Developing Formula Evaluation

by Amol Deshmukh, Yegor Kozlov

1. Introduction

This document is for developers wishing to contribute to the FormulaEvaluator API functionality.

When evaluating workbooks you may encounter a org.apache.poi.ss.formula.eval.NotImplementedException which indicates that a function is not (yet) supported by POI. Is there a workaround? Yes, the POI framework makes it easy to add implementation of new functions. Prior to POI-3.8 you had to checkout the source code from svn and make a custom build with your function implementation. Since POI-3.8 you can register new functions in run-time.

Currently, contribution is desired for implementing the standard MS excel functions. Place holder classes for these have been created, contributors only need to insert implementation for the individual "evaluate()" methods that do the actual evaluation.

2. Overview of FormulaEvaluator

Briefly, a formula string (along with the sheet and workbook that form the context in which the formula is evaluated) is first parsed into RPN tokens using the FormulaParser class . (If you dont know what RPN tokens are, now is a good time to read this.)

2.1. The big picture

RPN tokens are mapped to Eval classes. (Class hierarchy for the Evals is best understood if you view the class diagram in a class diagram viewer.) Depending on the type of RPN token (also called as Ptgs henceforth since that is what the FormulaParser calls the classes) a specific type of Eval wrapper is constructed to wrap the RPN token and is pushed on the stack.... UNLESS the Ptg is an OperationPtg. If it is an OperationPtg, an OperationEval instance is created for the specific type of OperationPtg. And depending on how many operands it takes, that many Evals are popped of the stack and passed in an array to the OperationEval instance's evaluate method which returns an Eval of subtype ValueEval.Thus an operation in the formula is evaluated.

Note:

An Eval is of subinterface ValueEval or OperationEval. Operands are always ValueEvals, Operations are always OperationEvals.

`OperationEval.evaluate(Eval[])` returns an Eval which is supposed to be of type ValueEval (actually since ValueEval is an interface, the return value is instance of one of the implementations of ValueEval). The valueEval resulting from evaluate() is pushed on the stack and the next RPN token is evaluated.... this continues till eventually there are no more RPN tokens at which point, if the formula string was correctly parsed, there should be just one Eval on the stack - which contains the result of evaluating the formula.

Of course I glossed over the details of how AreaPtg and ReferencePtg are handled a little differently, but the code should be self explanatory for that. Very briefly, the cells included in AreaPtg and RefPtg are examined and their values are populated in individual ValueEval objects which are set into the AreaEval and RefEval (ok, since AreaEval and RefEval are interfaces, the implementations of AreaEval and RefEval - but you'll figure all that out from the code)

OperationEvals for the standard operators have been implemented and tested.

3. What functions are supported?

As of Feb 2012, POI supports about 140 built-in functions, see Appendix A for the full list. You can programmatically list supported / unsuported functions using trhe following helper methods:

```
    // list of functions that POI can evaluate
    Collection<String> suportedFuncs = WorkbookEvaluator.getSupportedFunctionNames();

    // list of functions that are not supported by POI
    Collection<String> unsupportedFuncs = WorkbookEvaluator.getNotSupportedFunctionName
```

4. Two base interfaces to start your implementation

All Excel formula function classes implement either org.apache.poi.hssf.record.formula.functions.Function or org.apache.poi.hssf.record.formula.functions.FreeRefFunction interface. Function is a commonn interface for the functions defined in the binary Excel format (BIFF8): these are "classic" Excel functions like SUM, COUNT, LOOKUP, etc. FreeRefFunction is a common interface for the functions from the Excel Analysis Toolpack and for User-Defined Functions. In the future these two interfaces are expected be unified into one, but for now

you have to start your implementation from two slightly different roots.

5. Which interface to start from?

You are about to implement a function XXX and don't know which interface to start from: Function or FreeRefFunction. Use the following code to check whether your function is from the excel Analysis Toolpack:

```
if(AnalysisToolPack.isATPFunction(functionName)){
    // the function implements org.apache.poi.hssf.record.formula.functions.Functio
} else {
    // the function implements org.apache.poi.hssf.record.formula.functions.FreeRef
}
```

6. Walkthrough of an "evaluate()" implementation.

Here is the fun part: lets walk through the implementation of the excel function **SQRT()**

AnalysisToolPack.isATPFunction("SQRTPI") returns false so the base interface is Function. There are sub-interfaces that make life easier when implementing numeric functions or functions with fixed number of arguments, 1-arg, 2-arg and 3-arg function:

- org.apache.poi.hssf.record.formula.functions.NumericFunction
- org.apache.poi.hssf.record.formula.functions.Fixed1ArgFunction
- org.apache.poi.hssf.record.formula.functions.Fixed2ArgFunction
- org.apache.poi.hssf.record.formula.functions.Fixed3ArgFunction
- org.apache.poi.hssf.record.formula.functions.Fixed4ArgFunction

Since SQRTPI takes exactly one argument we start our implementation from org.apache.poi.hssf.record.formula.functions.Fixed1ArgFunction:

```
Function SQRTPI = new Fixed1ArgFunction() {
    public ValueEval evaluate(int srcRowIndex, int srcColumnIndex, ValueEval arg0)
        try {
            // Retrieves a single value from a variety of different argument types
            // Excel rules.  Does not perform any type conversion.
            ValueEval ve = OperandResolver.getSingleValue(arg0, srcRowIndex, srcCol

            // Applies some conversion rules if the supplied value is not already a
            // Throws EvaluationException(#VALUE!) if the supplied parameter is not
            double arg = OperandResolver.coerceValueToDouble(ve);

            // this where all the heavy-lifting happens
            double result = Math.sqrt(arg*Math.PI);

            // Excel uses the error code #NUM! instead of IEEE NaN and Infinity,
            // so when a numeric function evaluates to Double.NaN or Double.Infinit
```

```
                // be sure to translate the result to the appropriate error code
                if (Double.isNaN(result) || Double.isInfinite(result)) {
                    throw new EvaluationException(ErrorEval.NUM_ERROR);
                }

                return new NumberEval(result);
            } catch (EvaluationException e){
                return e.getErrorEval();
            }
        }
    }
```

Now when the implementation is ready we need to register it in the formula evaluator:

```
    WorkbookEvaluator.registerFunction("SQRTPI", SQRTPI);
```

Voila! The formula evaluator now recognizes SQRTPI!

7. Floating-point Arithmetic in Excel

Excel uses the IEEE Standard for Double Precision Floating Point numbers except two cases where it does not adhere to IEEE 754:

1. Positive/Negative Infinities: Infinities occur when you divide by 0. Excel does not support infinities, rather, it gives a #DIV/0! error in these cases.
2. Not-a-Number (NaN): NaN is used to represent invalid operations (such as infinity/infinity, infinity-infinity, or the square root of -1). NaNs allow a program to continue past an invalid operation. Excel instead immediately generates an error such as #NUM! or #DIV/0!.

Be aware of these two cases when saving results of your scientific calculations in Excel: “where are my Infinities and NaNs? They are gone!”

8. Testing Framework

Automated testing of the implemented Function is easy. The source code for this is in the file: o.a.p.h.record.formula.GenericFormulaTestCase.java This class has a reference to the test xls file (not /a/ test xls, /the/ test xls :) which may need to be changed for your environment. Once you do that, in the test xls, locate the entry for the function that you have implemented and enter different tests in a cell in the FORMULA row. Then copy the "value of" the formula that you entered in the cell just below it (this is easily done in excel as: [copy the formula cell] > [go to cell below] > Edit > Paste Special > Values > "ok"). You can enter multiple such formulas and paste their values in the cell below and the test framework will automatically test if the formula evaluation matches the expected value (Again, hard to put in words, so if you will, please take time to quickly look at the code and the currently entered

tests in the patch attachment "FormulaEvalTestData.xls" file).

9. Appendix A

Functions supported by POI (as of Feb 2012)

```
ABS
ACOS
ACOSH
ADDRESS
AND
ASIN
ASINH
ATAN
ATAN2
ATANH
AVEDEV
AVERAGE
CEILING
CHAR
CHOOSE
CLEAN
COLUMN
COLUMNS
COMBIN
CONCATENATE
COS
COSH
COUNT
COUNTA
COUNTBLANK
COUNTIF
DATE
DAY
DAYS360
DEGREES
DEVSQ
DOLLAR
ERROR.TYPE
EVEN
EXACT
EXP
FACT
FALSE
FIND
FLOOR
FV
HLOOKUP
HOUR
HYPERLINK
IF
INDEX
```

```
    INDIRECT
    INT
    IRR
    ISBLANK
    ISERROR
    ISEVEN
    ISLOGICAL
    ISNA
    ISNONTEXT
    ISNUMBER
    ISODD
    ISREF
    ISTEXT
    LARGE
    LEFT
    LEN
    LN
    LOG
    LOG10
    LOOKUP
    LOWER
    MATCH
    MAX
    MAXA
    MEDIAN
    MID
    MIN
    MINA
    MINUTE
    MOD
    MODE
    MONTH
    MROUND
    NA
    NETWORKDAYS
    NOT
    NOW
    NPER
    NPV
    ODD
    OFFSET
    OR
    PI
    PMT
    POISSON
    POWER
    PRODUCT
    PV
    RADIANS
    RAND
    RANDBETWEEN
    RANK
    RATE
    REPLACE
```

```
    RIGHT
    ROUND
    ROUNDDOWN
    ROUNDUP
    ROW
    ROWS
    SEARCH
    SECOND
    SIGN
    SIN
    SINH
    SMALL
    SQRT
    STDEV
    SUBSTITUTE
    SUBTOTAL
    SUM
    SUMIF
    SUMIFS
    SUMPRODUCT
    SUMSQ
    SUMX2MY2
    SUMX2PY2
    SUMXMY2
    T
    TAN
    TANH
    TEXT
    TIME
    TODAY
    TRIM
    TRUE
    TRUNC
    UPPER
    VALUE
    VAR
    VARP
    VLOOKUP
    WORKDAY
    YEAR
    YEARFRAC
```