Appendix B EUnit – test harness for Excel

B.1 Introduction

EUnit is a test harness to implement unit testing for Excel. EUnit is specific to Excel and is implemented as part of Excel, so it cannot be used with other spreadsheet packages. Therefore, unlike the rest of this guide, this appendix is only applicable to Excel and not to other spreadsheet packages.

A test harness is a piece of software that allows a selection of tests to be run, and the results recorded, without further interaction with the tester. Unit testing is a method of testing where each code unit (module, function, etc.) is tested, rather than testing the software as a whole. EUnit is design for unit testing of Excel applications, but can also used for system testing.

The implementation of unit testing in EUnit follows that of JUnit, for testing Java [10], including the choice of names for the testing procedures. JUnit uses Java exception handling to handle errors in tests, but it was not possible to mimic this in EUnit: instead global variables are used to record execution failures in tests.

Tests are written as procedures that ultimately call “assert” procedures that are used to indicate whether the tests pass or fail. The test procedures reside in code modules, which can also contain set-up and tear-down procedures that are run before and after each test in the module. Having written the tests, they are run using the EUnit GUI. The GUI allows you to select the tests, run the tests and display the results. The following sections describe how to write the tests and how to run them.

A beta release of EUnit is available from:
http://www.npl.co.uk/ssfm/download/documents/software/eunit/eunit-beta-1.3.zip

B.2 Writing the tests

The test procedures reside in code modules, these code module can contain other code and the test procedures are distinguished by having a name beginning with “`test`” (case does not matter). The code modules containing test can also include special procedures, “`setUp`” and “`tearDown`”, to be run before and after each test in the module. The modules can contain other functions and procedures, perhaps to be used to support the test procedures, as long as their names do not match the test procedures.

B.2.1 Test procedures

All test procedures should have the following structure:

```
Public Sub testNameOfTest()
    On Error GoTo traperrors
        ' The test code goes here
    Exit Sub
traperrors:
    Call fail("Failed executing the test",
              Err.Description,
              Err.Source,
              Err.Number)
End Sub
```

The trapping of errors within a test is vital; otherwise the test harness (as a whole) could fail.

This is a simple example test procedure

```
Public Sub testMultiply()
    On Error GoTo traperrors
        Range("A1").Value = 2
        Range("B1").Value = 3
        Range("C1").Formula = "=A1*B1"
        Call assertEquals(Range("C1").Value, _
                6, , "Multiply two numbers")
    Exit Sub
traperrors:
    Call fail("Failed executing the test", _
              Err.Description, _
              Err.Source, _
              Err.Number)
End Sub
```

Only test procedures, starting with “`test`”, will be executed as tests by the harness, all other procedures will not be executed (unless called by a test).

B.2.2 setUP and tearDown

Two other procedures can be part of a test module `setUP` and `tearDown`; these are executed respectively before and after each test in a module is executed. Their structure is similar to the test procedures and is shown below:

```
Public Sub setUP()
    On Error GoTo traperrors
        ' initialisation code goes here
    Exit Sub
traperrors:
    Call fail("Failed set up", _
              Err.Description, _
              Err.Source, _
              Err.Number)
End Sub
```

```
Public Sub tearDown()
    On Error GoTo traperrors
        ' termination code goes here
    Exit Sub
traperrors:
    Call fail("Failed tear down", _
              Err.Description, _
              Err.Source, _
              Err.Number)
End Sub
```

Here is an example of the use of `setUP` and `tearDown` to save and restore cells from the worksheet that are overwritten by the test (for example, `testMuliply` defined above).

```
Option Explicit
Dim a, b, c

Public Sub setUP()
    On Error GoTo traperrors
        a = Range("A1")
        b = Range("B1")
        c = Range("C1")
        Range("A1:C1").ClearContents
    Exit Sub
traperrors:
    Call fail("Failed set up", _
              Err.Description, _
              Err.Source, _
              Err.Number)
End Sub

Public Sub tearDown()
    On Error GoTo traperrors
        Range("A1:C1").ClearContents
        Range("A1") = a
        Range("B1") = b
        Range("C1") = c
    Exit Sub
traperrors:
    Call fail("Failed tear down", _
              Err.Description, _
              Err.Source, _
              Err.Number)
End Sub
```

B.2.3 Test results

Each test must have one (or more) of the several "assert methods" that assign the test result; these are described in the next sections. If a test has failed, this is recorded as the final result of that test, although the test may continue to execute (and if in an infinite loop will remain in it!).

A test can result in one of four results, which are explained in the table below:

Possible test result	**Meaning**
Pass	The test has passed.
Fail	The test has failed at some assert method.
Abort	No assertions have been made in the test.
Error	This should not occur – error in EUnit.

Table 4: Test results for EUnit tests

B.2.4 assertTrue

Test based on a Boolean expression.

Name	**assertTrue**			
Parameters	Exp	Boolean	M	Boolean expression which if true the assertion is then true
	TestDescription	String	O	Description of the test
	Description	String	O	Usually Err.Decription
	Source	String	O	Usually Err.Source
	Number	Long	O	Usually Err.Number

Examples:

```
Call assertTrue(True, _
                "Captured a deliberate error in the test", _
                 Err.Description, _
                 Err.Source, _
                 Err.Number)
Call assertTrue(Result, "setUP test 1")
```

B.2.5 assertEquals

Test if two values are equal.

Name	**assertEquals**			
Parameters	Expected	Variant	M	Expected result – Can be Double, Single, Integer, Long, Byte, Decimal or String
	Actual	Variant	M	Actual result – Can be Double, Single, Integer, Long, Byte, Decimal or String
	Delta	Double	O	Used when comparing Double or Single, has a default value of 4.94065645841247E-324
	TestDescription	String	O	Description of the test

Examples:

```
Call assertEquals(XY, _
                  Worksheets("testSheet1").Range("XY"), _
                  , _
                 "Test 1: integer")
Call assertEquals(5.75, _
                  Worksheets("testSheet1").Range("XY"), _
                  0.001, _
                  "Test 2: double/single")
Call assertEquals("Hello World", _
                  Worksheets("testSheet1").Range("XYs"), _
                  , _
                  "Test 4: strings")
```

B.2.6 fail

This is used when test have failed – it is equivalent to `assertTrue(False, …)`.

Name	**fail**			
Parameters	TestDescription	String	O	Description of the test
	Description	String	O	Usually Err.Decription
	Source	String	O	Usually Err.Source
	Number	Long	O	Usually Err.Number

Examples:

```
Call fail("Failed executing the test", _
          Err.Description, _
          Err.Source, _
          Err.Number)
Call fail("Fail to capture the error")
```

B.3 Running the tests

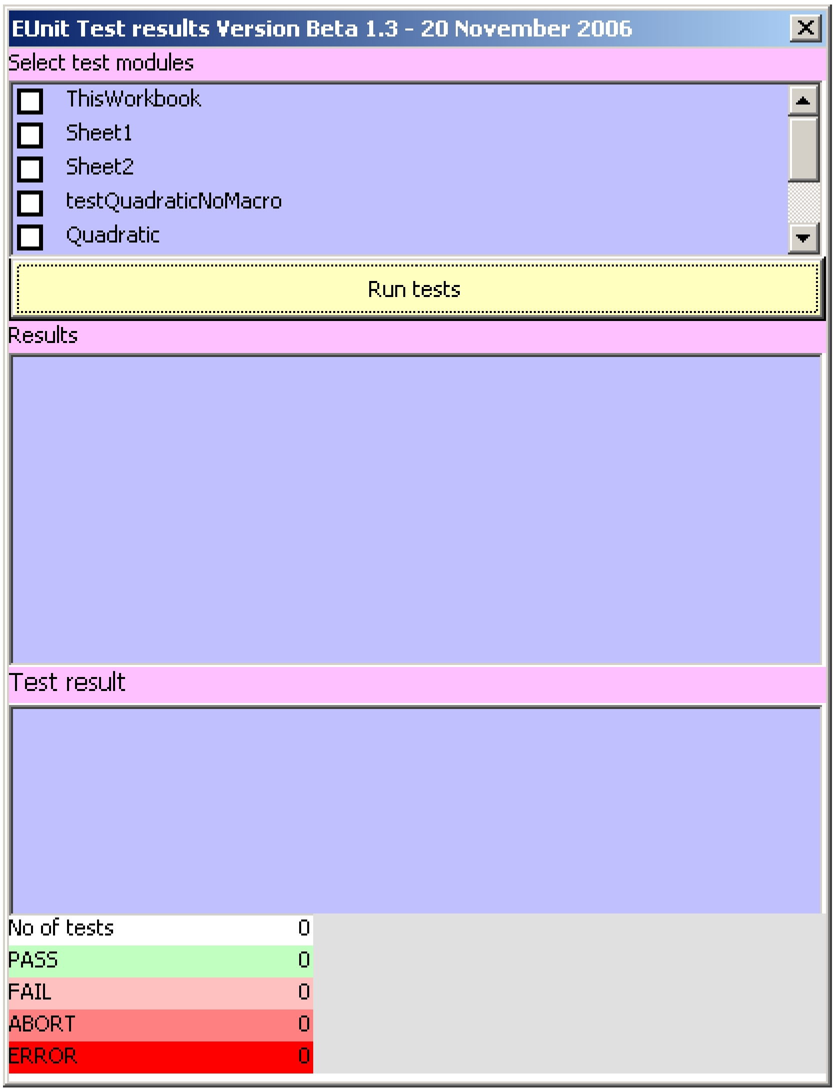
B.3.1 Installing EUnit

Before the tests can be run, EUnit must be attached to the spreadsheet application as a reference. The tests can then be executed through the supplied GUI.

From the Microsoft Visual Basic toolbar: Tools → References → Browse → EUnit.xls.

B.3.2 Using the GUI

The GUI is activated by calling the macro `EUnit.xls!showallresults`,
in the workbook with the tests in it, in which case you should see the following figure.



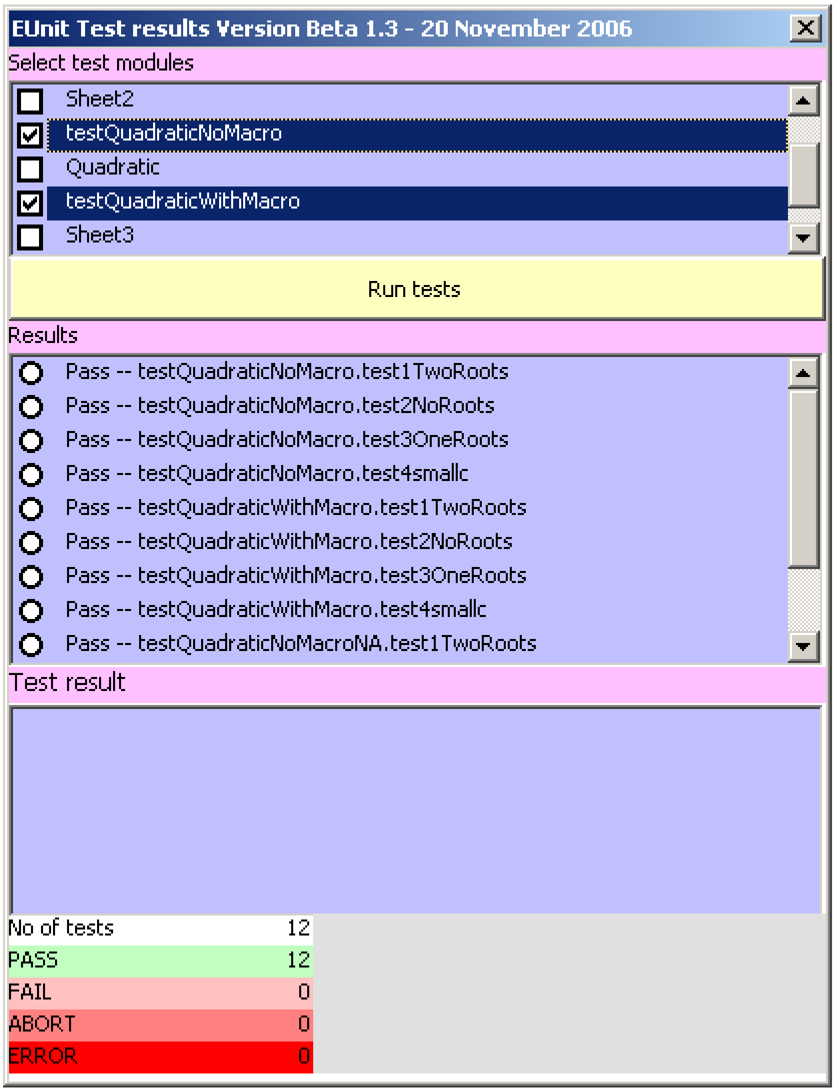
☐ ThisWorkbook
☐ Sheet1
☐ Sheet2
☐ testQuadraticNoMacro
☐ Quadratic

No of tests	0
PASS	0
FAIL	0
ABORT	0
ERROR	0

Figure 3: EUnit panel

B.3.3 Running tests

The top part of the GUI contains a list of all the possible code modules in the workbook. Select those with the tests and click on the yellow button. The results will appear, in the part of the GUI labelled Results, see below:



☐ Sheet2
☑ testQuadraticNoMacro
☐ Quadratic
☑ testQuadraticWithMacro
☐ Sheet3

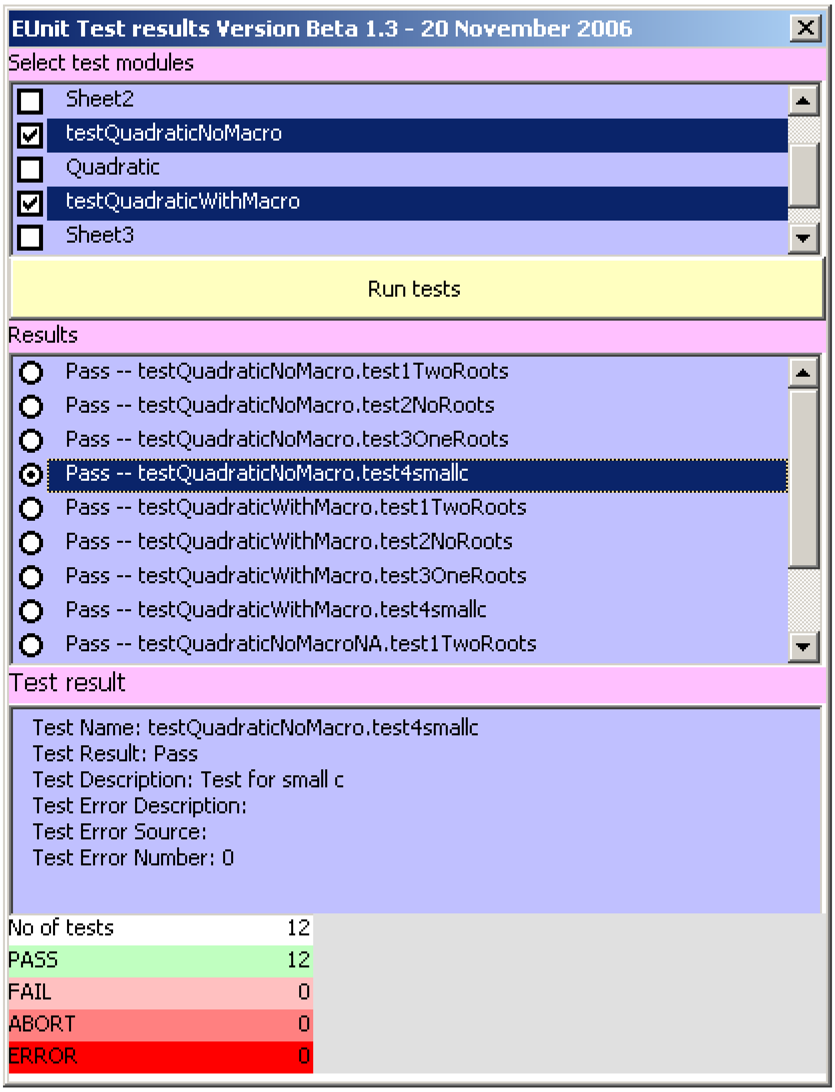
○ Pass -- testQuadraticNoMacro.test1TwoRoots
○ Pass -- testQuadraticNoMacro.test2NoRoots
○ Pass -- testQuadraticNoMacro.test3OneRoots
○ Pass -- testQuadraticNoMacro.test4smallc
○ Pass -- testQuadraticWithMacro.test1TwoRoots
○ Pass -- testQuadraticWithMacro.test2NoRoots
○ Pass -- testQuadraticWithMacro.test3OneRoots
○ Pass -- testQuadraticWithMacro.test4smallc
○ Pass -- testQuadraticNoMacroNA.test1TwoRoots

No of tests	12
PASS	12
FAIL	0
ABORT	0
ERROR	0

Figure 4: EUnit test results

B.3.4 Displaying details of tests

For any test further details can be obtained by clicking on it:



☐ Sheet2
☑ testQuadraticNoMacro
☐ Quadratic
☑ testQuadraticWithMacro
☐ Sheet3

○ Pass -- testQuadraticNoMacro.test1TwoRoots
○ Pass -- testQuadraticNoMacro.test2NoRoots
○ Pass -- testQuadraticNoMacro.test3OneRoots
● Pass -- testQuadraticNoMacro.test4smallc
○ Pass -- testQuadraticWithMacro.test1TwoRoots
○ Pass -- testQuadraticWithMacro.test2NoRoots
○ Pass -- testQuadraticWithMacro.test3OneRoots
○ Pass -- testQuadraticWithMacro.test4smallc
○ Pass -- testQuadraticNoMacroNA.test1TwoRoots

Test Name: testQuadraticNoMacro.test4smallc
Test Result: Pass
Test Description: Test for small c
Test Error Description:
Test Error Source:
Test Error Number: 0

No of tests	12
PASS	12
FAIL	0
ABORT	0
ERROR	0

Figure 5: EUnit test detail

References

These are books and articles we found useful and relevant. They cover the application of software engineering practice to spreadsheet application development and rapid application development.

1. Cook, H.R., M.G. Cox, M.P. Dainton, and P.M. Harris. *A methodology for testing spreadsheets and other packages used in metrology. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme.* NPL Report CISE 25/99, NPL, September 1999. http://www.npl.co.uk/ssfm/download/#cise25_99
2. Cook, H.R., M.G. Cox, M.P. Dainton, and P.M. Harris. *Testing spreadsheets and other packages used in metrology: A case study. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme.* NPL Report CISE 26/99, NPL, September 1999. http://www.npl.co.uk/ssfm/download/#cise26_99
3. Cook, H.R., M.G. Cox, M.P. Dainton, and P.M. Harris. *Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of Excel. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme.* NPL Report CISE 27/99, NPL, September 1999. http://www.npl.co.uk/ssfm/download/#cise27_99
4. Cox, M.G., M.P. Dainton, and P.M. Harris. *Testing functions for the calculation of standard deviation.* NPL Report CMSC 07/00, October 2000.
5. Cox, M.G., M.P. Dainton, and P.M. Harris. *Testing functions for linear regression.* NPL Report CMSC 08/00, October 2000.
6. Cox, M.G., P.M. Harris, E.G. Johnson, P.D. Kenward, and G.I. Parkin. *Testing the numerical correctness of software.* NPL Report CMSC 34/04, January 2004.
7. DACS, *Rapid Application Development (RAD).* Software Tech News, 1998. **2**(1).
8. ICA. *Spreadsheet Design.*, Institute of Chartered Accountant in England and Wales, 1994.
9. McDowall, R.D., C. Burgess, and B. Hardcastle, *Spreadsheets: heaven or hell.* Scientific Data Management, 1999. **3**(3): p. 8-17.
10. Object Mentor. *JUnit*: 2006. http://www.junit.org
11. Panko, R.R. and R.P. Halverson Jr. *Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks*. in *29th Annual Hawaii International Conference on System Sciences*. 1996. Hawaii: IEEE.
12. Panko, R.R., *What we know about spreadsheet errors.* Journal of End User Computing, 1998. **10**(2): p. 15-21.
13. Wichmann, B.A., R.M. Barker, and G.I. Parkin. *Validation of Software in Measurement Systems.* Software Support for Metrology Best Practice Guide No. 1, NPL, March 2004. http://www.npl.co.uk/ssfm/download/bpg.html#ssfmbpg1