ExcelAnt - Ant Tasks for Validating Excel Spreadsheets

by Jon Svede, Brian Bush

1. ExcelAnt - Ant Tasks for Validating Excel Spreadsheets

1.1. Introduction

ExcelAnt is a set of Ant tasks that make it possible to verify or test a workbook without having to write Java code. Of course, the tasks themselves are written in Java, but to use this frame work you only need to know a little bit about Ant.

This document covers the basic usage and set up of ExcelAnt.

This document will assume basic familiarity with Ant and Ant build files.

1.2. Setup

To start with, you'll need to have the POI 3.8 or higher jar files. If you test only .xls workbooks then you need to have the following jars in your path:

- poi-excelant-$version-YYYYDDMM.jar
- poi-$version-YYYYDDMM.jar
- poi-ooxml-$version-YYYYDDMM.jar

If you evaluate .xlsx workbooks then you need to add these:

- poi-ooxml-schemas-$version-YYYYDDMM.jar
- xmlbeans.jar
- dom4j.jar

For example, if you have these jars in a lib/ dir in your project, your build.xml might look like this:

```
<property name="lib.dir" value="lib" />

<path id="excelant.path">
    <pathelement location="${lib.dir}/poi-excelant-3.8-beta1-20101230.jar" />
    <pathelement location="${lib.dir}/poi-3.8-beta1-20101230.jar" />
    <pathelement location="${lib.dir}/poi-ooxml-3.8-beta1-20101230.jar" />
</path>
```

Next, you'll need to define the Ant tasks. There are several ways to use ExcelAnt:

- The traditional way:

```

    <typedef resource="org/apache/poi/ss/excelant/antlib.xml" classpathref="excelant.pa
```

Where excelant.path referes to the classpath with POI jars. Using this approach the provided extensions will live in the default namespace. Note that the default task/typenames (evaluate, test) may be too generic and should either be explicitly overridden or used with a namespace.

- Similar, but assigning a namespace URI:

```

<project name="excelant-demo"  xmlns:poi="antlib:org.apache.poi.ss.excelant">

    <typedef resource="org/apache/poi/ss/excelant/antlib.xml"
             classpathref="excelant.classpath"
             uri="antlib:org.apache.poi.ss.excelant"/>

    <target name="test-nofile">
        <poi:excelant>

        </poi:excelant>
    </target>
</project>
```

1.3. A Simple Example

The simplest example of using Excel is the ability to validate that POI is giving you back the value you expect it to. Does this mean that POI is inaccurate? Hardly. There are cases where POI is unable to evaluate cells for a variety of reasons. If you need to write code to integrate a worksheet into an app, you may want to know that it's going to work before you actually try to write that code. ExcelAnt helps with that.

Consider the mortgage-calculation.xls file found in the Examples (/examples/src/org/apache/poi/ss/examples/excelant/simple-mortgage-calculation.xls). This sheet is shown below:

This sheet calculates the principal and interest payment for a mortgage based on the amount of the loan, term and rate. To write a simple ExcelAnt test you need to tell ExcelAnt about the file like this:

```

<property name="xls.file" value="" />

<target name="simpleTest">
    <excelant fileName="${xls.file}">
        <test name="checkValue" showFailureDetail="true">
```

```
            <evaluate showDelta="true" cell="'MortgageCalculator'!$B$4" expectedValue="
        </test>
    </excelant>
</target>
```

This code sets up ExcelAnt to access the file defined in the ant property xls.file. Then it creates a 'test' named 'checkValue'. Finally it tries to evaluate the B4 on the sheet named 'MortgageCalculator'. There are some assumptions here that are worth explaining. For starters, ExcelAnt is focused on the testing numerically oriented sheets. The <evaluate> task is actually evaluating the cell as a formula using a FormulaEvaluator instance from POI. Therefore it will fail if you point it to a cell that doesn't contain a formula or a test a plain old number.

Having said all that, here is what the output looks like:

```

simpleTest:
 [excelant] ExcelAnt version 0.4.0 Copyright 2011
 [excelant] Using input file: resources/excelant.xls
 [excelant] 1/1 tests passed.
BUILD SUCCESSFUL
Total time: 391 milliseconds
```

1.4. Setting Values into a Cell

So now we know that at a minimum POI can use our sheet to calculate the existing value. This is an important point: in many cases sheets have dependencies, i.e., cells they reference. As is often the case, these cells may have dependencies, which may have dependencies, etc. The point is that sometimes a dependent cell may get adjusted by a macro or a function and it may be that POI doesn't have the capabilities to do the same thing. This test verifies that we can rely on POI to retrieve the default value, based on the stored values of the sheet. Now we want to know if we can manipulate those dependencies and verify the output.

To verify that we can manipulate cell values, we need a way in ExcelAnt to set a value. This is provided by the following task types:

- setDouble() - sets the specified cell as a double.
- setFormula() - sets the specified cell as a formula.
- setString() = sets the specified cell as a String.

For the purposes of this example we'll use the <setDouble> task. Let's start with a $240,000, 30 year loan at 11% (let's pretend it's like 1984). Here is how we will set that up:

```

<setDouble cell="'MortgageCalculator'!$B$1" value="240000"/>
<setDouble cell="'MortgageCalculator'!$B$2" value ="0.11"/>
<setDouble cell="'MortgageCalculator'!$B$3" value ="30"/>
<evaluate showDelta="true" cell="'MortgageCalculator'!$B$4" expectedValue="2285.576149"
```

Don't forget that we're verifying the behavior so you need to put all this into the sheet. That is how I got the result of $2,285 and change. So save your changes and run it; you should get the following:

```
Buildfile: C:\opt\eclipse\workspaces\excelant\excelant.examples\build.xml
simpleTest:
 [excelant] ExcelAnt version 0.4.0 Copyright 2011
 [excelant] Using input file: resources/excelant.xls
 [excelant] 1/1 tests passed.
BUILD SUCCESSFUL
Total time: 406 milliseconds
```

1.5. Getting More Details

This is great, it's working! However, suppose you want to see a little more detail. The ExcelAnt tasks leverage the Ant logging so you can add the -verbose and -debug flags to the Ant command line to get more detail. Try adding -verbose. Here is what you should see:

```
simpleTest:
 [excelant] ExcelAnt version 0.4.0 Copyright 2011
 [excelant] Using input file: resources/excelant.xls
 [evaluate] test precision = 1.0E-4 global precision = 0.0
 [evaluate] Using evaluate precision of 1.0E-4
 [excelant] 1/1 tests passed.
BUILD SUCCESSFUL
Total time: 406 milliseconds
```

We see a little more detail. Notice that we see that there is a setting for global precision. Up until now we've been setting the precision on each evaluate that we call. This is obviously useful but it gets cumbersome. It would be better if there were a way that we could specify a global precision - and there is. There is a <precision> tag that you can specify as a child of the <excelant> tag. Let's go back to our original task we set up earlier and modify it:

```
<property name="xls.file" value="" />

<target name="simpleTest">
    <excelant fileName="${xls.file}">
        <precision value="1.0e-3"/>
        <test name="checkValue" showFailureDetail="true">
            <evaluate showDelta="true" cell="'MortgageCalculator'!$B$4" expectedValue="
        </test>
    </excelant>
</target>
```

In this example we have set the global precision to 1.0e-3. This means that in the absence of something more stringent, all tests in the task will use the global precision. We can still override this by specifying the precision attribute of all of our <evaluate> task. Let's first run this task with the global precision and the -verbose flag:

```
simpleTest:
[excelant] ExcelAnt version 0.4.0 Copyright 2011
[excelant] Using input file: resources/excelant.xls
[excelant] setting precision for the test checkValue
    [test] setting globalPrecision to 0.0010 in the evaluator
[evaluate] test precision = 0.0  global precision = 0.0010
[evaluate] Using global precision of 0.0010
[excelant] 1/1 tests passed.
```

As the output clearly shows, the test itself has no precision but there is the global precision. Additionally, it tells us we're going to use that more stringent global value. Now suppose that for this test we want to use a more stringent precision, say 1.0e-4. We can do that by adding the precision attribute back to the <evaluate> task:

```

<excelant fileName="${xls.file}">
    <precision value="1.0e-3"/>
    <test name="checkValue" showFailureDetail="true">
        <setDouble cell="'MortgageCalculator'!$B$1" value="240000"/>
        <setDouble cell="'MortgageCalculator'!$B$2" value ="0.11"/>
        <setDouble cell="'MortgageCalculator'!$B$3" value ="30"/>
        <evaluate showDelta="true" cell="'MortgageCalculator'!$B$4" expectedValue="2285
    </test>
</excelant>
```

Now when you re-run this test with the verbose flag you will see that your test ran and passed with the higher precision:

```

simpleTest:
 [excelant] ExcelAnt version 0.4.0 Copyright 2011
 [excelant] Using input file: resources/excelant.xls
 [excelant] setting precision for the test checkValue
     [test] setting globalPrecision to 0.0010 in the evaluator
 [evaluate] test precision = 1.0E-4 global precision = 0.0010
 [evaluate] Using evaluate precision of 1.0E-4 over the global precision of 0.0010
 [excelant] 1/1 tests passed.
BUILD SUCCESSFUL
Total time: 390 milliseconds
```

1.6. Leveraging User Defined Functions

POI has an excellent feature (besides ExcelAnt) called User Defined Functions, that allows you to write Java code that will be used in place of custom VB code or macros is a spreadsheet. If you have read the documentation and written your own FreeRefFunction implmentations, ExcelAnt can make use of this code. For each <excelant> task you define you can nest a <udf> tag which allows you to specify the function alias and the class name.

Consider the previous example of the mortgage calculator. What if, instead of being a formula in a cell, it was a function defined in a VB macro? As luck would have it, we already have an example of this in the examples from the User Defined Functions example, so let's use that. In the example spreadsheet there is a tab for MortgageCalculatorFunction, which will use. If you look in cell B4, you see that rather than a messy cell based formula, there is only the function call. Let's not get bogged down in the function/Java implementation, as these are covered in the User Defined Function documentation. Let's just add a new target and test to our existing build file:

```
 <target name="functionTest">
     <excelant fileName="${xls.file}">
         <udf functionAlias="calculatePayment" class="org.apache.poi.ss.examples.formu
         <precision value="1.0e-3"/>
         <test name="checkValue" showFailureDetail="true">
             <setDouble cell="'MortgageCalculator'!$B$1" value="240000"/>
             <setDouble cell="'MortgageCalculator'!$B$2" value ="0.11"/>
             <setDouble cell="'MortgageCalculator'!$B$3" value ="30"/>
             <evaluate showDelta="true" cell="'MortgageCalculatorFunction'!$B$4" expec
         </test>
      </excelant>
   </target>
```

So if you look at this carefully it looks the same as the previous examples. We still use the global precision, we're still setting values, and we still want to evaluate a cell. The only real differences are the sheet name and the addition of the function.