RED1 COMPIERE WORKSHOP

MIX-MATCH

Inquisition of a Missing Match Invoice version 1.0

By Redhuan D. Oon ***a.k.a. red1***

Table of Contents

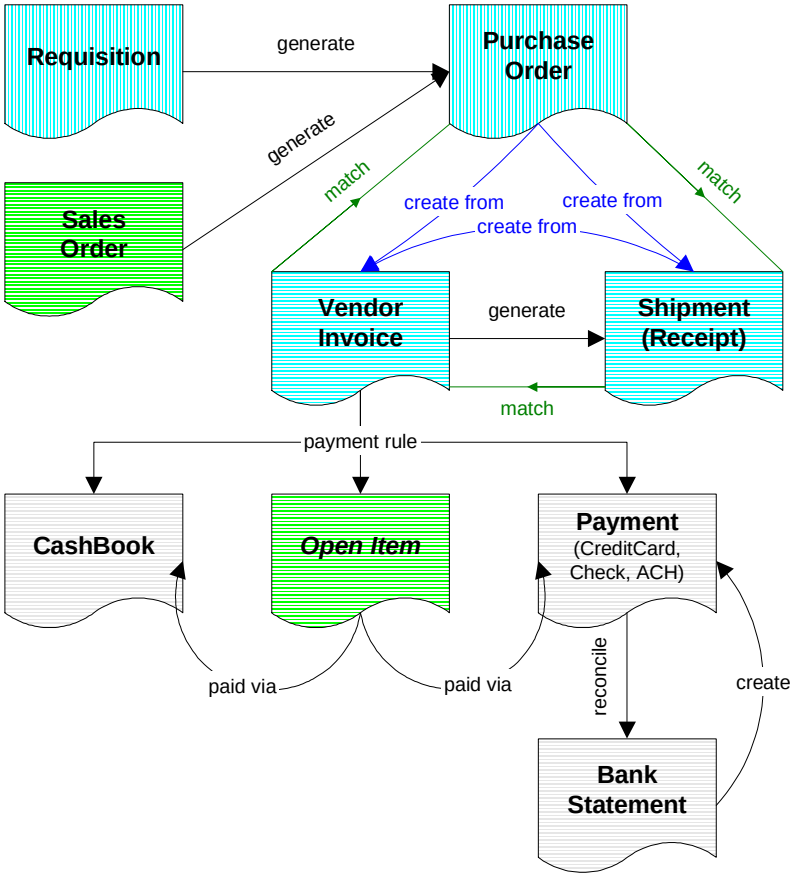
Introduction

Due to remorse and repentance, I am conducting here an exorcism to examine part of the Requisition-to-Invoice process up close. Here while chasing some bugs, we begin to understand some Java code patterns used and how to reuse them. We also get to see the underlying table structure and its influence over Compiere's design – albeit without excessive flesh.

This article is for those who want a rigorous stroll in the park to understand Compiere's innards. Its for a beginner developer like me too. Just that I dare to commit it to writing and invite the wrath of the Lords of Javanese accent, not to mention also to suffer ignominy at the feet of the priests of Khomphere.

However its best not to read this after a gluttonous meal as some innards may be too exposed. The Java class structure and hierarchy will be interesting desert I will say.

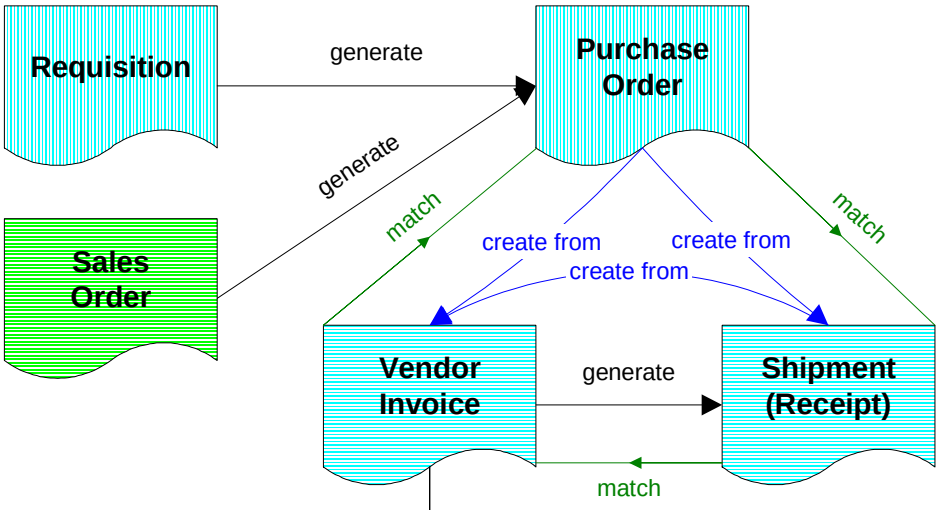
Ok then, lets dispense with the rituals of ice breaking – let us break some jars. But first, some anagram from three feet away:



The 'Create From' Process

We have a client who has to migrate from an old system, and convert its backlog of invoices and shipments records into the new Compiere system. The client thus require the process to be **Invoice > Material Receipt > Matching** – to keep its document numbering in place.

The usual or normal process cycle was to be **Purchase Order > Material Receipt > Invoice > Matching**. Now, both scenarios are flamboyantly allowed as shown in the diagram below (in the 'create from's):



It seems that Compiere understands the pressures of disagreeable business scenarios such as coming from our client where a conversion of a legacy system into Compiere must deal with the backlog of invoices as described. This is so that - the document numbering is intact as per source materials, to avoid the auto-numbering and overwriting of the numbers in the documents. So the process is:

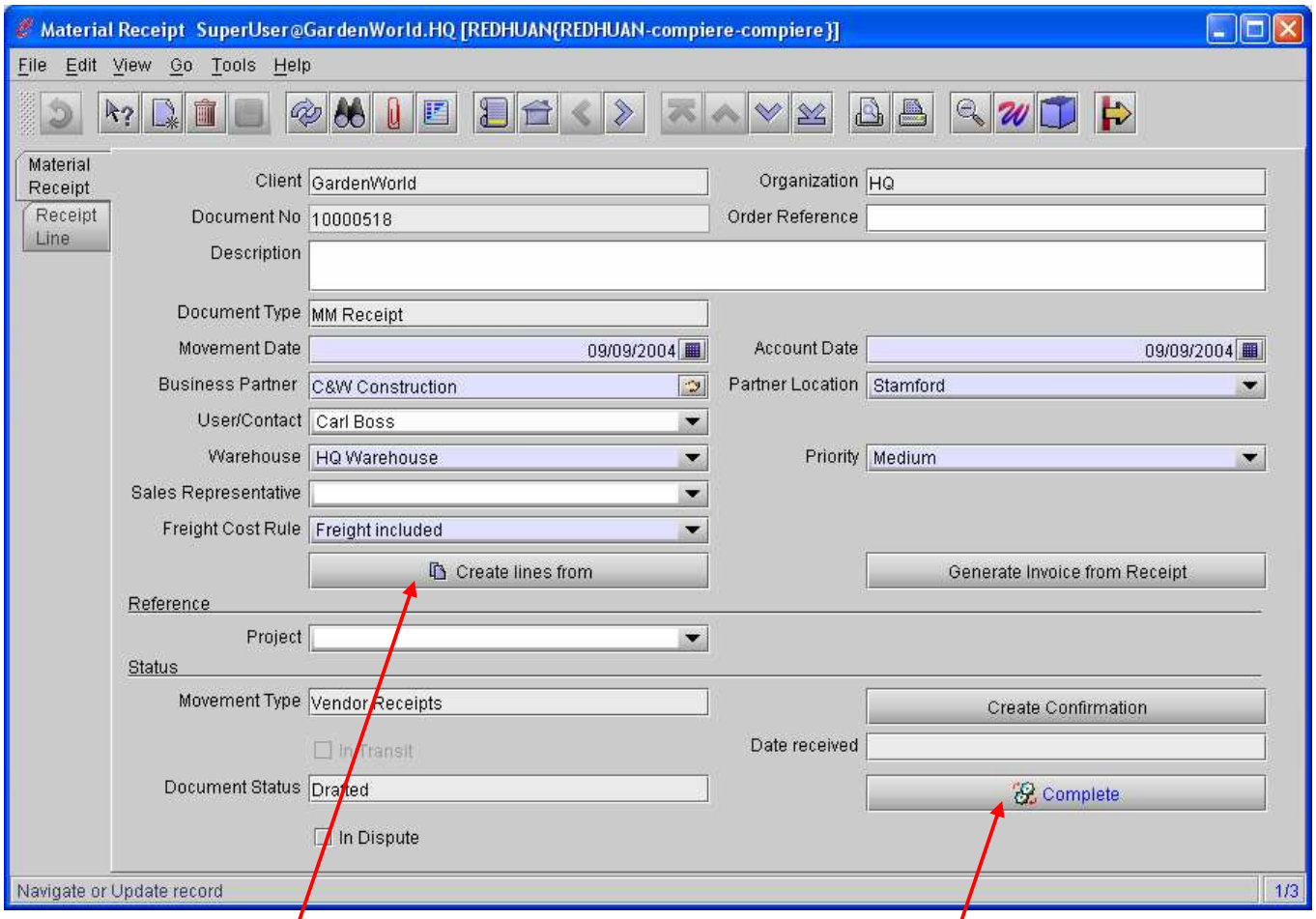
1. Recreate the backlog invoice, maintain the same numbering as source.
    a. Process Complete it.
2. Go to Material Receipts, create a new record.
    a. Create Lines From – from the Invoice created in (1).
    b. Process Complete it.

In version 251e there are two sins of omissions from the above 2 steps:-

The 'create from' within Material Receipt (Shipment) does not update the Invoice.

The 'process complete' within Material Receipt also does not do a Match Invoice.

Ah! So the high-priest lied, against the Temple's own holy tablet. But fear not, as soon as this bug was reported, a new amendment was sneaked into the commandments. Lo! Its now solved in the next version - 251f.



This button will launch VCreateFromShipment which should put the InOutLine_ID into InvoiceLine record string

This button will launch MInOut which should do a Match Invoice creation.

What these two functions forgot to do is stated in the two text boxes.

The Tables of the Last Supper

Now let's look at the database level. The Invoice previously created is stored in the C_Invoice table. The Material Receipt is stored in the M_InOut table. Its children are C_InvoiceLine and M_InOutLine. Their primary keys have the _ID suffix.

Let's peer into these tables. If we open the **C_Invoice**, and **M_InOut** tables, we find some interesting constructs.

M_InOut table has 2 fields - *C_Invoice_ID* and *C_Order_ID*. The later representing Purchase Order records. This is understandable as the Material Receipt (Shipment) usually happens after the PO completion. And as it also possess the *C_Invoice_ID* it also goes to show that the reverse generating process is also accepted – the generating of a Receipt based on an Invoice.

Now if we go to the tables' respective child-lines level, which are the **C_InvoiceLine** and **M_InOutLine** tables respectively, we find that the first has a *M_InOutLine_ID,* whereas the second has *C_OrderLine_ID* (PO line). This is a slight error in database design as the Receipt cannot thus remember who its InvoiceLine counterpart is. This lends to the first omission.

But with the InvoiceLine having a M_InOutLine_ID field will thus show it can still remember the ReceiptLine counterpart and thus absolved from the said error. However the purpose of that line ID, was to be populated during a 'Create Lines From' process, but from within the Invoice container, and not within the Receipt (Shipment) container. To correct this error, we have to figure out how to populate that from the Receipt. This means we now have the arduous task of locating the java class that handles both this angles and calls for that impregnation to occur on the virginal side.

Glad you are still awake.

I managed to trace the java classes involved by watching closely the debug console and issuing numerous breaks here and there.

Let's examine the 'Create From' process in both containers, Invoice (Vendor) and Material Receipt (Shipment) and see what java classes are been called. By the way, Shipment and Material Receipt seems to be interchangeable verses – presumably Compiere took it from a floating Gospel before it came to shore. Below we can see the 2 Java classes called.

- Create Material Receipt, then create Invoice with 'create lines from' Receipt:
  - Calls Java class – VCreateFromInvoice.java
- Create Invoice (Vendor), then create Material Receipt with 'create lines from' Invoice:
  - Calls Java class – VCreateFromShipment.java

The second Java class is what we want to home in.

Upon dissecting into both classes reveal that the first class populates the M_InOutLine_ID, whereas the second does not. So now to plug the bug, it's a matter of repeating the method used in the first container onto the second class.

Let's take a closer look at the first snippet to see how it is done:

The Java Classes

In all, we will only amend two javas – VCreateFromShipment and MInOut.

Examining VCreateFromInvoice.java

```
package org.compiere.grid;
public class VCreateFromInvoice extends …
…
                  //    Shipment Info
                        if (inoutLine != null)
                              invoiceLine.setShipLine(inoutLine);
```

The above setShipLine a method in MInvoiceLine's:

```
      public void setShipLine (MInOutLine sLine)
      {
            setM_InOutLine_ID(sLine.getM_InOutLine_ID());
```

The last line here sets it!

Amending VCreateFromShipment.java

So now, how do we do the same, reversely in VCreateFromShipment? We say reversely because in the later class, we are in a different container – in Shipment and not Invoice. We thus have to call the Invoiceline, and insert the InOut_ID before saving it. And indeed VCreateFromShipment didn't contain such an insertion:

```
      if (!iol.save())
            Log.error("VCreateFromShipment.save - Line NOT created #" + i);
            }   //   if selected
      }   //  for all rows
```

It only seems to commit the InOutLine (iol.save), without any hoot to an InvoiceLine. (It did reference a C_OrderLine (PO) earlier on (not shown here), but when we are pulling an Invoice, no POLine work is needed).

So we now have to do the InvoiceLine job in Shipment container.

We do that by injecting the above with our own snippet (in bold) making it look like this:

```
if (!iol.save())
      Log.error("VCreateFromShipment.save - Line NOT created #" + i);
//red1 save InvoiceLine with this M_InOutLine_ID
 m_invoiceline = new MInvoiceLine (Env.getCtx(), C_InvoiceLine_ID);
//red1  -- get the InvoiceLine
 m_invoiceline.setM_InOutLine_ID(iol.getM_InOutLine_ID());
//red1 -- get the InOutLine, set it to InvLine
 if (!m_invoiceline.save())
      Log.error("VCFS - red1 InvLine.save - Not Updated #" + i);
//red1 saving its InOutLineID -- end --
      }   //   if selected
}   //  for all rows
```


We deduce the lines in bold by examples within VCreateFromInvoice. You have to undergo a baptism of fire through many javas before finding the chosen one.

We then check the **InvoiceLine** in the Oracle Database and found it to have worked. Now that we have solved the populating of **InvoiceLine** with the *M_InOutLine_ID*, we turn our holiest intentions to the Match Invoice creation.

Examining MInOut.java

We deduce the java class that handles the process by looking at the debug console again. Noticing that MInOut java was called along the way when we press the 'Complete' button. This is logical as we are in the Material Receipt window where the button is.

MInOut.java thus handles the creation of the Material Receipt record, but didn't create the Matching Invoice record that is needed to affect the match. When we examine the innards of it, we found the reason:

```
package org.compiere.model;
public class MInOut extends…
…
//    PO Matching
      if (!isSOTrx() && line.getC_OrderLine_ID() != 0)
      {
            log.debug("completeIt - PO Matching");
            BigDecimal matchQty = line.getMovementQty();
            MMatchPO po = new MMatchPO(line, getMovementDate(), matchQty);
                  if (!po.save())
                  {
                        m_processMsg = "Could not create PO Matching";
                        return DocAction.STATUS_Invalid;
                  }
      }     //    PO Matching
```

Ahah! It only does PO Matching instead! Which is true, because its thinking that a Shipment occurs after a PO – another proof that the early design is suspect. So what then? Well, we just have to add a method to do Match Invoice. We find that method by looking at its counter java class. This last deduction is elementary, my dear Watson, ;). The counter class is MInvoice.java.

Refering to MInvoice.java

The counter class to MInOut here would be MInvoice, and sure enough we find the scriptures:

```
//    Matching
      if (!isSOTrx() && line.getM_InOutLine_ID() != 0)
      {
            BigDecimal matchQty = line.getQtyInvoiced();
            MMatchInv   inv   =   new   MMatchInv(line,   getDateInvoiced(),
matchQty);
            if (!inv.save())
            {
                  m_processMsg = "Could not create Inventory Matching";
                  return DocAction.STATUS_Invalid;
            }
```


Amending MInOut.java

Now just move this Noah's Ark over to the MInOut.java and we can all go home. So under the PO Matching, we give an **else** routine after the PO Matching and… :

```
}     //    PO Matching
else //red1 Inv Matching
{
      int InvLineID = DB.getSQLValue(
      "SELECT C_InvoiceLine_ID FROM C_InvoiceLine WHERE M_InOutLine_ID=?",
                  line.getM_InOutLine_ID());
      if    (!isSOTrx() && InvLineID != -1)
      {
            mInvLine = new MInvoiceLine(Env.getCtx(), InvLineID);
            BigDecimal matchQty = mInvLine.getQtyInvoiced();

      MMatchInv inv = new MMatchInv(mInvLine, getMovementDate(), matchQty);
      if (!inv.save())
       {
            m_processMsg = "Could not create Inventory Matching";
            return DocAction.STATUS_Invalid;
       }
      }
}  //red1 -- end --
```

Just that the first part is a hack, a rewriting of the scriptures as our context has changed. Since we are in the MInOut.java we do not have the InvoiceLine context. We thus have to recreate it. We have to find out which InvoiceLine bears the InOutLine_ID, done by our first fix job.
We call that InvoiceLine with the

```
      int InvLineID = DB.getSQLValue(
      "SELECT C_InvoiceLine_ID FROM C_InvoiceLine WHERE M_InOutLine_ID=?",
                  line.getM_InOutLine_ID());
```

construct, which on finding it will proceed to give us the InvoiceLine_ID for the Match record string. If it fails, it will return a -1 value and this is checked by an **if**... :

```
      if    (!isSOTrx() && InvLineID != -1)
```

When its not, signified by the '**!**', then it is ready to call the MatchInv.java that will form the Match Invoice record:

```
MMatchInv inv = new MMatchInv(mInvLine, getMovementDate(), matchQty);
```

When I hover my mouse pointer over the method I discover the arguments needed are the InvoiceLine, a Date and a Qty, so I deduce the above 3, deciding that the Date should be the when the Shipment Receipt is created. Then we have to a final confession to the priest:

`if (!inv.save())` - this ensures the record is saved or else.

Conclusion

Upon testing we find that the Match Record is created, and thus the accounting consequence is correct and both bugs considered plugged.

As these two issues are fixed in the next version, you may consider switching to 251f, otherwise you can do this plugs for your earlier version at 251e.

~ the End ~
1.15am GMT+8, 12th September, v0.1
6.41pm, v1.0

Happiness is the X quantity of what you achieve,
Divided by the Y quantity of what you expect.
Thus to be infinitely happy,
Reduce Y to zero.

- **Bertrand Russell**

The World is Enough for ALL People,
But not enough for one greedy person.

- **Mahatma Gandhi**

IN GOD WE TRUST;
The rest of you pay cash.

- the evil of paper money

A tree falls in the forest,
But no one is around to see it fall,
Did the tree really fall?

- **Zen**

To allow the gods to look favourably upon you please sign http://red1.org/guestbook/

To get further blessings please register with http://red1.org/forum/

And slap your other cheek.

