytable (Table & Column Tab). For the newly created Column, set reference to Button and Process with the name of the Process you create above. In the Procedure Name write the name of the procedure you are going to create at point 3. Let this new Column show in one or more Tab (Window, Tab and Field window).

3. PL/SQL part.

   To explain how to write your own PL/SQL code, let's study an existing Procedure:

   ```
   CREATE OR REPLACE PROCEDURE M_Product_BOM_Check (
     PInstance_ID IN NUMBER
   )
   ```

   Every procedure you create should looks this way, with an IN parameters of type Number. As you have a row for each defined Process in the AD_Process you have a new row every time you call a Process in the AD_PInstance table (Process Instance).

   You have to communicate to the java code that the procedure is running. For this reason at the beginning of each Procedure you should do this:

   ```
   UPDATE AD_PInstance
   SET Created = SysDate,
       IsProcessing = 'Y'
   WHERE AD_PInstance_ID=PInstance_ID;
   COMMIT;
   ```

   "Well, but how to know the record's ID the user want to Process?": In the same table, AD_PInstance, there is the column Record_ID, so that is sufficent when you read the column for your PInstance_ID.

   ```
   select Record_ID into yuor_preferred_var_name
   from AD_PInstance
   WHERE AD_PInstance_ID = PInstance_ID;
   ```

   The table the user is working on is identified by the Tab from where you are calling the Procedure. In this example it is M_Product_ID. If the call is generated from the Order Tab, the Record_ID is C_Order_ID. For the Production Tab, it is M_Production_ID. And so on.

Parameters: when you setup some parameters in the GUI, they are passed to the procedure in the AD_PInstance_Para. Each Process Instance have a set of parameters. What you wrote in the Column Name field of the Parameter Tab is passed to the ParameterName column. The value passed by the user for this Process Instance is either in the P_Number, P_Date, or P_String depending on the parameter's type.

Usually a Compiere PL/SQL procedure have a cursor on AD_PInstance_Para as its very first task when the then procedure runs through its parameters and setup variables.

After this you can do your business logic.

After this, the procedure can end successfully or report an error.

- The Procedure ends OK. In this case the code communicates to java that there it ran successfully, together with the message you want Compiere to show in the status bar:

  ```
  update AD_PInstance
  set Updated = sysdate,
      IsProcessing = 'N',
      Result = 1, -- Success
      ErrorMsg = 'Everything OK!'
  where AD_PInstance_ID = PInstance_ID;
  commit;
  return;
  ```

- There was an error. The update statement is similar to the one above, but Result should be 0 (Error), and ErrorMsg should be your error message. If you want to use the Compiere standard add at the end error number and error message returned by Oracle.

  ```
  update AD_PInstance
  set Updated = sysdate,
      IsProcessing = 'N',
      Result = 0, -- Error
      ErrorMsg = 'Sorry there was an error!!!'
  where AD_PInstance_ID = PInstance_ID;
  commit;
  return;
  ```

4. Last thing: the log.

   Have you tried processing a shipment before? Or import new Columns into Compiere AD after adding new columns to a db table? At the end Compiere shows the list of Columns created in a text message box. There are other processes that do not. For example the procedure that Verify a BOM of a product do not show anything. So let us edit it and add this after the first begin statement.

   ```
   BEGIN
   INSERT INTO AD_PInstance_Log (AD_PInstance_ID, Log_ID, P_ID, P_Msg)
          VALUES (PInstance_ID, 1, 2, 'Hello from Bom check!');
   INSERT INTO AD_PInstance_Log (AD_PInstance_ID, Log_ID, P_ID,
                                 P_Msg, P_Number, P_Date)
          VALUES (PInstance_ID, 3, 4,
                  'Msg with 123 and tomorrows date.', 123, sysdate+1);
   ```

   Then try the procedure. You will notice your messages appearing.

001, Using your java code and jars. For example using Jasper Reports.

Recently André Legendre published a patch to Jasper Reports to include barcode interfacing. He and members of Compiere MFG+SCM have been kind to help me include their version of Jasper Report and setup the functioning example. Under `NNCompiereJR/lib` I included the patched version of Jasper Reports together with a modified version of barbecue (library for generating barcode) and other jars needed by JR. Everything is available at http://compiere-mfgscm.sourceforge.net

Here you have the modified version of JR in jar and source code, a modified version of iReport to design with barcode (download section). Under the “more docs” section there is an Howto about iReport.

1. Modify AD

   We want to use Jasper Reports so we are first going to design a report. Those who wants to use other classes/jars can go to the next point of this topic. What kind of Jasper Report can we test? I am choosing a Report with a parameter. Let’s design a Report that shows the Orders a Business Partner have, where our parameter is the Bussines Partner ID. First we create a view with the information we need:

   ```
   create or replace view NN_BPartnerOrder_V
   as
   select C_BPartner_ID, Name, GrandTotal, DateOrdered
   from C_Order join C_BPartner using (C_BPartner_ID);
   ```

   Then create a Jasper Report on this view with your preferred tool, I used iReport. The Report should look, more or less, as the file `001/JRBPartnerOrder.xml` . You can use this file directly if the dir. `/tmp` is accessible in your system. Barbecue needs a temporary dir. in order to create and place the jpg of the barcode. You should modify it if `/tmp` is not accessible on your system or you will see the report without the barcode. Here you can find the .jasper too, in fact you need the xml or the jasper file for the next step.

2. The java part.

   Usually you have some .java file that uses .jar, directly or indirectly (jars used by others jars). Where are we going to put the jars? We have 2 options:

   - Don’t want to unjar anything. This is the case of `oracle.jar` used by Compiere. These jars are then included in the classpath everywhere, when compiling and when running Server or Client or WebStart application. If you want to add a jar of this type you have to modify some build.xml, some .jnlp, some .sh
   - The second possibility is to include the classes you want to use in a jar already present in the build/run script classpaths. If you look at the creation of `CTools.jar` the `build.xml` logic compile some code and unjar some jars in the same build directory then jar everything together.

     The build/run scripts need to include only `CTools.jar` to use new classes created compiling java code and classes simply expanded in the build directory before packing `CTools.jar`

     We are going to use the ladder method: our compiled classes will be packed together with the expanded version of jasperreports-0.5.2.jar and every jar used by Jasper Reports or by java code. This will result in a NNCompiereJR.jar and build/run scripts will be modified to let Compiere use this new jar. You can find this work in the NNCompiereJR module on the CVS.

     The final step of this point is to generate the NNCompiereJR.jar: download the module go to the NNCompiereJR dir. and type ant (ant 1.6.1 is required). At the end the jar will be ready.

     Note: if you want to upgrade/downgrade Jasper Reports or any other jar just update the lib dir. and the new jar present here will be used during jar creation and packed together in the final jar.

3. Modify build/run scripts.

   - `utils_dev/build.xml`

     I added my dir here so NNJasperReport get compiled when I compile Compiere. I add this line: `<ant inheritAll="false" dir="NNCompiereJR">` and similar information in the clean section.

   - `client/build.xml`

     Here we have to add the unjar of one jar more before to create CClient.jar . The line I added is: `<unjar src="..\NNCompiereJR\dist\NNCompiereJR.jar" dest="${build.dir}" />`

   - `utils/compiereDirectTemplate.jnlp`
     The precedent modifications are important at compile time. This and the one that follow is important at run time. Add the following line to the file indicated in the <resources> section: `<property name="ru.compiere.report.path" value="/tmp"/>` /tmp is a directory on the client, you can use a network path and is were Compiere will look for .xml and .jasper files
   - `serverRoot/src/web/compiere.jnlp`
     Same modification as above.
4. Modify the Application Dictionary. As you can imagine, if you are not new to JasperReport our Reports will always be generated by the same class, that will read a different .xml or .jasper from Report to Report. So we need to add a Column/Field in the Process tab. Add a column:
   ```
   SQL> alter table AD_Process add (JasperReport VARCHAR2(200));
   ```
   load the column inside Compiere:
   ```
   SQL>  db/maintain/Maintenance/0_Add_New_Column.sql
   ```
   create the new field:
   ```
   SQL> get db/maintain/Maintenance/0_Add_New_Field.sql
   SQL> /
   ```
5. Specify a new Process.
   As you can see when you go to specify a new Process there is a new Field. Create a Process with:
   ```
   classname: ru.compiere.report.RusReportStarter
   JasperReport: JRBPartnerOrder.xml
   ```
   with a Parameter with:
   ```
   ColumnName: C_BPartner_ID
   Type: TableDir
   ```
   This class internally looks first at the `ru.compiere.report.path` property for the path. Then it uses `$COMPIERE_HOME/russian/reports` when searching for xml . If you are lazy and wants it fast, just put reports under `$COMPIERE_HOME/russian/reports`. Now you can try the Process: when you run it the `ru.compiere.report.RusReportStarter` class will be executed. This class will read from AD_Process to know which Jasper Report file to use, and look for it in the path inside `ru.compiere.report.path` (`/tmp` in our case).
   This will read the C_BPartner_ID from AD_PInstance_Para and shows you the Sales Orders this BPartner has.

002, Compiere Application Dictionary.

(To be continued...)

```
AD_ Application dictionary
A_ Asset Management
C_ Core Functionality
GL_ General Leder
I_ Import
K_ Knowledge Base
M_ Material Management
PA_ Performance Analysis
R_ Request
RV_ Report Viewer
S_ Service Management
T_ Temporary
W_ Webstore
X_ Generated Model
```

003, Using Attribute Set.

Summary

1. You can create Attribute, this can be:
   - instance Attribute or not.
   - list Attribute or not.
2. Then you should create Attribute Set this can be Instance Attribute Set or not.
3. Go on and assign the Attribute Set created to your Product.
4. What you can do it is:
   - in the Product window you can specify the non-instance Attributes with the Attribute Set Instance field.
   - in windows like Sales Order or Shipment you can specify the Instance Attributes.
   - in the warehouse Products show their Attribute Set Instance if present.

Note: You can not set Attribute in the Order Lines Tab. If you want to do it you have to change the code. See point 004.

An example

1. Attributes

   Take a look at the Attribute window. In the GardenWorld demo there are Attributes. "Color" is not an instance attribute: this means that you can specify it in the Product catalog (the Product window) and it's fixed so you cannot specify it when compiling an Sales Order. It is a list Attribute this means that when you go to choose it Compiere pop-up a list of values.

   Description is an Instance Attribute so you can specify it for every Product movement and you cannot specify it in the Product window. It isn't e list Attribute so you should not specify a list of values, as for the Color Attribute, and Compiere pop-up a text field when requesting to you to input a value for this Attribute.
2. Attribute Sets

   Look at the Attribute Set window. There are same examples in the GardenWorld demo. Go on and create a new one, let's say the ColorWDescription Attribute Set. For it check Instance Attribute. Now under the Attribute Use Tab, specify our two Attributes Color and Description.
3. Specify the Product Attribute Set.

   Open the Product window and take a demo Product. Let's say "Oak Tree". And for it specify Attribute Set as "ColorWDescription". Save.

   In the same window press the icon on the Attribute Set Instance field. Compiere will let you choose the color but not the description. Choose your favorite color and save.
4. Material Receipt.

   Open the Material Receipt window. Create a receipt with one line with a Oak Tree Red (i like red so I choose it, the Attribute selector is in the upper right corner of the Search Product window). In the receipt line you can choose Description but not the Color. Process the Material Receipt.

   Open the Warehouse. You can see an Oak Tree with your description. And the Color Attribute?? It does not show, infact the similar example you can find, that with T-Shirt, the color is putted together with the Product Name. But you can search the warehouse for "red product" because the Attribute selector let you filter Products by their Attribute Values.

More examples

- You sell T-Shirt with different size and color. The price is the same. In this case is better if you define this 2 as Instance Attributes. You create just one T-shirt Product with its price. Then choose Attributes when filling a Sales Order.
- You sell T-Shirt with different size and color. The price vary with respect to the T-shirt size. You should create a Product for each size. The Attribute Size should be non-instance

attribute so you specify it in the Product window, e.g. for the “T-shirt M” product you choose M for the Size Attribute.

And choose the color in the Sales Order Line. You have more Products in this case but when searching you can still search for a Size or for a Color.

Same decision if you have Product that differ in something. BOM or price or cost.. depending on Size or Material, etc, that force you to create more Products. non-Instance Attributes can still be used to group them.

004, Using Attribute Set from developers point of view.

I will refer some problems in Compiere 2.5.1b and explain some classes, resolving these problems.

1. First of all the Attribute Set Field in the Order Line Tab is read only. So open the System Client go to the Window,Tab&Field Window and select the Sales_Order.Order_Line Tab. The field Attribute Set Instance should not be read only.
2. Now if you go to the Order Line Tab and try to specify Attribute for a product that had been setted up correctly you receive a

   > No Product attribute information.

   message.

   This is due to the `client/Src/com/compiere/grid/ed/VPAttribute.java` code. If you look at setField method you will find which Fields are enabled for Attribute Set. The Fields of the Sales Order Line is missing. Because the AD_Column_ID of the Attribute Set Instance Field of the Order Lines Tab is 8767 modify this:

   ```
   m_instanceWindow = AD_Column_ID == 8772 || AD_Column_ID == 8552;
   ```

   with:

   ```
   m_instanceWindow = AD_Column_ID == 8772 || AD_Column_ID == 8552
                    || AD_Column_ID == 8767;
   ```

   Then recompile Compiere. Now if the Attribute Set setup is ok you would be able to insert Order Lines with Products and Attributes.
3. The other problem is the New Record button in release 2.5.1b. In the old 2.5.0d was not possible to choose: always Compiere creates new record so InOut Lines were not summed in the Warehouse.

   The same happens if user creates a new record for Red-Large when there is already a Product with that Attribute Set in the Warehouse. You have to change the `VPAttributeDialog.java` class and probably `MAttributeSetInstance.java` class model.

   One solution could be add a method to the model to find Attribute Set Instance with same Description of itself. Then modify the dialog in the saveSelection method to check out for similar Attribute Set Instance before to return newly created record ID.

005, Trees

How a trees are organized? What kind of trees we have? Tables that control some kinds of trees are:

```
AD_TREE
AD_TREEBAR
AD_TREENODE
AD_TREENODEBP
AD_TREENODEMM
AD_TREENODEPR
```

`AD_TREE` contains the trees you are mantaining:

```
Menu
Primary Account Element Value
Primary Product
Primary Business Partner
Primary Organization
```

```
Primary Project
Primary Sales Region
GardenWorld ElementValue (Account, etc.)
GardenWorld Product
GardenWorld Bus Partner
GardenWorld Organization
GardenWorld Project
GardenWorld Sales Region
GardenWorld Campaign
GardenWorld Activity
```

The column `TreeType` is a reference:

```
SQL> select name from ad_ref_list where ad_reference_id = 120;
NAME
------------------------------------------------------------
Activity
BoM
BPartner
Element Value
Campaign
Menu
Organization
Product Category
Project
Product
Sales Region
```

Trees can be edited from the "Tree Maintenance" window. Menus in the left menubar are in `AD_TREEBAR`, in fact `AD_tree_ID` in `AD_TREEBAR` is always 10, the ID of the "Menu" tree. Look at the Menu point to learn about Menu Bar.

For a product let specify a product as a "Summary Level". Then go to the "Tree maintenance" Window: here you can see a folder for that product (I used Azalea Bush). Move some products under that folder. (I put 2 product under Azalea Bush). Below you can see the result:

```
select level, substr('          ', 1, level*3)
              ||name as name, node_id, parent_id
from ad_treenodepr t join M_product m on (m.m_product_id = t.node_id)
start with parent_id = 0
connect by prior node_id = parent_id
/
     LEVEL NAME                                      NODE_ID  PARENT_ID
---------- ------------------------------------- ---------- ----------
         1    Planting Service                          126          0
         1    Rose Bush                                 127          0
         1    Azalea Bush                               128          0
         2       Azalea Bush1                       1000000        128
         2       Azalea Bush2                       1000001        128
         1    Holly Bush                                129          0
         1    TShirt Blu                            1000008          0
```

Same for the BPartner, but join then `C_BPartner` table with `AD_TreeNodeBP`. Other types of tree are reported in `AD_TreeNode`. For example if you make a Summary Sales Region "qqq" and under it the "aaa" Sales Region, this will show this way inside the db:

```
select level,
       substr('          ', 1, level*3)||name as name, node_id, parent_id
from ad_treenode t join c_salesregion m on (m.c_salesregion_id = t.node_id)
start with parent_id = 0
connect by prior node_id = parent_id
/
     LEVEL NAME                                        NODE_ID  PARENT_ID
```

```
---------- ---------------------------------------- ---------- ----------
         1    West                                            102          0
         1    qqq                                         1000000          0
         2       aaa                                      1000001    1000000
         1    East                                            101          0
```

Seems that the same will be implemented for Product Category.

006, Menus.

Menus are defined in the `AD_Menu` table, better if you look at this from the System Client. To create a new menu you have to add a row here (Of course you can do it with the GUI but that's quite boring for me). You can define folders with the `isSummary` column. `Action` is a reference:

```
SQL> column name format a30
SQL> column value format a4
SQL> select name, value from ad_ref_list where ad_reference_id = 104;
 NAME                           VALU
------------------------------ ----
Workbench                      B
WorkFlow                       F
Process                        P
Report                         R
Task                           T
Window                         W
Form                           X
```

Depending on it you specify a `AD_Window_ID`, or a `AD_Process_ID`, or a `AD_Task_ID`, etc. This gives you a menu list. To organize a menu tree you have to use `AD_TreeNodeMM`, in fact `AD_Tree_ID` is always 10 in this table. Trees start with `PARENT_ID` = 0. Let shows it:

```
SQL> column name format a40
select level,
       seqno, substr('          ', 1, level*3)
              ||name as name, node_id, parent_id
from ad_treenodemm t join ad_menu m on (m.ad_menu_id = t.node_id)
start with parent_id = 0
connect by prior node_id = parent_id
--order by seqno
/
     LEVEL NAME                                        NODE_ID  PARENT_ID
---------- ---------------------------------------- ---------- ----------
         1    Quote-to-Invoice                               166          0
         2       Sales and Marketing                         272        166
         3          Commission                               257        272
         3          Sales Setup                              269        272
         3          Mail Template                            238        272
         3          Sales Region                             137        272
         3          Marketing Channel                        136        272
         3          Marketing Campaign                       134        272
         3          Commission Run                           264        272
         2       Invoice Inquiry                             100        166
         3          Invoice Transactions (Doc)               252        100
         3          Invoice Detail                           253        100
         3          Invoice Transactions (Acct)              202        100
```

As you can see the `SeqNO` column gives the order to show them in the GUI, not easy to show it with a select using the `connect by` because it is referred to the tree level.

Menu Bar

The definition of the menu bar is inside `AD_TreeBar`.

```
column name format a30
SQL> r
  1  select AD_USER_ID,  NODE_ID , t.AD_CLIENT_ID, t.ad_org_id, m.name
  2  from ad_treebar t join ad_menu m on (t.node_id = m.ad_menu_id)
  3* where t.ad_client_id = 11 and ad_user_id = 100

AD_USER_ID    NODE_ID AD_CLIENT_ID  AD_ORG_ID NAME
---------- ---------- ------------ ---------- ------------------------------
       100        110           11         11 Business Partner
       100        126           11         11 Product
       100        129           11         11 Sales Order
       100        383           11         11 Cache Reset
       100        479           11         11 Product Catalog WH
```

You could get different result, this is my case where I have this 5 menus on the left menubar.

007, Reporting Engine.

There are 2 ways to start a report/process:

1. Create a menu that points to a report. In this case `VTreePanel` fire the `firePropertyChange` that comunicate with a particular `AMenu`. Than a `AMenuStartItem` is initialized, and started (AMenu:356).

   `AMenuStartItem.run()`: you can reach this with Menu and Workflow. In the first case the code looks into `AD_Menu` in the second into `AD_WF_Node`. Based on `Action` you jump to `startWindow`, `startProcess`, `Workbench`, `Workflow`, `Task` (OS command), `Form`.

   `AMenuStartItem.startProcess()` create a ProcessDialog that show description and if user choose “OK” calls `ProcessCtl.process()`.
2. Push a button associated with a report. The event reach `APanel.actionPerformed` and from here `actionButton`. Here you have special meaning for columns **PaymentRule**, **DocAction**, **CreateFrom**, **Posted**. If none of the above apply then the static method `ProcessCtl.process()` is fired after asking to the user.

`ProcessCtl.process()`

- ProcessParameter is used to check and eventually input parameters. `Than a ProcessCtl` instance is created and `start()` called.
- `run()` is reached after `start()`, here a big SQL get all infos about the Process/Report. Then:
  1. If you write a `ClassName` this is called.
  2. If is a report you **should** specify a Report View.
  3. Report: the specified procedure is called if present, as a pre-report task. Then `ReportCtl.start()` is called.
  4. Process: the specified procedure is called if present.

`ReportCtl.start()`

This is module client, `org.compiere.print` package. At the beginning you can see come hard-coded reports with ID: 110, 116, 117 and more. Else `startStandardReport()` is called. Here a big SQL statement get info about Process and Table and Report View. The joins between tables are:

```
Process Instance -- Process -- Report View -- Table
                                      |
                                   (outer)
                                 Print Format
```

Before launching the report:

1. An `MQuery` is created based on Report Parameters, dinamic part, and where clause of the Report View, static part.

2. A Print Format is created or copied from System client. This do not interest here because right now we do not want deal with autogenerated Print Format. Keep in mind that the Print Format used is the one that have the same Report View of our Report.

Then a ReportEngine is initialized with (`ReportCtl.java`:155) Print Format and Query. Finally the call `new Viewer(re)` shows the report.

`ReportEngine`

This is in the print module, under `org.compiere.print` package. In the constructor the Print Format is simply copied in a private field.

A `setQuery()` call load data calling `setPrintData()`. Here is used DataEngine to load data to be rendered with Viewer and LayoutEngine in the next step.

DateEngine.getPrintData: here you have 2 cases:

1. You have a Print Format with Report View, in this case the Table Name is taken from Report View.
2. You don't have a Report View. In this case Table Name is taken from Print Format.

Then a call to `getPrintDataInfo()` and `loadPrintData()` that get/load what you want to print e.g. Columns, "sum" you want, "count" and so on. Now the history: this first piece of work was done due to a **problem**:

> I had a Report View with a Column named "NN_ProductionQty" and I would like to get sum of the report lines of this column. There was no way to get "sum" printed on the report. Everythingelse works fine.

The engine get confusing because of the implementation of `PrintDataGroup`: in the `addFunction()` of the class used in `getPrintDataInfo()`, sum of Column `Qty` is added as string `Qty_S`. This when `loadPrintData()` uses `PrintDataGroup.getFunctions()` to retrive functions on the report it gets confused in the case of "NN_ProductionQty_S" because it uses "_" to separate Column Name and Function Name.

HINT: do not use underscore character for Column Name either for Tables and Report View, or it will be impossible to get Sum, or Count or whatever function calculated for that Column.

`Viewer`

To be continued...

008

For release 0.0.8

009

For release 0.0.9

4 Contributions

This section is not an Invariant Section. So additional works can go here. Finally this section came: first contributor joins.

010, Application Dictionary (by Peter Shen)

You can find the original `.doc` document under the 010 directory. I did not put there a pdf version a could generate with Open Office, but you can use it to view the file. Here there is a text formatted version without images.

```
Windows  <-- Tabs    <--  Fields
              ^             ^
              |             |
             Tables  <--  Columns
```

How is a window created? A Window normally contains several Tabs, and a Tab contains several Fields. A Tab is based on a "Table", so a Field in a tab is just a "Column" in a "Table".

So the process to generate a window is to:

- Create a table into Oracle (Or use an existing Table)
- Create a table in Application Dictionary (Or use an existing Table)
- Create Columns in Application Dictionary for this Table
- Create A window in Application Dictionary
- Create Tabs for this window using a table in Application Dictionary
- Create fields for the Tab
- Create a new Menu item and add a window menu into the menu tree

Login again, **IS THAT WHAT YOU WANT?**

What is a Table in the Application Dictionary? A table in Application Dictionary can be physical table or a view in oracle.

The "Tables and Columns" Window.

Open the "Tables and Columns" Window, the "Table" Tab appears.

- DB Table Name, The name of physical table or view in Oracle.
- View, Click if this is a view in Oracle.
- Data Access Level, System Only means its data can only be accessed by System Client, Client/Organization means the data can be accessed by Client(*) or organization (GardenWorld, HQ), Organization: the data can only be accessed by organization (GardenWorld, HQ), System/Client: the data can be accessed by System Client and Client (GardenWorld, *)
- Window, Means which window uses this table.
- Record deleteable, Means the user can delete records in the table.
- High volume, This table may contains huge records, so display the search dialog first.
- Create Columns from DB, Read the table's fields from oracle and generate relative columns.

Click on the "Column" Tab.

- DB Column Name, The column name in the real table or view in Oracle.
- System Element, This is used in displaying like report.
- Reference, The column's Data Type, also it decide what the field will look like in the window
- Validation, see Validation Rule (missing)
- Reference Key, see Reference (missing)
- Default Logic, Set a default value for this column, you can use constant value like 1, or the value in Context like @AD_Client_ID@, or just or sql query like @SQL=select?..
- Key Column, Something like primary key.
- Parent link Column, Used in Master/Detail tabs, indicated with which column the detail tab link the Master Tab.
- Mandatory, The column cannot be null, otherwise, the system will give an error message.
- Updateable, The value of column cannot be updated, u save the value and then the field become readonly.
- Read Only Logic, In which case, the column is readonly, it should be Boolean value like 1=2
- Callout, Callout is a small piece of java cold, called when the user initialize the window or some value is edited, for example: there're 3 fields in window, a price, a quantity, and an amount, whenever the user change the value of price, or Quantity, the amount should be changed automatically. Then you add a callout like that:

```
private static String ProductCode (Properties ctx, int WindowNo,
MTab mTab, MField mField, Object value, Object oldValue)
{
Integer M_Product_ID = (Integer)value;
if(M_Product_ID != null)
mTab.setValue("M_Product_ID2" , M_Product_ID);
    return "";
}//ProductCode
```

- Selection Column, Means this column is a search key, when u push the search. **Attention**: The 3 column will become search key automatically, Value, Name, Description

You have almost finished with Table, but before that, you must check these:

1. A table must contain such columns: AD_Client_ID, AD_Org_ID, IsActive, Created, CreatedBy, Updated, UpdatedBy
2. All the column name is case Sensitive

The "Windows, Tabs and Fields" Window.

The Window is the real thing you should define to show. In the "Window" Tab:

- Window Type **Maintain** means in this window you can create record, delete record, update record.

  Query Only means in this window u can only view the data, no creating, no deleting.

  Transaction: transaction is generally used in some business transactions like sales order or shipments, often it has a column called Processed, when processed='Y' then this record cannot be edited And the transaction window only show the data which processed='N' and created today, and after you push the history button in tool bar , then the other data can be displayed.

in the "Tab" Tab:

- Table, Which table is the tab used, (a tab must depend on a table defined in table).
- Sequence, The display sequence, from up to bottom.
- Tab level, Means the relationship in master/detail tabs.
- Single row layout, The data will displayed as single or multi rows by default.
- Has Tree, The tab will showed with a tree, like that you can see in the "Menu" Window. **Attention**: It does not mean a tree will displayed automatically if you click this, only some table has tree (M_Product, AD_Menu, C_BPartner, and more), which is hardcoded
- Order Tab, In this Window in the Tab "Field Sequence" is an example of Order Tab.
- Process, You can define a process for the tab, it is normally used for report.

in the "Field" Tab:

- Column, The table's column, a field must depend on a column.
- Field Group, In the "Sales Order.Order" Tab you can see 2 Field Group: Reference and Status.
- Display Logic, In which case the field is displayed, you can use constant value like 1, or the value in Context like `@`AD_Client_ID`@`, or just or sql query like `@`SQL=select?..
- Heading Only, Show the field's name only.
- Field Only : show the field's value only.

Last thing to do open the "Menu" Window and add a new Menu. Select Action=Window and in the "Window" Field select the Window you have created.

Login again, to see what happened!

011, Compiere Source (by Redhuan D. Oon)

Overview of Topics

Topic 011 treats the subject of handling codes in Compiere into various chapters. We touch first on the background of Compiere and the forces that leads to this document, for a reconnaissance plane's view. The level of modifications referred to here ranged from trivial but survival, to minor surgery to tweak and optimise fully of what Compiere can be without increasing the burden of maintaining it. We are covering mostly simple functional issues and business needs and not cosmetic or computing issues. We pick up some idea on what kind of codes we are facing. We will also learn to setup Eclipse IDE sufficiently to debug Compiere. Then we see how to debug the codes, and how best to approach it. The dangers and best practice are explored. Final work is then compiled and deployed.

Open Questions

Where did it come from?

Compiere Source looks very huge and very well designed. It must have a history before the OSS way. It was complete even before it got ported to SourceForge. Compiere provided this link for us to know more of its yesteryears: `http://www.accorto.com`. Having said that, it is so timely and groundbreaking to have such a powerful business application put in the open. It is unprecedented.

Is it truly Open?

There are still classes within its source (`maintain.jar`), that do not have java source attached. The classes there for Migration and Replication worked via a paid service provided by Compiere Inc. It also has to debunk itself from Oracle as the database. Jorg Janke, the creator of Compiere has put this on the agenda. It is also debunking itself from JBoss, to achieve Application Server independence.

What about been Open Knowledge?

To learn Compiere can be a mixed experience. The foundation User Manual, gives terse albeit biblical description to all its features. That cost USD40, sold on the Web Store of `www.compiere.org`. The on-site literature provided is however sufficient and well given to get Compiere up and going, especially so on Windows platform. If you investigate further from their website you can also get some reference as to Compiere's architecture and development pointers. But now there is a growing need for more context information to the source codes, and for a more wider audience, accelerated by the Open Source market. SourceForge forums has become very cluttered and the search engine is archaic. The Development Sub-Forum may be overwhelmed in demystifying this expensive architecture. Compilo and RED1.ORG attempts to give it sanity as they are controlled by individuals.

What about Market Forces?

Since Compiere is a business application suite, it therefore draws business concerns and commercialisation forces more than anything else. Business can be selfish and territorial, oxymoronic to open borderless sharing. It is debatable which is more better for business - closed and governed and sanitised, or open and anarchic and impassioned. There are fears that others will devise means to ride on the Open Source wave while evolving closed commercial spin-offs from it and erode the base application's growth. Again debatable as laissez faire. For now, its long journey can be complete if it becomes number one and stays there - the final importance. That means strong leadership. To prepare for chaos.

The Source

Compiere is written mostly in Java and are located in sub folders under a main folder called Compiere-all. It also has embedded and independent SQL queries. Its architecture follows closely J2EE's N-tier model. There are java client, generated jsps in the front; Java beans, servlets, XMLs and java factory components in the middle; JBoss Tomcat application server and Oracle Database in the back. The apps server service the accounting engine and Web interface.

The source is considered well-annotated and well written. It is fully integrated in its design from the ground up and thus poses a huge challenge to even thinking of changing it. The source is also getting heavier with each version release, unzipped at around 200 Mbytes for version 251e. When compiled, the zipped binaries finished at 27 Mbytes.

The errors or bugs that may be are usually typo and omission mistakes that are easily corrected. The source codes controls most of Compiere's core application processes. After this source, there is also another tool for changing application functionality using metadata, called the Application Dictionary (AD). This subject is dealt elsewhere as we are only mostly concerned about the raw source here. But suffice to say the AD is the most important developer tool given by Compiere.

Compiere source versions changes steadily and is expected each month to come up with new features and capabilities. If you do not bother with the new releases and carry on with your present version, there is no issue. But whenever you wish to upgrade your codes to the latest release, you have to do your changes all over again.

Thus documentation is important to remember what code changes you have made. Study the future release information before deciding on the freeze version. Weigh between waiting for the version to arrive or make your changes now, and repeat later when taking on the new version. Upgrading codes also go hand in hand with migrating your database to be in sync. Migration is a paid service provided by `www.compiere.org`.

The Source Layout

The image on the right is taken from the Eclipse IDE, to show the packaged folders. The BASE folder is expanded to show its various packages.

Often debugging is in `org.compiere.model` and `org.compiere.process` of this folder.

The Model package also contains the Callout source codes, which should be the focus of any field level business logic.

The Process package houses most of the push-button process logic.

This folder is compiled into the `CClient.jar` with other folders and any changes to them should only affect that jar.

The accounts process and posting actions are handled by the `org.compiere.acct` package of the Server folder. Any changes there are compiled into the `CServer.jar`. This folder cannot be debugged in Eclipse as its execution is passed into JBoss that runs outside Eclipse during debugging.

Some insight into other code folders are touched on by Marco in earlier chapters such as the one on Jasper Integration. Marco also touched on Ant Build.

You shouldn't change any logic of other codes that are highly integrated and abstract. Most functional changes are capable via the Application Dictionary (AD) and Callouts or Procedures. Compiere's claim of 98% programming not-needed rule is believable, as we are not concerned about integration work here but normal debugging and business rules changes.

Cracking the source

Understanding Compiere's architecture, processes and activity flow from scratch is tricky and offers a steep learning curve. Learning and practicing in modifying the codes can give the academic experience but the most effective way to do it is on the job, with real clients - hands-on, building your "flying hours".

Its best and safest to start with an off-the-shelf Client that requires no modifications and minimal demand on performance. Take the Boat-Plane. Forget about the Jumbo Jet for the moment. Work for free. Be paid in experience.

Business Functionality

It can take a long while to understand how Business Functionality is weaved inside Compiere. Usually after some practice all this becomes clearer. Try from this angle. Any business module is first defined by its table-column structure in the AD. There, you also define certain validation rules. Then you develop detailed business logic in the Callouts and push-button Processes. SQL Procedures may be created if the logic is strictly document based.

You may look at a Callout as an extension at the field level. SQL Procedure is then an extension at the process level.

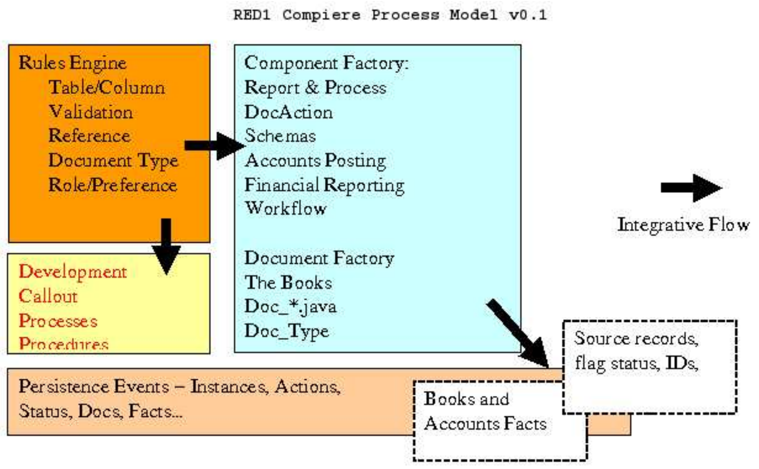
By push-button, we mean the process buttons that appear in many windows that is pressed to begin a process on the record. These processes plays around with `Doc_Action` and `Doc_Status` values in the record.

Much of the document handling logic is controlled by values in the Document Type such as GL Type and Base Type.

Much of the User Access and personalised behaviour is controlled by Roles and Preferences. The later can also be done on the fly by the user in any screen.

Recently with the advent of WFMC (Work-Flow Management Coalition), Compiere has incorporated a workflow engine to handle much of the push-button processing. It is still too new and abstract and shall be treated in another article.

The processed document's accounting consequence is handled by the `Doc_*.java` classes to effect the General Ledger and the accounting books. Let's try to see this in a simple model:



Thus a typical new module may go through the following development tasks.

Development Work Cycle

The RED1 Compiere Development Model v0.1

- RULES ENGINE Setup in the Application Dictionary (AD)
  - Table-Column structure
  - Input handling and validation rules
  - Document Type & Print Format
  - Define Default Accounts
  - Workflow process
  - Callout definition and process-button definition consolidated via Report & Process
  - Final Consolidation in Windows under the Menu tree
- DEVELOPMENT Work

  Place the codes in respective containers

  Callouts in `org.compiere.model` package of `base`

  Process Actions in `org.compiere.process` of `base`

  SQLs in `db/database/procedures` of folder tree

Tied to definition in AD

Reuse the FACTORY aspects of Compiere to minimise impact.

- Review integrative logic between RULES ENGINE and FACTORIES

The ending point on reuse may have minimal design impact if we correctly weave our module into the present integrative adaptors. For example, if we want to explore with a Payroll module, the final figures accounts posting can be done via the default accounts-id schema in the Business Partner Category and inducted into the present Accounts Payable model (Invoice of expenses, in this case - wages). Other regulatory figures or books has to be reviewed during accounts posting, whether coding flows are correct.

All said, that reuse point may be the 90% headache! If you do not integrate and link your own suite then your codes is out of the architecture and much features of the architecture such as security, access control, user preferences, window design and report generation may be lost from your separated codes.

For experts, you can follow the pattern of Compiere codes to create different functional modules.

Compiling from Source

When we modify the Java codes or SQL queries, we must also compile and rebuild the code packages before such changes can take effect. Such a process is carried out by `RUN_build.bat` in the `Compiere-all\utils-dev\` folder, which refers to the parameters in `build.xml`. These routines require Apache Ant and Java SDK to work.

Build Process

You can check some settings in `myDevEnv.bat` of `Compiere-all\Utils-dev\` and proceed to build your source codes for redeployment. If that batch file does not exist it has to be copied from `myDevEnvTemplate.bat`.

The settings concern the paths of your Compiere-all source directory, target directory of your binary where Compiere2 will or is residing. Then there is the install repository that you want the zip binaries to be stored in. This repository is like a backup where you can let other users access it to pick up the changed binaries. It is not important to put it in the same place as Compiere2.

Next file that must be there is your `build.xml` file that specifies all the work that is going to happen in the build process. Usually there is nothing to touch in there so you can leave it alone.

After doing your backups, you can execute `RUN_Build.bat` from the utils-dev directory. Note the log messages from the the dos prompt. If the build failed, you have to check what is the cause or causes. Initially when you just started, it is always the path problems.

Even if the build process does not complete and exit on error, you can check `Compiere-all\install\build\` to see if your binaries are there and date-stamped correctly (the time you run the build). If so, then you can use that and copy to Compiere2. Otherwise it's a java source error that needs code correction.

SQL Debugging

There are also significant source in the form of independent SQL procedures, views, triggers and functions that exist at the database level. With the effort to debunk from Oracle, the triggers have been removed from the database domain into the source code domain, as of version 251d. Where are the triggers now? Fortunately ELDIR of The Netherlands answered it in RED1/forum:

> The trigger code is now part of the model classes. The base class for all of them `PO`, provides a method called `beforeSave()`. This method is called when `save()` is called on any derivate of `PO`. Final model classes can implement this method to perform any "trigger" operations. As a consequence, all updates should be done by means of the Compiere persistence engine, as there are no "real" database triggers.

SQL routines that still exist, such as procedures are generated by source i.e. `db\database\Procedures` which create themselves as stored procedures in the database. You can use any SQL editor to debug them. Oracle's EMC has one such tool. A popular third party tool is TOAD. For some review of using Toad see `http://compiere.red1.org/Toad.zip`. Such SQLs are saved and carried in the `ExpDat.dmp` during `DB_Export` as an Oracle DB backup.

Researching from the Web

As there is much of java-J2EE and Oracle, and other common technologies that goes with it, the Web is the richest source of background development and technical information in demystifying Compiere. Most developers agree that the best style to learn is looking at similar samples, comparing them. Compiere seems the biggest single coherent and integrated mass of source examples!

To look in the web for Compiere's equivalent is almost non-existent. Thus to compare enterprise codes and concepts to shorten the learning curve has to be done en-bloc taking Compiere as a whole, or looking at parts from scratch. For example if one tries to learn via the J2EE blueprint or Open For Business (OFBiz) samples will find them to be quite apart in design and direction. Its best to go through `www.compiere.org` once over, go through some exercises here, pick a good book on J2EE and look for patterns to follow, and return again to step one - iterating and evolving your comprehension. For oracle troubleshooting try `www.searchoracle.com` as it is most authoritative and well organised.

Using the Forums

The SourceForge forums which houses Compiere is most important in the pursuit of Open Source know-how. Software presented in such fashion is without warranty and may lack a principal vendor's support. However Compiere Inc, USA, the founder has created a support basis which gives priority to paying customers. Still most of the forum community aren't paying customers so there exist a critical mass of users to rely upon.

Be patient and observe the usual netiquette in pursuing your interests. Before even asking a question in there, you should use the search engine in `www.sourceforge.net` to look for clues first. Half of your problems are probably answered by someone before. Its just buried somewhere. Digging through the forums has a side-benefit of giving ground to you. The knowledge base that you pick up is important to build your confidence, and always wanting spoon-feeds may hurt you in the long run.

Also try to contribute back, once you reach any level of competency. Another good tip is that when others see more of your nick in there, especially contributing ideas and answers and pleasant, then someone will feel motivated to give you attention. Remember that all of the others were once upon a time like you - dummies. You can be more open and provide some background info in your user profile or upkeep the diary provided. Honesty is the best policy.

Bugs are for bugs and follow the instructions on what to do. Do not plaster yourself everywhere. Even in forums, there are proper sub-forum channels for you to participate. The Open Discussion sub-forum, like its message says, is not meant for support or bug reports, and yet many trample that short note.

The Developer channel is more suitable for us, from novice to wizards - sharing development and source code findings and issues. The Database Independence channel is for those who wants to discuss on how to tackle its namesake. The ERP channels separates out the topics based on functionalities. If you have a specific problem and you know its category, its best to ask it in the right channel, or else you may get lost and the place look messy. If unsure, look for someone busy, then ask directly. If your English is bad, you are certainly not alone. If in doubt, just use some smilies :) ;) :o). The web community is self-governing and so there need not be much written rules.

How to Setup Eclipse for Compiere

Compiere was previously developed using JBuilder as the Integrated Development Environment (IDE). Now its developer team is using Eclipse, also an Open Source project. We know its

Eclipse from the tell-tale files in Compiere Source - the presence of `.classpath` and `.project` files. Having an IDE greatly helps in tracking code execution and code modifications.

Downloading Eclipse

You can download the latest version or just version 3.1M from `www.eclipse.org`. From the tar or zip file, extract them into a root directly i.e. C: or D:. The sub-folders will extract itself, i.e. `C:\eclipse\plugins\..` . For more visual screenshots guidance, please refer to `http://compiere.red1.org/Callout.zip`.

Setting up perspectives and views

The perspectives we often use are Java and Debug. Select those from the top menu bar. Also select various views if they do not automatically appear later. Initial important view is 'Problems' as when we import Compiere afterwards, problems are bound to happen and we need this view to show us what they are.

Importing Compiere Source into Eclipse

First you download Compiere source from SourceForge. Choose version such as Compiere-Source251d or via CVS. The CVS version may have to be older for stability from bugs.

Right-click on the left - the large empty panel to pop up for the import task. Once selected you specify your Compiere Source location. Then go one more level down to the `compiere-all` folder. Before that, copy the `.classpath` and `.project` files into the `compiere-all` directory. For samples of those two files use from `http://compiere.red1.org/eclipsefiles.zip`. You have to change some path settings in there to reflect your Compiere Source location and other utilities such as Java SDK. The project name is also set in `.project`. You can also change it from the Eclipse properties menu.

When the import is complete, stay in Java Perspective mode to see if there are errors, which usually will be for a fresh import. Errors showed up as red X icons. Bring up or look at the Problems View to see what it is. You may double click on the problem line to be directed to the problem area. If there is a message about duplicate `CompiereCtrlMBean` just delete that file from the source tree. Then right click the root folder to select 'refresh'. Wait while the progress bar finishes (will prompt on right side at the bottom). If errors still persist, see what they are in the problems view. You may even need to close, exit and reopen the project. When deleting the project, you need not select the option to delete source unless you know what you are doing.

Setting up Debug and RUN modes

When source is error-free then you may proceed to test it out. Define your DEBUG and RUN settings from the top menu bar. Select your main class to be Compiere and click on Debug. But before this you should take a breather and spend more time browsing through the source tree and familiarize yourself with the codes structure and arrangement. Notice that the codes are bundled in packages. Take note also that somehow certain codes for Web Store and CServer.jar cannot be debug in Eclipse. They are executed as proxy through the JBoss Application Service. JBoss is integrated into Compiere to give it the enterprise heavy duty servicing of web clients and multiple client access to the Compiere server. CServer codes are also inclusive of the accounts posting and commitment, thus is critical to be resolved by JBoss for more transactional and secured performance. You can still amend Web and accounts code, just that you can't test them in Eclipse and have to compile them and run as binary to do that.

How to Modify a Callout

Compiere tries to absorb user changes by its model of Application Dictionary. However whenever the AD is insufficient to do that, the next promising option is modifying or creating new callouts.

Searching for the Callout

Callouts are attached in context to fields of any table defined in Compiere's AD Table & Column. Under the Column Tab you will find the Callout field which will call the Callout class whenever the field undergoes input activity. Callout can change the values of other fields in the same Window in scope or in use at that time. Callout can also be used to do data processing but may be clumsy if that overlaps with the proper Java components. Then the App design is looked at again for more elegant planning of changes. You can view examples of Callouts via Eclipse by going to the Base resource and expanding to `org.compiere.model` tree.

Defining the Callout

The callout implements the CalloutEngine interface and uses the Window context. As its in Java, you can call any imported method to be reused for your purpose. But the main methods inherent in a Callout is the getValue and setValue methods which correspond to the field in context. You can think of the Callout as trying to give you something of an excel spreadsheet.

Opening the Callout in Eclipse

When you open the right Callout java class, usually its `CalloutSystem.java` for creating new small snippets. Bigger snippets may have to be in its own class for tidiness.

Callout Design

The Callout allow us to take the values of fields of any window in context to be manipulated before putting them back to the fields.

```
public String Assignment_Product (Properties ctx,
                                  int WindowNo,
                                  MTab mTab,
                                  MField mField,
                                  Object value)
{
if (isCalloutActive() || value == null)
return "";
// get value
int S_ResourceAssignment_ID = ((Integer)value).intValue();
if (S_ResourceAssignment_ID == 0)
return "";
setCalloutActive(true);

int M_Product_ID = 0;
String Name = null;
String Description = null;
BigDecimal Qty = null;
String sql = "SELECT p.M_Product_ID, ra.Name, ra.Description, ra.Qty "
+ "FROM S_ResourceAssignment ra"
+ " INNER JOIN M_Product p ON (p.S_Resource_ID=ra.S_Resource_ID) "
+ "WHERE ra.S_ResourceAssignment_ID=?";
try
{
PreparedStatement pstmt = DB.prepareStatement(sql);
pstmt.setInt(1, S_ResourceAssignment_ID);
ResultSet rs = pstmt.executeQuery();
if (rs.next())
{
M_Product_ID = rs.getInt (1);
Name = rs.getString(2);
Description = rs.getString(3);
29 Qty = rs.getBigDecimal(4);
30 }
31 rs.close();
32 pstmt.close();
33 }
34 catch (SQLException e)
35 {
36 log.error("Assignment_Product", e);
37 }
38             //
39 log.debug("S_ResourceAssignment_ID="
39                         + S_ResourceAssignment_ID
39                         + " - M_Product_ID=" + M_Product_ID);
40 if (M_Product_ID != 0)
41 {
42 mTab.setValue ("M_Product_ID", new Integer (M_Product_ID));
43 if (Description != null)
44 Name += " (" + Description + ")";
45 if (!".".equals(Name))
46 mTab.setValue("Description", Name);
47 //
48 String variable = "Qty";
49 if (mTab.getTableName().startsWith("C_Order"))
50 variable = "QtyOrdered";
51 else if (mTab.getTableName().startsWith("C_Invoice"))
52 variable = "QtyInvoiced";
53 if (Qty != null)
54 mTab.setValue(variable, Qty);
55 }
56 setCalloutActive(false);
57 return "";
58 } // Assignment_Product
```

Above is part of a Callout class. It has a Callout method that is linked to the Resource Assignment field of the OrderLine Table. It begins with certain arguments:

```
(Properties ctx, int WindowNo, MTab mTab, MField mField, Object value)
```

the WindowNo will inform the system which window is referred to. This we can understand as when the callout happens, we were in a window screen. So when the callout finishes its job, the result update will appear in the same window.

MTab concerns the Tab (that is linked to a table & field context) that is in focus. You can hover your mouse pointer over any word and see the highlights. If you press the Ctrl key while you hover over them, and click, you may really dial in - to the class that handles the objects. You can explore further by opening the Parent class that it extends from such as the `CalloutEngine.java`, `MTab.java` among others.

Look at SQL statement on line 15 onwards. Its pulling out from the Resource Assignment table (line 16). So here it is retrieving from what a pop-up (line 26 to 29) and populate the OrderLine Description with it (line 42 to 54). It will also wrap the Resource Assignment Description with brackets before presenting (line 44).

Notice also the pattern format

```
mTab.getTableName().startsWith("C_Order")
```

to check the table name in context.

Getting and setting values

This is easily achieved as shown in the example:

```
Qty = (BigDecimal)mTab.getValue("Qty");
```

```
       Price = ((BigDecimal)mTab.getValue("Price"));
       BigDecimal Total = Qty.multiply(Price);
       mTab.setValue("Total", Total);
```

The getValue pattern basically obtain the value from the Window field in scope. The setValue will then place a new value into the Window Field. The Total field changes as you put in a new value into either Qty or Price.

Accessing other tables' values

Let's say you want the Price to come from the Product table which is not refered to by the window. Here's is an idea of what you must do prior to the above, in the form:

```
     String sql = "SELECT p.M_Price "
     + "FROM M_Product p "
     + "WHERE p.M_Product_ID=?";
     try {
     PreparedStatement pstmt = DB.prepareStatement(sql);
     pstmt.setInt(1, M_Product_ID);
     ResultSet rs = pstmt.executeQuery();
     if (rs.next())
     { M_Price = rs.getInt (1); }
     mTab.setValue ("Price", M_Price));
```

How to Debug Compiere

Identifying the problem

It is said that the harder thing than getting answers is asking the right questions. To identify the real bug is often the issue. Often than not, you may think its a bug when its actually your own setup data. The rule of Garbage In Garbage Out applies. To understand if its a business rule or logic that you have mishandled is not easy as you'll need Compiere subject matter knowledge. You should have undergone Compiere training or gone through the User Manual on sale at USD40 in Compiere's Webstore. Even upon having such training and User Manual doesn't guarantee you an easy passage to all problems.

Setting the debug level

Probably the first ever step in knowing a bug is from the debug prompt window which pops up when you run `RUN_Compiere2.bat -debug`. In the Compiere window, you can set the debug level to 10 so that you can display as much feedback from the code execution. When debugging in Eclipse we also have the console view which shows similar debug logs. The advantage of using Eclipse to view the debug messages is that you can click on the java exception in the console view to jump straight to the java code in question!

Please refer to `http://compiere.red1.org/ZeroPrice.zip` for an illustrated sample.

Reading the debug logs

The debug logs are earnest messages trying to convey to you the most important clues of what's happening during execution. Of course it cannot guess everything but at least it include the time of execution in hundredths of a second! It also indicates which java class is executing. You can correlate this with the upper left panel which shows the java thread in session. The debug prompts are the result of Compiere incorporating log4j technology into the source codes. Without it, we probably will have abandoned Compiere! As much as 90% time savings is possible using the debug logs.

Understanding the problem

There shouldn't be any errors in Compiere Java codes, otherwise they wouldn't have made their binary compiled form in the first place.

Try to check the bug section in the SourceForge forum. And more often than not, you find that bug been reported by someone else. Otherwise you should define the bug there, giving good info on how it happen so that others and Compiere's team can zoom in. Don't expect bugs stated there to be solved lightning fast as it is a huge base. But they do get resolved one way or another perhaps by the next quarter.

Surprisingly, bugs are often in the form of very minute business logic. This the greatest stumbling block in Compiere. Not been a core developer of Compiere myself, my guess is as good as yours. Why and what the codes are trying to do can be the bigger preoccupation then trying to change them. My advice is keep tracing the logic and flow that concerns you, writing down the activity faithfully, then ask some simple questions to a subject matter colleague or even the client who may have better clues to its business logic. Or even post it in my forum under ASK RED1. Do contribute back what you find out. In due time we will get a complete picture.

You may be strong in Java but that only gives you a slight advantage. If you are, then you may be most helpful in cases where java bugs that isn't due to Compiere coding but Java's APIs themselves! Speaking of which - any Java, JBoss or even Database - references, deprecates, or versions changes; will certainly be a mine of future bugs. So far I have only encounter this in the Swing or GUI interface.

All in all, from my one year experience in Compiere, most bugs are cases of typos and omissions, or due to unexpected business logic. Its the later case that concerns you the most as you try to take advantage of available Open Source - customising to any peculiar unique need of a client user.

Handling SQL codes

Embedded SQLs take the form of this sample:

```
String sql = "SELECT * FROM AD_Role WHERE AD_Client_ID=?";
ArrayList list = new ArrayList ();
PreparedStatement pstmt = null;
try
{
pstmt = DB.prepareStatement (sql);
pstmt.setInt (1, Env.getAD_Client_ID(ctx));
ResultSet rs = pstmt.executeQuery ();
while (rs.next ())
list.add (new MRole(ctx, rs));
rs.close ();
```

You only debug this when it returns an error during execution or testing in Eclipse. You may copy and paste the part between quotes into an editor to test its functions. Usually SQL queries are pure `SELECT` or `VIEW` statements. Then they are safe to test as they won't disrupt your data in the Oracle Database. If its `UPDATE`, `CREATE` or `DELETE` then its very dangerous as executed statements may affect the database. You may still play with `UPDATE SET` statements but take note what they `SET` and revert them in the database table when you need to. Ensure the `WHERE` clause is used to limit the execution. In the example above, the "`WHERE AD_Client_ID=?`" is solved by putting "`Env.getAD_Client_ID(ctx)`" into Eclipse watch function for the "?" value. Then paste that value over "?" in the SQL.

Testing SQL statements

The independent SQL codes as found in the procedures section of the database or in source i.e. `D:\Compiere251\compiere-all\db\database\Procedures` are easily debugged in Toad's Procedure Editor. However you have to sandbox such procedures as they often fetch parameters from AD_PInstance. You merely remove the passing syntax and replace with a declaration or initialising of it so that the procedure thinks it got the parameter already and proceed to execute. Toad in this fashion acts just like Eclipse in its IDE functions - giving you the step in and step over capabilities to examine each step the code takes. You have to click on the left horizontal bar in Toad's editor to make a break appear where you want it. You can peep into the variable's memory by hovering the mouse over it when the execution is suspended.

Tracing SQL Procedures

To know what SQL independent codes to debug, you first trace from the AD (App. Dictionary). Login in as System, look for them within the Table & Column, to first trace which AD Report & Process it is refering to. Then go to that AD Report & Process to find it. SQL are easily identified as ending in *.sql. Now you know which business application process is tied to which SQL. Make sure you get the right one before debugging. You do not need to compile source yet, to test the codes on your database. You confirm the perfection of your codes in Toad itself. When you are ready to compile, save the procedure or replace the ones in `..compiere-all\db\database\Procedures\`.

Integrating into Compiere

Likewise to integrate new procedures into Compiere will involve creating the necessary Report & Process in the AD, and calling that from the Table & Column part. You can refer to Compiere's own present examples to see how its done.

Handling Java codes

The Source Layout discussed earlier explains how the source is organised. Read the comments in the java codes you open. Usually they are well documented. Debugging may run into other folders and packages mentioned. GUI or interface issues happens in the `org.compiere.grid` packages. Database handling happens in the `org.compiere.db` areas.

`GenerateModel.java`

Generating new table-field interface of setter and getters are done through `GenerateModel.java`. This is done after any new changes table & column in the AD. The rules that you define in the AD will be incorporated during running of GenerateModel, which has a main method to run on its own. After that you can examine the generated codes which are `X_*.java`. They are located in org.compiere.model of the dbPort base in Eclipse. You cannot amend the codes directly as they will be overwritten whenever you regenerate. Try to change your business rules in the AD instead.

Finding out the root cause

When you identified what java classes are handling the routine from the debug window, you may look them up in Eclipse. You use the search function on the menu bar and key in the whatever.java and make your search faster by using the `*.java` wildcard. Upon locating the file, you can open it and place breaks in it, to get it to stop when that part is executed. It will take you a while to nest into the right spot. At every step you have to constantly guess what the code is trying to do. Its best to consider the following:

1. If the issue is a simple basic business or accounting matter, Compiere most likely has already such a capability. For example at one time, a user couldn't do a Bal b/f on the Financial Reporting Engine, and asked me to modify the codes. In the end I found out that it was due to not understanding how to use the engine. I found the solution because I looked in the right direction, which is based on the believe that surely Compiere must be able to handle such a basic function.
2. Look again at Compiere's overall solution. Can you reuse some of them without touching the codes? For example at one time I wanted to refactor the Aging classes to make it report on sales performance. After a while someone else told me that you can achieve that by using the Financial Report Engine!

3. Know where the logic is really set. Definitely not in `X_*.java`. Usually not in `M_*.java` unless its a logic bug. Specific business logic is mostly done in org.compiere.process classes. And you can guess them easily from their namesakes i.e. `InvoiceGenerate.java`.

For a live case please refer to the case in http://compiere.red1.org/ZeroPrice.

Amending Codes

To correct bugs, there are some examples you can refer to the Bugs Galore! section of http://red1.org/forum/.

Most code changes are assisted by Eclipse auto-code-assist. You cannot change X_ type of codes as explained before. Keep your new snippets either in Callouts or Process and make them consistent in location and style. Use easily understood names in new variables.

Recompiling Compiere

Here is the last leg of the affair. Code changes are already made and you are ready to compile, build and deploy them to the target environment. Its best to have two target environments - one been the development before trying out on the live production environment. Any mistakes detected in development environment would not affect the live one, giving you a chance to review the impacts and make more plans.

Avoiding change impact

Important rule is to realise that any change you do will create and impact and thus becomes a risk. You have to stop and think and better write them down prominently what you think are the impact and risks. But the philosophy here is to always avoid changes unnecessarily. Many occasions I have experienced solutions that were solved with almost no coding! Its either solved by the Application Dictionary, or just understanding Compiere's functionalities and features.

Backing up and risks scenarios

Its also important to state whether there can be reversals of the impacts. Usually it is just making backups of your `CClient.jar` and `ExpDat.dmp` before replacing them. When an error is detected you have the option of using back the backup set.

Before you intend to touch a java class safe the contents in another place. Leave a read-me text note stating why its is there.

Documenting your work

Documenting is a must. There were times when I myself forgot what I did an hour ago. You cannot play the fool when it comes to software. One mistake can just blow up your system, so to speak. Consider it a discipline, and build it into a habit. If you are the manager and the developer is working under you, insist on a documenting standard and sign-off the developer's work or else its not considered work!

Documentation must happen in two or three places. First, prepare and plan your changes in a Change Request Form which will state the issue or problem, detailing the user scenario or business rule affected. It also states the suggested remedy and technical options if possible. It is a living document in the sense that it will act as a continuous lab report on further observations and actions as a result of ensuing investigation into the problem.

The second documenting is in the codes itself. At the beginning of the code snippet that you introduce should have comments such as at least the following:

// red1 - start snippet to handle new factor.

Then at the end of the snippet can be:

//red1 - end of snippet - 3.14pm, 19/1/04.

You can also indent the remarks differently so as to make it stand out. With your initials stated there, we can search for the code changes easily with `Ctrl-F` in the Eclipse panel.

If you are not making new codes but just amending the present one, then you can comment after the change like this:

```
      callmethod(m_action.Prepare) //red1 - old -m_action.Complete.
```

The old portion that you deleted is commented so that you are aware what you changed from.

Even if your change is only a single character, you must still say so! These will be time savers, believe me and also give your other team members and the users more sleep-easy nights. Finally you can write a solution paper like I did in my website. It helps to explain the case to the client and in training the handover team.

Deploying the `CClient.jar`

After the changes you build your codes again. In your own test or development environment, you already specify your target binary folder, where the new jars will be deployed. Usually changes to the non web-store and accounts posting codes will result in the `CClient.jar`. Thus you only copy over that to the `Compiere2\lib\` directory of your target live production environment. The LAN clients using webstart will also automatically refresh itself when they log in as a check will be done to compare with their caches, and will reload again the jars.

Deploying the `CServer.jar`

Changes to the accounts posting will impact the CServer. Deploying this is more hassle than `CClient.jar`. Basically is like redoing it as a full setup. You thus have to delete the old Compiere2 directory and its sub-folders! Then you compile and build to produce Compiere2. Then `RUN_Setup` and proceed as normal.

Common pitfalls

Programming is fun but can be a nightmare for the newbie. The golden rule is do not panic and always take things easy. Slow is faster. Do work in pairs. Help each other. Go to the forums. Spend more time researching. Respect each other's different style and mood. Inspiration happens at odd hours, then be flexible in your plans. If the going gets tough, take a break! Then return to the problem refreshed.

Wrong understanding of business scenario

Much common and popular business logic that exist in enterprise business software is present in Compiere. If it is not, it is in the works. Thus when trying to incorporate your own extension or business logic, its best to reuse what Compiere has. You may refer to the numerous cases in my website, which proves that minimal coding if any is needed. Most often, extra work is wasted due to a misunderstanding of how Compiere is designed with metadata and generators.

Attempting risky changes

Risky changes are those that involve too much changes. Good safe ones are those that are very short and involves usually one or two lines. Suspecting wrong codes can lead to more risks. Study more, work less.

Poor Planning

Do not forget to plan first if the risk is there. Write your plan and review it. If it involves the client, inform them of the risks so that you pass the decision to them. If so, let them sign your plan, with the words that they are aware and taking the risks themselves. Reduce the risks by making proper and the right backups. Before you replace the CClient.jar, you must keep a copy of it first. If things go wrong, do not panic or try anything hastily. Stay calm and return to the planning room.

Poor documentation

Document your codes and keep a logbook of whatever changes you do to the particular client instance. Create checklists whenever you can so that you can repeat your work whenever you need to in a fallback situation. If you are a weak documenter, start practising, or you stay cheap.

Checking out unstable version

Check with the forums for the right time to checkout your codes. Sign up as a Compiere Partner if you want more inside knowledge of what is planned into the next release. Otherwise check with the bug and support section of sourceforge.org to find out when certain things are fixed. Then you checkout the codes. Immediately do a `RUN_Build` to confirm that the codes are clean from mistakes. If there are, then you have to abandon them and wait for the next right time to checkout from CVS.

Migration Process (by Redhuan D. Oon)

Migration though is handled by Compiere directly and among Compiere's Partners, still persist with certain issues, which will be covered here as a checklist.

Backup your present instance

- Bring down your system. Chase all the users out.
- Uninstall your Compiere JavaService!
- DBExport
- Rename your Compiere2
- Place the new binary Compiere2
- Copy over the latest nightly build of Compiere.dmp from the Partners' Forum
- RUN_Setup!
- ImportReference
- Migration

Ensure that the target indicates your present version, and the source shows the new version. The source must be newer than the target.

Run the three steps with the test mode off. As you have the backups done you shouldn't waste time! Do not need to run the tests again if hit errors. As it is already set to run twice to iron out missing calls. Errors are logged int the log file stated. Keep that info for reference. Check what kind errors happened. If they are non-critical ones, and their figures are a few i.e. 3 errors, you can proceed to use it.

Post migration tests

Check cache reset

Login as a recently created user, check role and org and locator is ok

Login as GardenAdmin, create and complete a sales order cycle thru invoice (Customer). Have 2 order lines in the Order.

Print some reports.

Take note of bugs or 'ding' sound when debug hits error.

When no error do DBExport

Release to users

If fail with errors, you have to trace from the debug mode and using Eclipse - to give insight as to what can be the cause. Usually it's due to the new version's introduction of new fields or settings. So checking the new release information is useful. Once it was a new language rule, which we solved by looking into the AD and finding it, we set to another option, and it becomes ok.

5 Contributors

1. Peter Shen, point 010.
2. Redhuan D. Oon, points 011 and 012.

   email: red1@red1.org

   feedback: www.red1.org/guestbook/

   to receive updates join www.red1.org/forum/.

6 GNU Free Documentation License

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
   D. Preserve all the copyright notices of the Document.
   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
   F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
   G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
   H. Include an unaltered copy of this License.
   I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
   J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
   K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

ADDENDUM: How to use this License for your documents

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

7 Final Note

The original version of this HowTo was completely written using emacs. Marco LOMBARDO, 2004-06-27, Lucca, Italy. marco.lombardo@enneenne.com

8 Concept Index