# User Defined Functions

**by Jon Svede, Brian Bush**

### 1. How to Create and Use User Defined Functions

## 1.1. Description

This document describes the User Defined Functions within POI. User defined functions allow you to take code that is written in VBA and re-write in Java and use within POI. Consider the following example.

## 1.2. An Example

Suppose you are given a spreadsheet that can calculate the principal and interest payments for a mortgage. The user enters the principal loan amount, the interest rate and the term of the loan. The Excel spreadsheet does the rest.

mortgage calculation spreadsheet

When you actually look at the workbook you discover that rather than having the formula in a cell it has been written as VBA function. You review the function and determine that it could be written in Java:

VBA code

If we write a small program to try to evaluate this cell, we'll fail. Consider this source code:

```
import java.io.File ;
import java.io.FileInputStream ;
import java.io.FileNotFoundException ;
import java.io.IOException ;

import org.apache.poi.openxml4j.exceptions.InvalidFormatException ;
import org.apache.poi.ss.formula.functions.FreeRefFunction ;
import org.apache.poi.ss.formula.udf.AggregatingUDFFinder ;
import org.apache.poi.ss.formula.udf.DefaultUDFFinder ;
import org.apache.poi.ss.formula.udf.UDFFinder ;
import org.apache.poi.ss.usermodel.Cell ;
import org.apache.poi.ss.usermodel.CellValue ;
```

```
import org.apache.poi.ss.usermodel.Row ;
import org.apache.poi.ss.usermodel.Sheet ;
import org.apache.poi.ss.usermodel.Workbook ;
import org.apache.poi.ss.usermodel.WorkbookFactory ;
import org.apache.poi.ss.util.CellReference ;

public class Evaluator {



    public static void main( String[] args ) {

        System.out.println( "fileName: " + args[0] ) ;
        System.out.println( "cell: " + args[1] ) ;

        File workbookFile = new File( args[0] ) ;

        try {
            FileInputStream fis = new FileInputStream(workbookFile);
            Workbook workbook = WorkbookFactory.create(fis);

            FormulaEvaluator evaluator = workbook.getCreationHelper().createFormulaEval

            CellReference cr = new CellReference( args[1] ) ;
            String sheetName = cr.getSheetName() ;
            Sheet sheet = workbook.getSheet( sheetName ) ;
            int rowIdx = cr.getRow() ;
            int colIdx = cr.getCol() ;
            Row row = sheet.getRow( rowIdx ) ;
            Cell cell = row.getCell( colIdx ) ;

            CellValue value = evaluator.evaluate( cell ) ;

            System.out.println("returns value: " +  value ) ;


        } catch( FileNotFoundException e ) {
            e.printStackTrace();
        } catch( InvalidFormatException e ) {
            e.printStackTrace();
        } catch( IOException e ) {
            e.printStackTrace();
        }
    }
}
```

If you run this code, you're likely to get the following error:

```
Exception in thread "main" org.apache.poi.ss.formula.eval.NotImplementedException: Erro
    at org.apache.poi.ss.formula.WorkbookEvaluator.addExceptionInfo(WorkbookEvaluator.j
    at org.apache.poi.ss.formula.WorkbookEvaluator.evaluateAny(WorkbookEvaluator.java:2
    at org.apache.poi.ss.formula.WorkbookEvaluator.evaluate(WorkbookEvaluator.java:221)
    at org.apache.poi.hssf.usermodel.HSSFFormulaEvaluator.evaluateFormulaCellValue(HSSF
```

```
    at org.apache.poi.hssf.usermodel.HSSFFormulaEvaluator.evaluate(HSSFFormulaEvaluator
    at poi.tests.Evaluator.main(Evaluator.java:61)
Caused by: org.apache.poi.ss.formula.eval.NotImplementedException: calculatePayment
    at org.apache.poi.ss.formula.UserDefinedFunction.evaluate(UserDefinedFunction.java:
    at org.apache.poi.ss.formula.OperationEvaluatorFactory.evaluate(OperationEvaluatorF
    at org.apache.poi.ss.formula.WorkbookEvaluator.evaluateFormula(WorkbookEvaluator.ja
    at org.apache.poi.ss.formula.WorkbookEvaluator.evaluateAny(WorkbookEvaluator.java:2
    ... 4 more
```

How would we make it so POI can use this sheet?

## 1.3. Defining Your Function

To 'convert' this code to Java and make it available to POI you need to implement a FreeRefFunction instance. FreeRefFunction is an interface in the org.apache.poi.ss.formula.functions package. This interface defines one method, evaluate(ValueEval[] args, OperationEvaluationContext ec), which is how you will receive the argument values from POI.

The evaluate() method as defined above is where you will convert the ValueEval instances to the proper number types. The following code snippet shows you how to get your values:

```
public class CalculateMortgage implements FreeRefFunction {

@Override
public ValueEval evaluate( ValueEval[] args, OperationEvaluationContext ec ) {
    if (args.length != 3) {
        return ErrorEval.VALUE_INVALID;
    }

    double principal, rate, years,  result;
    try {
        ValueEval v1 = OperandResolver.getSingleValue( args[0],
                                                       ec.getRowIndex(),
                                                       ec.getColumnIndex() ) ;
        ValueEval v2 = OperandResolver.getSingleValue( args[1],
                                                       ec.getRowIndex(),
                                                       ec.getColumnIndex() ) ;
        ValueEval v3 = OperandResolver.getSingleValue( args[2],
                                                       ec.getRowIndex(),
                                                       ec.getColumnIndex() ) ;

        principal  = OperandResolver.coerceValueToDouble( v1 ) ;
        rate  = OperandResolver.coerceValueToDouble( v2 ) ;
        years = OperandResolver.coerceValueToDouble( v3 ) ;
```

The first thing we do is check the number of arguments being passed since there is no sense in attempting to go further if you are missing critical information.

Next we declare our variables, in our case we need variables for:

- principal - the amount of the loan
- rate - the interest rate as a decimal
- years - the length of the loan in years
- result - the result of the calculation

Next, we use the OperandResolver to convert the ValueEval instances to doubles, though not directly. First we start by getting discreet values. Using the OperandResolver.getSingleValue() method we retrieve each of the values passed in by the cell in the spreadsheet. Next, we use the OperandResolver again to convert the ValueEval instances to doubles, in this case. This class has other methods of coercion for gettings Strings, ints and booleans. Now that we've got our primitive values we can move on to calculating the value.

As shown previously, we have the VBA source. We need to add code to our class to calculate the payment. To do this you could simply add it to the method we've already created but I've chosen to add it as its own method. Add the following method:

```
public double calculateMortgagePayment( double p, double r, double y ) {

    double i = r / 12 ;
    double n = y * 12 ;

    double principalAndInterest =
         p * (( i * Math.pow((1 + i),n ) ) / ( Math.pow((1 + i),n) - 1))  ;

    return principalAndInterest ;
}
```

The biggest change necessary is related to the exponents; Java doesn't have a notation for this so we had to add calls to Math.pow(). Now we need to add this call to our previous method:

```
              result = calculateMortgagePayment( principal, rate, years ) ;
```

Having done that, the last things we need to do are to check to make sure we didn't get a bad result and, if not, we need to return the value. Add the following code to the class:

```
private void checkValue(double result) throws EvaluationException {
    if (Double.isNaN(result) || Double.isInfinite(result)) {
        throw new EvaluationException(ErrorEval.NUM_ERROR);
    }
}
```

Then add a line of code to our evaluate method to call this new static method, complete our try/catch and return the value:

```
        checkValue(result);

    } catch (EvaluationException e) {
        e.printStackTrace() ;
        return e.getErrorEval();
    }

    return new NumberEval( result ) ;
```

So the whole class would be as follows:

```
import org.apache.poi.ss.formula.OperationEvaluationContext ;
import org.apache.poi.ss.formula.eval.ErrorEval ;
import org.apache.poi.ss.formula.eval.EvaluationException ;
import org.apache.poi.ss.formula.eval.NumberEval ;
import org.apache.poi.ss.formula.eval.OperandResolver ;
import org.apache.poi.ss.formula.eval.ValueEval ;
import org.apache.poi.ss.formula.functions.FreeRefFunction ;

/**
 * A simple function to calculate principal and interest.
 *
 * @author Jon Svede
 *
 */
public class CalculateMortgage implements FreeRefFunction {

    @Override
    public ValueEval evaluate( ValueEval[] args, OperationEvaluationContext ec ) {
        if (args.length != 3) {
            return ErrorEval.VALUE_INVALID;
        }

        double principal, rate, years,  result;
        try {
            ValueEval v1 = OperandResolver.getSingleValue( args[0],
                                                           ec.getRowIndex(),
                                                           ec.getColumnIndex() ) ;
            ValueEval v2 = OperandResolver.getSingleValue( args[1],
                                                           ec.getRowIndex(),
                                                           ec.getColumnIndex() ) ;
            ValueEval v3 = OperandResolver.getSingleValue( args[2],
                                                           ec.getRowIndex(),
                                                           ec.getColumnIndex() ) ;

            principal  = OperandResolver.coerceValueToDouble( v1 ) ;
            rate  = OperandResolver.coerceValueToDouble( v2 ) ;
            years = OperandResolver.coerceValueToDouble( v3 ) ;

            result = calculateMortgagePayment( principal, rate, years ) ;

            checkValue(result);
```

```
        } catch (EvaluationException e) {
            e.printStackTrace() ;
            return e.getErrorEval();
        }

        return new NumberEval( result ) ;
    }

    public double calculateMortgagePayment( double p, double r, double y ) {
        double i = r / 12 ;
        double n = y * 12 ;

        //M = P [ i(1 + i)n ] / [ (1 + i)n - 1]
        double principalAndInterest =
             p * (( i * Math.pow((1 + i),n ) ) / ( Math.pow((1 + i),n) - 1))  ;

        return principalAndInterest ;
    }

    /**
     * Excel does not support infinities and NaNs, rather, it gives a #NUM! error in th
     *
     * @throws EvaluationException (#NUM!) if <tt>result</tt> is <tt>NaN</> or <tt>Infi
     */
     static final void checkValue(double result) throws EvaluationException {
         if (Double.isNaN(result) || Double.isInfinite(result)) {
             throw new EvaluationException(ErrorEval.NUM_ERROR);
         }
     }

}
```

Great! Now we need to go back to our original program that failed to evaluate our cell and add code that will allow it run our new Java code.

## 1.4. Registering Your Function

Now we need to register our function in the Workbook, so that the Formula Evaluator can resolve the name "calculatePayment" and map it to the actual implementation (CalculateMortgage). This is done using the UDFFinder object. The UDFFinder manages FreeRefFunctions which are our analogy for the VBA code. We need to create a UDFFinder. There are a few things we need to know in order to do this:

- The name of the function in the VBA code (in our case it is calculatePayment)
- The Class name of our FreeRefFunction

UDFFinder is actually an interface, so we need to use an actual implementation of this interface. Therefore we use the org.apache.poi.ss.formula.udf.DefaultUDFFinder class. If

you refer to the Javadocs you'll see that this class expects to get two arrays, one containing the alias and the other containing an instance of the class that will represent that alias. In our case our alias will be calculatePayment and our class instance will be of the CalculateMortgage type. This class needs to be available at compile and runtime. Be sure to keep these arrays well organized because you'll run into problems if these arrays are of different sizes or the alias aren't in the same relative position in their respective arrays. Add the following code:

```
String[] functionNames = { "calculatePayment" } ;
FreeRefFunction[] functionImpls = { new CalculateMortgage() } ;

UDFFinder udfs = new DefaultUDFFinder( functionNames, functionImpls ) ;
UDFFinder udfToolpack = new AggregatingUDFFinder( udfs ) ;
```

Now we have our UDFFinder instance and we've created the AggregatingUDFFinder instance. The last step is to pass this to our Workbook:

```
workbook.addToolPack(udfToolpack);
```

So now the whole class will look like this:

```
import java.io.File ;
import java.io.FileInputStream ;
import java.io.FileNotFoundException ;
import java.io.IOException ;

import org.apache.poi.openxml4j.exceptions.InvalidFormatException ;
import org.apache.poi.ss.formula.functions.FreeRefFunction ;
import org.apache.poi.ss.formula.udf.AggregatingUDFFinder ;
import org.apache.poi.ss.formula.udf.DefaultUDFFinder ;
import org.apache.poi.ss.formula.udf.UDFFinder ;
import org.apache.poi.ss.usermodel.Cell ;
import org.apache.poi.ss.usermodel.CellValue ;
import org.apache.poi.ss.usermodel.Row ;
import org.apache.poi.ss.usermodel.Sheet ;
import org.apache.poi.ss.usermodel.Workbook ;
import org.apache.poi.ss.usermodel.WorkbookFactory ;
import org.apache.poi.ss.util.CellReference ;

public class Evaluator {

    public static void main( String[] args ) {

        System.out.println( "fileName: " + args[0] ) ;
        System.out.println( "cell: " + args[1] ) ;

        File workbookFile = new File( args[0] ) ;
```

```
        try {
            FileInputStream fis = new FileInputStream(workbookFile);
            Workbook workbook = WorkbookFactory.create(fis);

            String[] functionNames = { "calculatePayment" } ;
            FreeRefFunction[] functionImpls = { new CalculateMortgage() } ;

            UDFFinder udfs = new DefaultUDFFinder( functionNames, functionImpls ) ;
            UDFFinder udfToolpack = new AggregatingUDFFinder( udfs ) ;

            workbook.addToolPack(udfToolpack);

            FormulaEvaluator evaluator = workbook.getCreationHelper().createFormulaEval

            CellReference cr = new CellReference( args[1] ) ;
            String sheetName = cr.getSheetName() ;
            Sheet sheet = workbook.getSheet( sheetName ) ;
            int rowIdx = cr.getRow() ;
            int colIdx = cr.getCol() ;
            Row row = sheet.getRow( rowIdx ) ;
            Cell cell = row.getCell( colIdx ) ;

            CellValue value = evaluator.evaluate( cell ) ;

            System.out.println("returns value: " +  value ) ;


        } catch( FileNotFoundException e ) {
            e.printStackTrace();
        } catch( InvalidFormatException e ) {
            e.printStackTrace();
        } catch( IOException e ) {
            e.printStackTrace();
        }
    }
}
```

Now that our evaluator is aware of the UDFFinder which in turn is aware of our FreeRefFunction, we're ready to re-run our example:

```
Evaluator mortgage-calculation.xls Sheet1!B4
```

which prints the following output in the console:

```
fileName: mortgage-calculation.xls
cell: Sheet1!B4
returns value: org.apache.poi.ss.usermodel.CellValue [790.7936267415464]
```

That is it! Now you can create Java code and register it, allowing your POI based appliction to run spreadsheets that previously were inaccessible.

This example can be found in the src/examples/src/org/apache/poi/ss/examples/formula folder in the source.