This is a living document that I am doing for Marco LOMBARDO's Compilo project at https://sourceforge.net/projects/compilo/ :D so it is a toddler now, and will continue to grow there and the usual under http://compiere.red1.org/

The content has assumptions from the author and thus cannot represent Compiere Inc, other associates in the Compilo project or other parties. Bias is my own.

RED1 COMPIERE WORKSHOP

MAKING IT HAPPEN!!!

011 Compiere Source 101

012 Migration Process

version 0.3

(By red1 d. oon poh shin)

email: red1@red1.org

to feedback sign: www.red1.org/guestbook/

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

011) COMPIERE SOURCE 101 3

011.1) Overview of Topics 3

011.2) Open Questions 3

011.2.1) Where did it come from? 3
011.2.2) Is it truly Open? 3
011.2.3) What about been Open Knowledge? 3
011.2.4) What about Market Forces? 4

011.3) The Source 4

011.3.1) The Source Layout 5
011.3.2) Cracking the source 5
011.3.3) Business Functionality 6
011.3.4) Development Work Cycle 7

011.4) Compiling from Source 8

011.4.1) Build Process 8
011.4.2) SQL Debugging 8

011.5) Researching from the Web 9

011.5.1) Using the Forums 9

011.6) How to Setup Eclipse for Compiere 10

011.6.1) Downloading Eclipse 10
011.6.2) Setting up perspectives and views 10
011.6.3) Importing Compiere Source into Eclipse 10
011.6.4) Setting up Debug and RUN modes 11

011.7) How to Modify a Callout 11

011.7.1) Searching for the Callout 11
011.7.2) Defining the Callout 11
011.7.3) Opening the Callout in Eclipse 11
011.7.4) Callout Design 12
011.7.5) Getting and setting values 13
011.7.6) Accessing other tables' values 13

011.8) How to Debug Compiere 14

011.8.1) Identifying the problem 14
011.8.2) Setting the debug level 14
011.8.3) Reading the debug logs 14
011.8.4) Understanding the problem 14
011.8.5) Handling SQL codes 15
011.8.6) Handling Java codes 16
011.8.7) Finding out the root cause 16
011.8.8) Amending Codes 17

011.9) Recompiling Compiere 18

011.9.1) Avoiding change impact 18
011.9.2) Backing up and risks scenarios 18
011.9.3) Documenting your work 18
011.9.4) Deploying the CClient.jar 19
011.9.5) Deploying the CServer.jar 19

011.10) Common pitfalls 20

011.10.1) Wrong understanding of business scenario 20
011.10.2) Attempting risky changes 20
011.10.3) Poor Planning 20
011.10.4) Poor documentation 20
011.10.5) Checking out unstable version 20

012.0) MIGRATION PROCESS 21

012.1) Backup your present instance 21

012.2) Post migration tests 21

011) Compiere Source 101

(by Redhuan D. Oon. http://red1.org/)

011.1) 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.

011.2) Open Questions

011.2.1) 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.

011.2.2) 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.

011.2.3) 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.

011.2.4) 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.

011.3) 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.

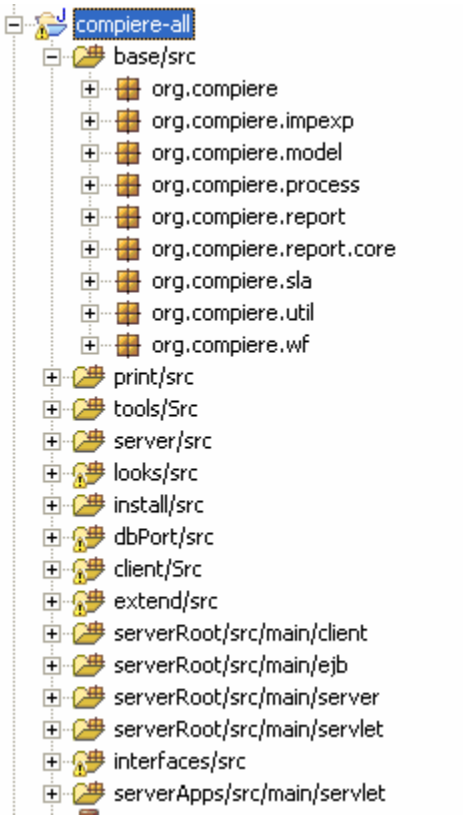
The source is considered well-annotated and well written. It is fully integrated in its design from the ground up and thus poses a huge challenge to even thinking of changing it. The source is also getting heavier with each version release, unzipped at around 200 Mbytes for version 251e. When compiled,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

011.3.1) 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.

011.3.2) 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.

011.3.3) 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.

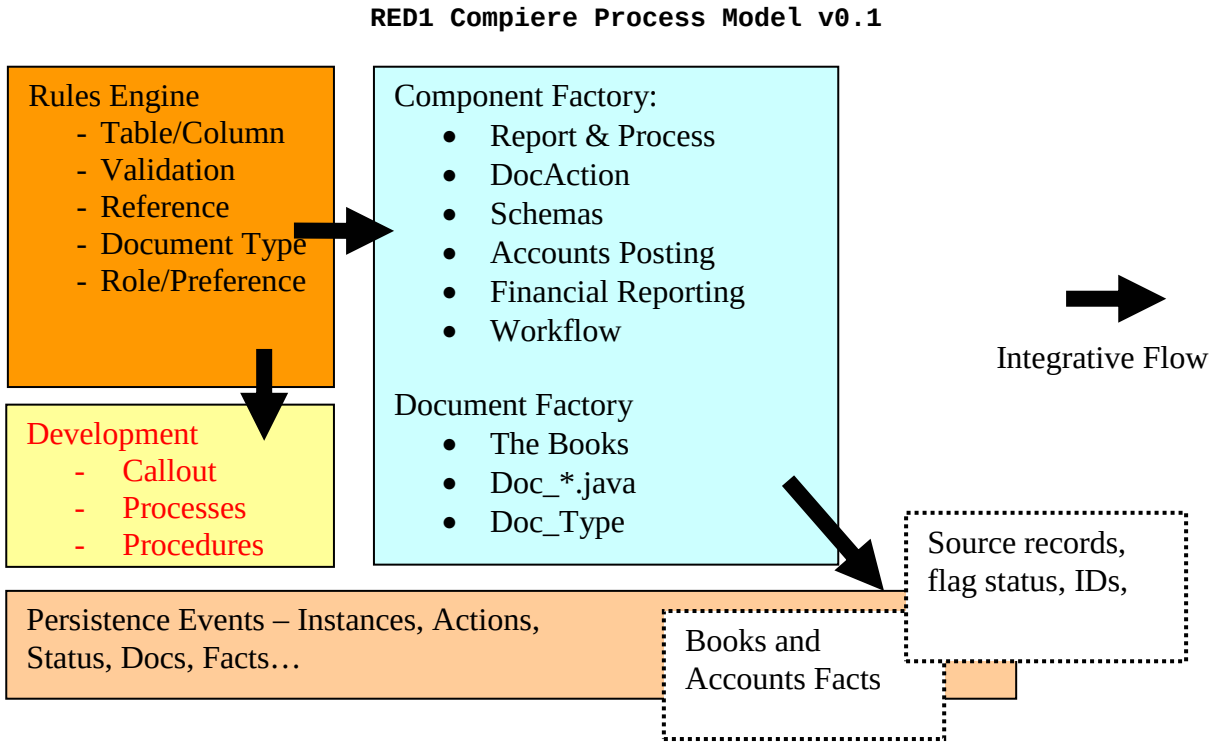
Much of the document handling logic is controlled by values in the Document Type such as GL Type and Base Type.

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.

011.3.4) 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.

011.4) 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.

011.4.1) 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.

011.4.2) 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. D:\Compiere251\compiere-all\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.

011.5) 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.

011.5.1) 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.

011.6) 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.

011.6.1) 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.

011.6.2) 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.

011.6.3) Importing Compiere Source into Eclipse

First you download Compiere source from SourceForge. Choose version such as CompiereSource251d 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.

011.6.4) 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.

011.7) 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.

011.7.1) 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.

011.7.2) 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.

011.7.3) 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.

011.7.4) 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.

```
1 public String Assignment_Product (Properties ctx, int WindowNo, MTab mTab, MField mField, Object value)
2       {
3               if (isCalloutActive() || value == null)
4               return "";
5       //      get value
6       int S_ResourceAssignment_ID = ((Integer)value).intValue();
7               if (S_ResourceAssignment_ID == 0)
8                       return "";
9               setCalloutActive(true);
10
11              int M_Product_ID = 0;
12              String Name = null;
13              String Description = null;
14              BigDecimal Qty = null;
15              String sql = "SELECT p.M_Product_ID, ra.Name, ra.Description, ra.Qty "
16                      + "FROM S_ResourceAssignment ra"
17                      + " INNER JOIN M_Product p ON (p.S_Resource_ID=ra.S_Resource_ID) "
18                      + "WHERE ra.S_ResourceAssignment_ID=?";
19              try
20              {
21                      PreparedStatement pstmt = DB.prepareStatement(sql);
22                      pstmt.setInt(1, S_ResourceAssignment_ID);
23                      ResultSet rs = pstmt.executeQuery();
24                      if (rs.next())
25                      {
26                              M_Product_ID = rs.getInt (1);
27                              Name = rs.getString(2);
28                              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=" + S_ResourceAssignment_ID + " - 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.

011.7.5) 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.

011.7.6) 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));
```


011.8) How to Debug Compiere

011.8.1) 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.

011.8.2) 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.

011.8.3) 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.

011.8.4) 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.

011.8.5) 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.

011.8.5.2) 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.

011.8.5.3) 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\.

011.8.5.4) 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.

011.8.6) 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.

011.8.6.2) 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.

011.8.7) 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:

a) 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.
>
> b) 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!
>
> c) 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.

011.8.8) 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.

011.9) 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.

011.9.1) 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.

011.9.2) 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.

011.9.3) 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.

011.9.4) 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.

011.9.5) 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.

011.10) 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.

011.10.1) 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.

011.10.2) 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.

011.10.3) 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.

011.10.4) 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.

011.10.5) 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.

012.0) Migration Process

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.

012.1) 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.

012.2) 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.

~ end ~

"Working on the ferry along the Mississippi River is not work. It is PLAY"

- Mark Twain

"Do you notice that RICH people are DIFFERENT from us?"

"Yeah, They have MORE money."

- Young Ernest Hemingway with a friend

"IF I Ever Lose My Hands, I DON'T Have To Work No More"

- *MoonShadow* sang by Cat Stevens

"GOD has No Sense of Humour. HE Knows The Punch Lines"

- Edward de Bono

