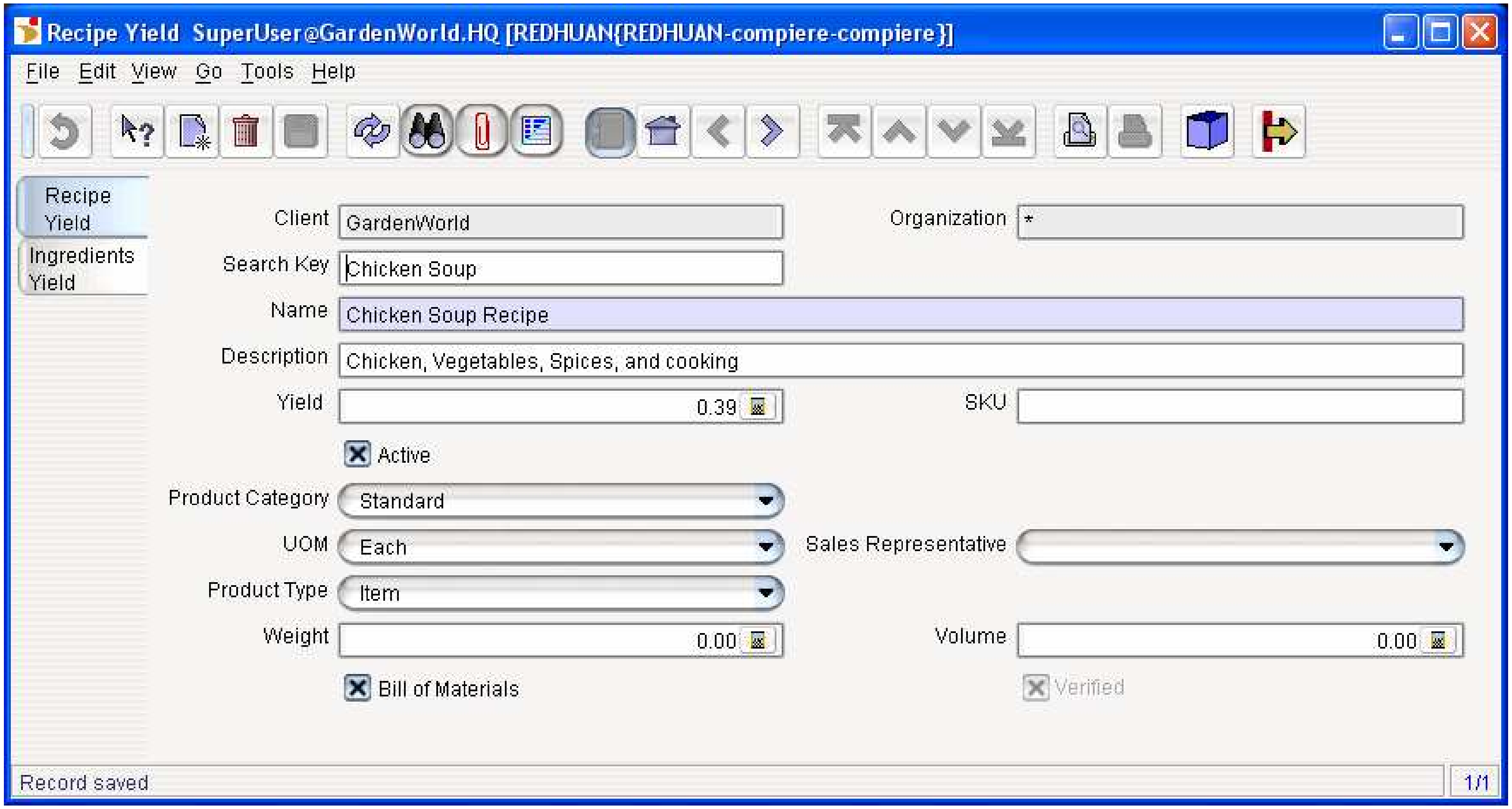

Recipe Yield SuperUser@GardenWorld.HQ [REDHUAN{REDHUAN-compiere-compiere}]
File Edit View Go Tools Help
Recipe Yield
Ingredients Yield

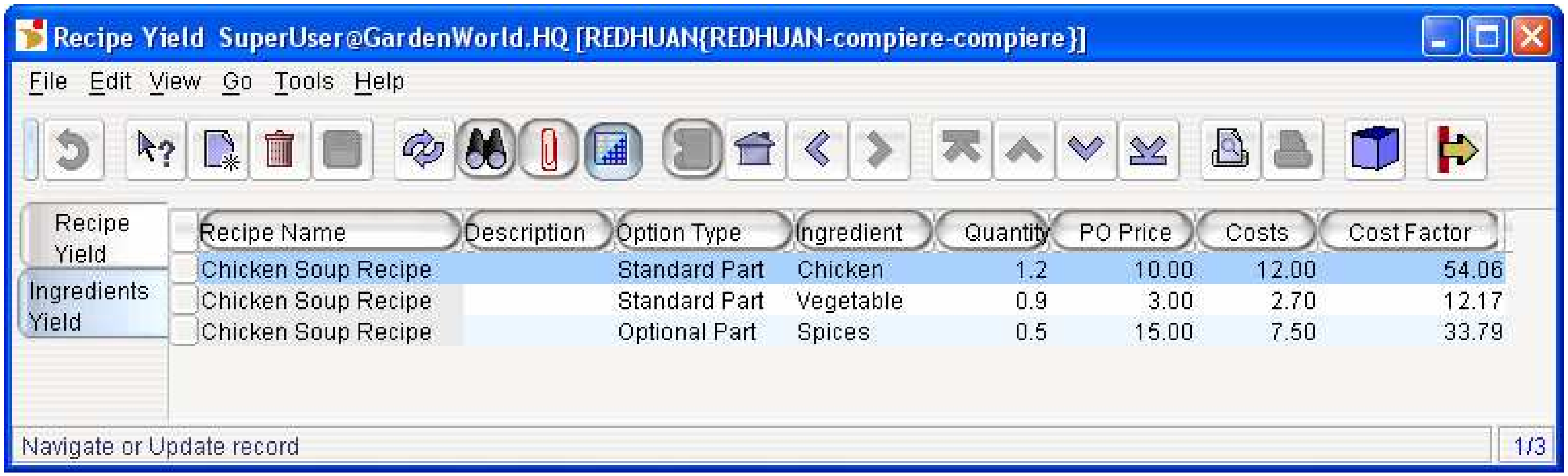
Client: GardenWorld
Organization: *
Search Key: Chicken Soup
Name: Chicken Soup Recipe
Description: Chicken, Vegetables, Spices, and cooking
Yield: 0.39
SKU:
[✓] Active
Product Category: Standard
UOM: Each
Sales Representative:
Product Type: Item
Weight: 0.00
Volume: 0.00
[✓] Bill of Materials
[✓] Verified

Record saved 1/1

Figure 1: Main Tab of Recipe Yield

Adapted from the Product screen. As you can see in Figure 1, you have to rename the Product Tab to Recipe Yield and the BOM Tab to Ingredients Yield. Add one field called Yield as shown. For guide on how to do callout and Application Dictionary changes, please refer to another article in http://compiere.red1.org/Callout.zip.

The Yield Factor above is calculated as the parent single unit (kg) against the BOM's child kg units. Thus the yield or wastage is a decimal fraction when 1 is divided by the total kg units.


Recipe Yield SuperUser@GardenWorld.HQ [REDHUAN{REDHUAN-compiere-compiere}]
File Edit View Go Tools Help
Recipe Yield
Ingredients Yield
Navigate or Update record
1/3

Recipe Name	Description	Option Type	Ingredient	Quantity	PO Price	Costs	Cost Factor
Chicken Soup Recipe		Standard Part	Chicken	1.2	10.00	12.00	54.06
Chicken Soup Recipe		Standard Part	Vegetable	0.9	3.00	2.70	12.17
Chicken Soup Recipe		Optional Part	Spices	0.5	15.00	7.50	33.79

Figure 2: Second Tab of Recipe Yield

In this second tab , we add three new fields which are PO Price, Costs and Cost Factor.
A callout is created in the Table Column field – BOMQty which is also uploaded : http://compiere.red1.org/YieldCallout.zip.
The callout will pull the PO Price from the M_Product_Costing table which is the average costing as the business progresses. The Costs field will take the PO Price and multiple by the Quantity. The Cost Factor will show the breakdown for all the siblings cost percentages which will add to 100%.

To modify the Product Window, first you modify the underlying Table-Column structure in the Application Dictionary.
The second Tab refers to the M_Product_BOM Table. So we need to go into that table and add the fields as well as specify the Callout. Below is a screenshot to show the last three fields added: r_BOMCost, r_BOMPO, and r_Factor. I choose 'r' so that I can spot my stuff easily, follows my initial – red1 ☺. Notice the BOMQty is calling a Callout – "SE_BOMYield". This means the Callout will be coded in CalloutSystem.java

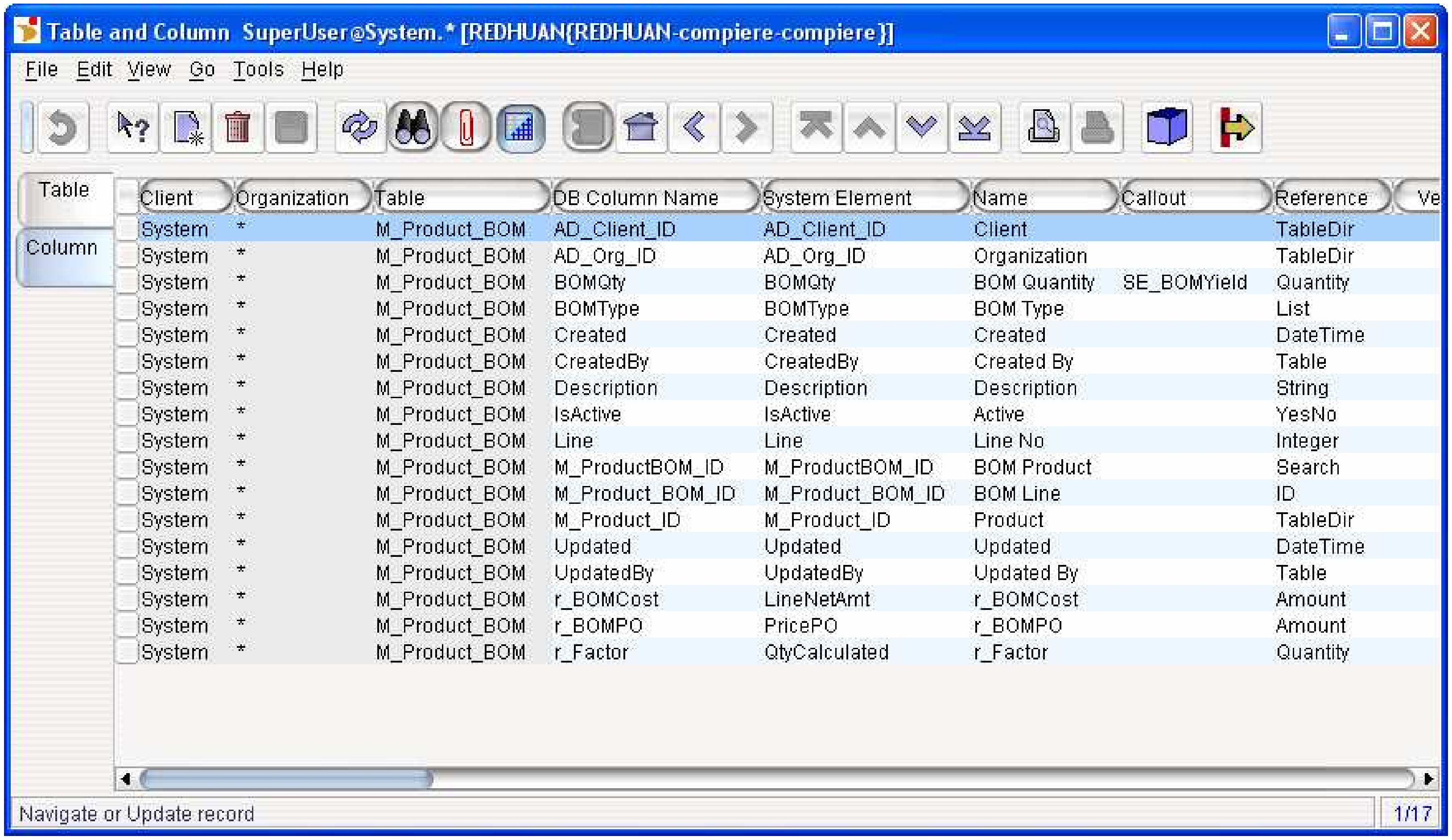
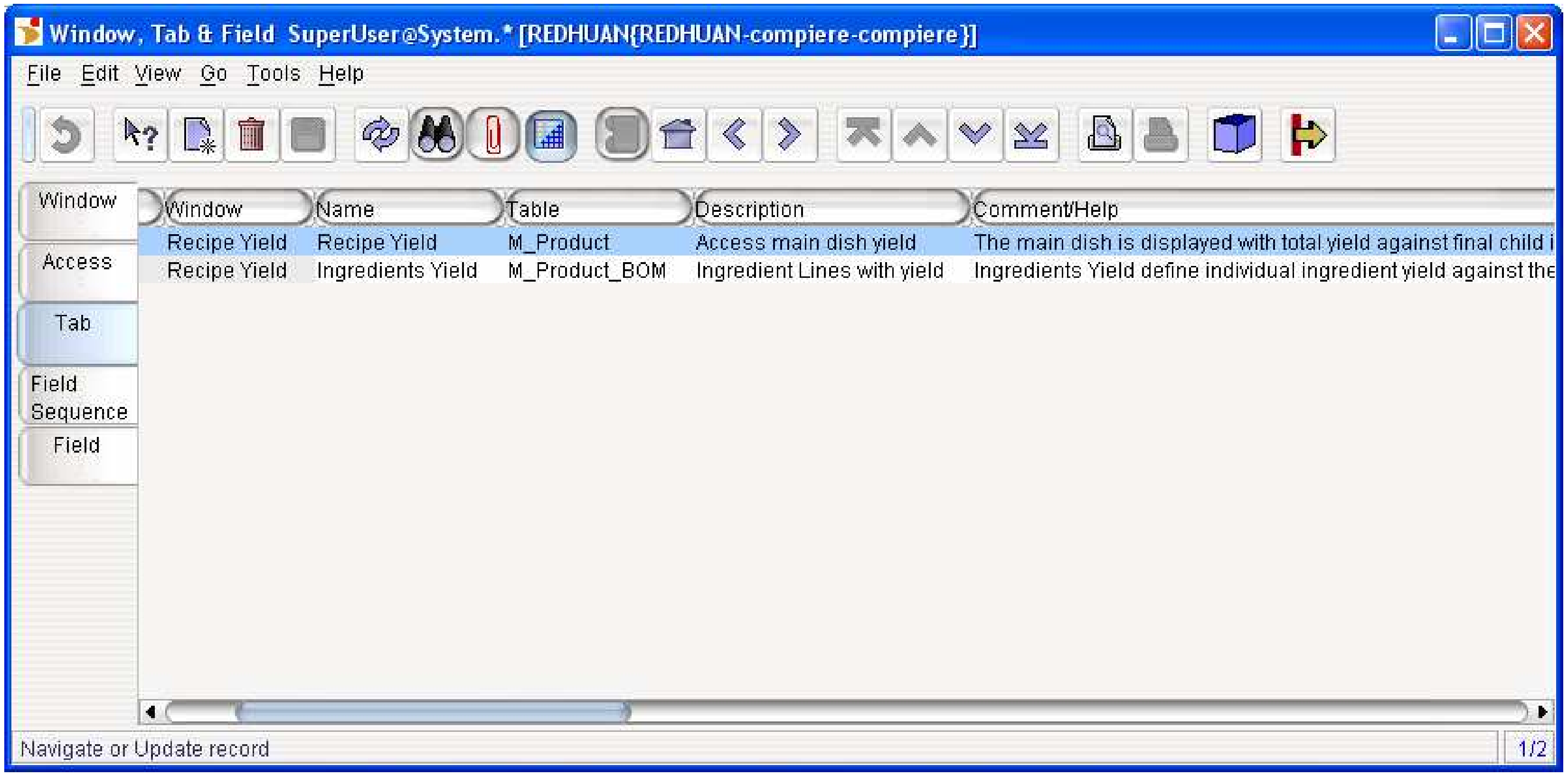


Table and Column SuperUser@System.* [REDHUAN{REDHUAN-compiere-compiere}]
File Edit View Go Tools Help
Table
Column
Navigate or Update record
1/17

Client	Organization	Table	DB Column Name	System Element	Name	Callout	Reference
System	*	M_Product_BOM	AD_Client_ID	AD_Client_ID	Client		TableDir
System	*	M_Product_BOM	AD_Org_ID	AD_Org_ID	Organization		TableDir
System	*	M_Product_BOM	BOMQty	BOMQty	BOM Quantity	SE_BOMYield	Quantity
System	*	M_Product_BOM	BOMType	BOMType	BOM Type		List
System	*	M_Product_BOM	Created	Created	Created		DateTime
System	*	M_Product_BOM	CreatedBy	CreatedBy	Created By		Table
System	*	M_Product_BOM	Description	Description	Description		String
System	*	M_Product_BOM	IsActive	IsActive	Active		YesNo
System	*	M_Product_BOM	Line	Line	Line No		Integer
System	*	M_Product_BOM	M_ProductBOM_ID	M_ProductBOM_ID	BOM Product		Search
System	*	M_Product_BOM	M_Product_BOM_ID	M_Product_BOM_ID	BOM Line		ID
System	*	M_Product_BOM	M_Product_ID	M_Product_ID	Product		TableDir
System	*	M_Product_BOM	Updated	Updated	Updated		DateTime
System	*	M_Product_BOM	UpdatedBy	UpdatedBy	Updated By		Table
System	*	M_Product_BOM	r_BOMCost	LineNetAmt	r_BOMCost		Amount
System	*	M_Product_BOM	r_BOMPO	PricePO	r_BOMPO		Amount
System	*	M_Product_BOM	r_Factor	QtyCalculated	r_Factor		Quantity

Figure 3: Column definitions of M_Product_BOM Table

Now we go to Window-Tab-Column to duplicate the Product Window but rename it accordingly. We will also throw out the other uneeded Tabs. Here is a screenshot of the Window details.


Window, Tab & Field SuperUser@System.* [REDHUAN{REDHUAN-compiere-compiere}]
File Edit View Go Tools Help
Window
Access
Tab
Field Sequence
Field

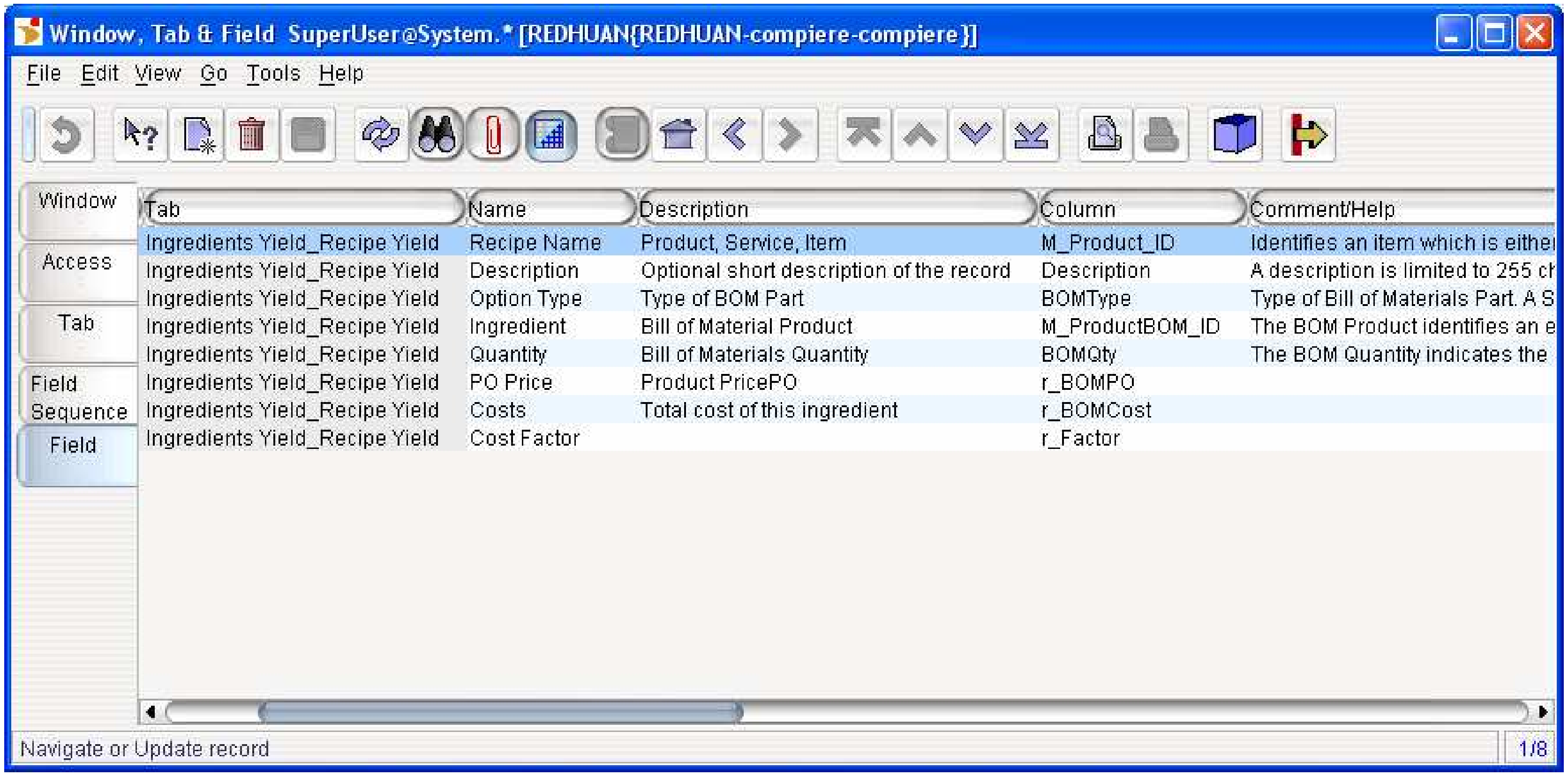
Window	Name	Table	Description	Comment/Help
Recipe Yield	Recipe Yield	M_Product	Access main dish yield	The main dish is displayed with total yield against final child i
Recipe Yield	Ingredients Yield	M_Product_BOM	Ingredient Lines with yield	Ingredients Yield define individual ingredient yield against the

Navigate or Update record 1/2

Figure 4: Window, Tab & Field definition for Recipe Yield

Remember that this is copied from the Product Window. You also copy the Tabs along with it. IN doing so, you save a lot of hassle. Remove the other Tabs, and rename accordingly as presented above.

Here is the second tab screen – Ingredients Yield, showing its fields that you have to include from the M_ProductBOM table.
In all, besides renaming the Window and Tab labels, you can also put in Description and Help/Comment details which will provide information to the user when she clicks on the Help Screen!


Window, Tab & Field SuperUser@System.* [REDHUAN{REDHUAN-compiere-compiere}]
File Edit View Go Tools Help
Window
Access
Tab
Field Sequence
Field
Navigate or Update record
1/8

Tab	Name	Description	Column	Comment/Help
Ingredients Yield_Recipe Yield	Recipe Name	Product, Service, Item	M_Product_ID	Identifies an item which is either
Ingredients Yield_Recipe Yield	Description	Optional short description of the record	Description	A description is limited to 255 ch
Ingredients Yield_Recipe Yield	Option Type	Type of BOM Part	BOMType	Type of Bill of Materials Part. A S
Ingredients Yield_Recipe Yield	Ingredient	Bill of Material Product	M_ProductBOM_ID	The BOM Product identifies an e
Ingredients Yield_Recipe Yield	Quantity	Bill of Materials Quantity	BOMQty	The BOM Quantity indicates the
Ingredients Yield_Recipe Yield	PO Price	Product PricePO	r_BOMPO	
Ingredients Yield_Recipe Yield	Costs	Total cost of this ingredient	r_BOMCost	
Ingredients Yield_Recipe Yield	Cost Factor		r_Factor	

Figure 5: Field definitions of the Ingredients Yield (BOM) Tab

To incorporate the 3 new fields – r_BOMPO, r_BOMCost and r_Factor, you create and select from the Column pulldown list which will retreive them from the M_Product_BOM new Table & Column’s definitions, that we have put in earlier.
The last thing left is to create a new Menu that links to this Window.

For the Callout to work, you have to recompile the source with the method BOMYield included in your CalloutSystem.java. Take this http://compiere.red1.org/YieldCallout.zip and replace in your source first.

Below is the method source reproduced and indexed. Red comments are included to explain further. A more wholesome commentary will close this document.

```
      /**
      * red1 BOMYield for Recipe Yield window
             */  The BOMYield method is placed in CalloutSystem.java for convenience and avoid fuss
1           public String BOMYield (Properties ctx, int WindowNo, MTab mTab, MField mField, Object value)
2           {
3                 if (isCalloutActive() || value == null)
4                       return "";
5                 //    init variables and get values
6                 BigDecimal r_BOMQty = (BigDecimal)value;
7                 BigDecimal r_BOMCost = Env.ZERO;
8                 BigDecimal r_factor = Env.ZERO;
9                 BigDecimal CostPrice = Env.ZERO;
10                BigDecimal TotalCost = Env.ZERO;
11                BigDecimal TotalQty = Env.ZERO;
12                BigDecimal r_Yield = new BigDecimal ("1");
13                Integer ii = (Integer)mTab.getValue("M_ProductBOM_ID"); Child record-id
14                if (ii == null)         turning it into an integer as record-ids are numerical
15                      return "";
16                int M_ProductBOM_ID = ii.intValue();  defining the variable to accept the record id
17                if (M_ProductBOM_ID == 0)
18                      return "";
19                Integer id = (Integer)mTab.getValue("M_Product_ID");
20                if (id == null)
21                      return "";
22                int M_Product_ID = id.intValue();   same thing with Parent record-id
23                if (M_Product_ID == 0)
24                      return "";
25 // red1 begin workings
26                setCalloutActive(true);
27                String sql = "SELECT pc.CurrentCostPrice "
28                      + "FROM M_Product_Costing pc "   pulling out the Cost Price for this child record
29                      + "WHERE pc.M_Product_ID=?";
```

```
        try
        {
            PreparedStatement pstmt = DB.prepareStatement(sql);
            pstmt.setInt(1, M_ProductBOM_ID);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next())
                CostPrice = rs.getBigDecimal(1);
            rs.close();
            pstmt.close();
        }
        catch (SQLException e)
        {
            log.error("red1 ERROR - CurrentCostPrice", e);
        }
// red1 CostPrice calculation
        r_BOMCost = CostPrice.multiply(r_BOMQty);
        log.debug("red1  -  BOMCost=" + r_BOMCost + " (CostPrice=" + CostPrice + "* BOMQty=" + r_BOMQty + ")");
        mTab.setValue("r_BOMPO", CostPrice);  putting values into their columns
        mTab.setValue("r_BOMCost", r_BOMCost);

//    red1 set yield factor = r_BOMCost / whole BOMCost of own parent's children, then * 100 to become percentage            TODO here is needed to avoid refreshing every child BOMQty to recalculate
        // red1*TODO but a change in current r_factor affects other siblings yield factor!
        //         maybe can be solved by new loop for all childs
        // red1** also take opportunity to get all BOMQtys to set M_Product.r_yield **
        String sql2 = "SELECT b.r_BOMCost, b.BOMQty "
            + "FROM M_Product_BOM b "
            + "WHERE b.M_Product_ID=?";   //    Getting all Children with same Parent
        try
        {
            PreparedStatement pstmt = DB.prepareStatement(sql2);
            pstmt.setInt(1, M_Product_ID);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) // red1*TODO have to loop thru array of other BOMCosts
            {   CostPrice = rs.getBigDecimal(1);
                TotalCost = TotalCost.add(CostPrice); Accumulating for denominators
                r_BOMQty = rs.getBigDecimal(2);
                TotalQty = TotalQty.add(r_BOMQty); //red1** getting the totalqty for r_Yield
            };
```

```
                        rs.close();
                        pstmt.close();
                }
                catch (SQLException e)
                {
                        log.error("red1 ERROR - TotalCostPrice", e);
                }
                if (TotalCost != Env.ZERO )
                        {     r_factor=r_BOMCost.divide(TotalCost,4,0);           //red1*TODO           other r_factors=*BOMCosts/TotalCost
                              r_factor=r_factor.multiply(new BigDecimal ("100")); //red1*
                        };
                mTab.setValue("r_factor", r_factor); //red1*TODO UPDATE siblings r_factors
                log.debug("red1 - (Yield Factor %)" + r_factor + "=(BOMCost)"+CostPrice+"/(TotalCost)"+TotalCost);
//    red1** calculate Yield for M_Product ( = 1 / child BOMQtys )
                r_Yield=r_Yield.divide(TotalQty,2,0);
                String sql3 = "UPDATE M_Product mp SET" Parent record called to set its Yield result.
                        + " r_Yield=" +r_Yield
                        + " WHERE M_Product_ID=" +
                        M_Product_ID;
                        int i = DB.executeUpdate(sql3.toString());
                        log.debug("red1 M_Product r_Yield - Updated if 1 => " + i+" Yield is "+r_Yield);

                setCalloutActive(false);
                return "";
        }       //    BOMYield
//    red1 end
```

Note: There will be a ‘divide by zero’ error when it is populating the children BOM records, but after the frist round, it will be okay. Just go through it ignoring that error, and let the program prove its concept. Later if this program proves useful you can then correct that error by placing an IF statement to filter out that erroenous situation.

MORE WHOLESOME COMMENTARY

Now I will explain the Callout adventure in such a graphic detail that governmental guidance is advised. ☺ . You may already now the basics of the Callout in the Callout.zip article I provided. Now, you may ask again - as a review, when does this Callout happen? Remember when we were at the Recipe Yield Screen. This Callout happens at the 2nd Tab, pulling the child record which is the BOM Record. It happens when the BOMQty value is changed or refreshed. This is when the cursor is placed inside the BOMQty field and any action is done to move the cursor out via Enter and Tab keys. Any mouse action without value change will not trigger the Callout, but an eventual Save action will. My favourite move is always the Tab key as it's the easiest keyboard action.

Why are we placing this Callout in CalloutSystem.java instead as a class of its own? Well, remembering that we are both not wizards in programming and also that the whole idea of a business app is to be productive, and that you have to roll out prototype and funcional extensions real fast, as your boss has no idea that a server has nothing to do with a waiter. She just wants you to justify your exhorbitant salary in terms of deliverables, so Compiere is like godsend as it is already off the shelf, and you just need to tweak it ever so in this way. By specifying the SE_BOMYield in the Callout field of the Table-Column definition, it will search for the mentioned method in CalloutSystem.java instead of other Callout javas. Only later when your Callout work that you can refactor it out, but that's for another day when we are somewhere in Bandung, Java Island highlands for a Alps-like equatorial vacation.

Notice that Line 26 and Line 85 set the Callout Active flag to true when the Callout is in session and then false when the sessoin ends. This is so that when you have another thread elsewhere winding along, you can set up checkpoints to notify that "Hey buddy! Are you blind? Can't you see the train passing by?". That checkpoint query is asked in Line 3.

These Active variables are protected by the superclass CalloutEngine.java which does the switch setting so as to keep track of all the Callouts or rather that only one Callout happens at a time and must end, before another begins. Now, that's a good station master – no crash is bound to happen here.

Line 13 is fetching the record id in session which is the M_ProductBOM_ID as we are in the second tab which pulls up the M_ProductBOM table. Note that this id is a M_Product_ID in the first place! But not to be confused with the present M_Product_ID which is the parent id. The child M_Product_ID (…BOM_ID) is then used to fetch its corresponding CurrentCostPrice from the M_ProductCosting table (set at line 33).

Line 19 is fetching the parent record id in session which is the M_Product_ID. This is then used to fetch the siblings records which has the same parent record (set in line 52).

Phew! It sure is challenging to raise a family huh? But SQL and Callout framework can achieve anything it seems. What contributes to the solution is also what is known as the Database's Entity Relationships. Compiere's database is high relational and normalised so that you can quite understand who is related to who. Now that you have got all your brothers and sisters belonging to the same BOM Parent, you can then group your common data together, such as CostPrice and BOMQty for the Yield. This is handled in lines 50 through 63. Notice that we use sql2 and so on as the sql statement storage as there are other statement variables and we do not want to be in conflict.

Notice the log.debug statements that litter many parts of the program. They are helpful to assist you in debugging the codes as they display the variables under fire. You can study these debug notes from the debug command line window when you launch Compiere2 with a –debug choice. Of course you have to compile the codes first. Otherwise when using your IDE, such as Eclipse, you view these debug prompts from the console display.

I like to raise a point on problem analysis and solution architecting. The original challenge is more than this. The prospect requires food nutrition info to be tied to each recipe, and that the yield must iterate through various levels. Before I arrive at a prototype definition, I probably spend hours going round in circles trying to understand the problem question and thinking of a solution. At first I thought I should set up a Table View that join the data string that I need. Then perhaps a Report & Process can pull that out and iterate under manipulation. But for this particular Proof of Concept it wasn't easy to go too far ahead into the problem. All I need was just to proof a limited functionality to give the prospect confidence that her needs are achievable and that if awarded the deal, we can live up to any expectations with the reasonablest of risk and costs.

You can also direct any queries or comments back to me at red1@red1.org. Thank you for reading me! Have a rich life!

red1 – 3.19 pm (GMT +8.00), 31st. of July, 2004.