Embunit User Guide

Release 1.0.1

Table of contents

Table of contents 2
1 Introduction 4
  Conventions....4
2 Getting Started....5
  Installing Embunit 6
  License management 7
  Building unit tests 8
  Configuration 8
  Getting help 8
  Examples....9
  Uninstalling Embunit....9
  System requirements....9
3 Test suite 10
  Edit test suite 11
  Test result generation....12
4 Includes 13
5 Stubs....14
6 Test case 15
  Edit test case 16
  Test case items 17
  Call 18
    Value 19
    Variable 20
  Context 20
  Create....21
    Initializers 22
    Array initializers 23
  Delete 23
  Test 24
    Assertions 25
    Tolerance 26
7 New....27
8 Edit & Delete....28
9 Up/Down 29
10 Generate test harness 30
  Command line interface....30
  Output folder....31
  File names....31
11 Test framework functions....32
  Target-specific functions....33
  Target-independent functions....34
  Memory allocation in C 35

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

12 How do I? 36
Report test results 36
Save test results to a file 36
Create test-specific stub code 36
Create a whitebox test in C++ 37
Disable exception handling 37
Assign or increment a value 38
Structure my tests 38
Insert a copyright block 38
Compare two arrays 39
Compare two strings 40
13 Error messages 41
Errors 41
Warnings 42
Code generation errors 42

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

1 Introduction

Thank you for choosing Embunit by Apollo Systems Contracts.

Embunit is a unit testing tool aimed at the particular needs of embedded systems development. It has been developed with flexibility in mind and makes no assumptions about the resources available on the target system, or the tools used to download and run the unit tests.

What does it do?

Embunit essentially does two things:

- It allows you to specify and organise your unit tests at a high level.
- It automatically generates the source code for your unit tests, including code to log the test results.

Your test specification is stored as a text file in XML format. It can be kept under version control and changes can be reviewed using a file comparison utility. The automatic code generation means that you have more time to focus on what needs to be tested. You can write tests first, even if you have not finished defining the interface. The generated source code can be reviewed and also kept under version control.

What does it not do?

- It does not compile your unit tests.
- It does not run your unit tests.
- It does not generate reports.

Due to the wide range of tools used for embedded systems development Embunit cannot build and run your tests, you must do this yourself. Many modern IDE's can be scripted to automate this process, consult the documentation for the tools you are using. The way that your system indicates whether the tests have passed or failed is heavily dependent on the tools you are using and the available I/O. Embunit uses a set of functions that can be customised to suit your target. The example implementation supplied with Embunit uses the standard output to report test results.

What languages does it support?

Embunit supports C and C++ and is fully compatible with Embedded C++.

Conventions

The following conventions are adopted throughout this User Guide:

- When the location of a folder or file created at installation time is specified, the default installation folder (C:\Program Files\Apollo Systems\Embunit) is assumed.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

2 Getting Started

Installing Embunit

Embunit must be licensed before you can use it.

License management

Once Embunit is running you can start creating unit tests straight away.

Embunit creates an empty test suite, to which you add your test cases, specify what header files to include, and reference any stub files that are needed.

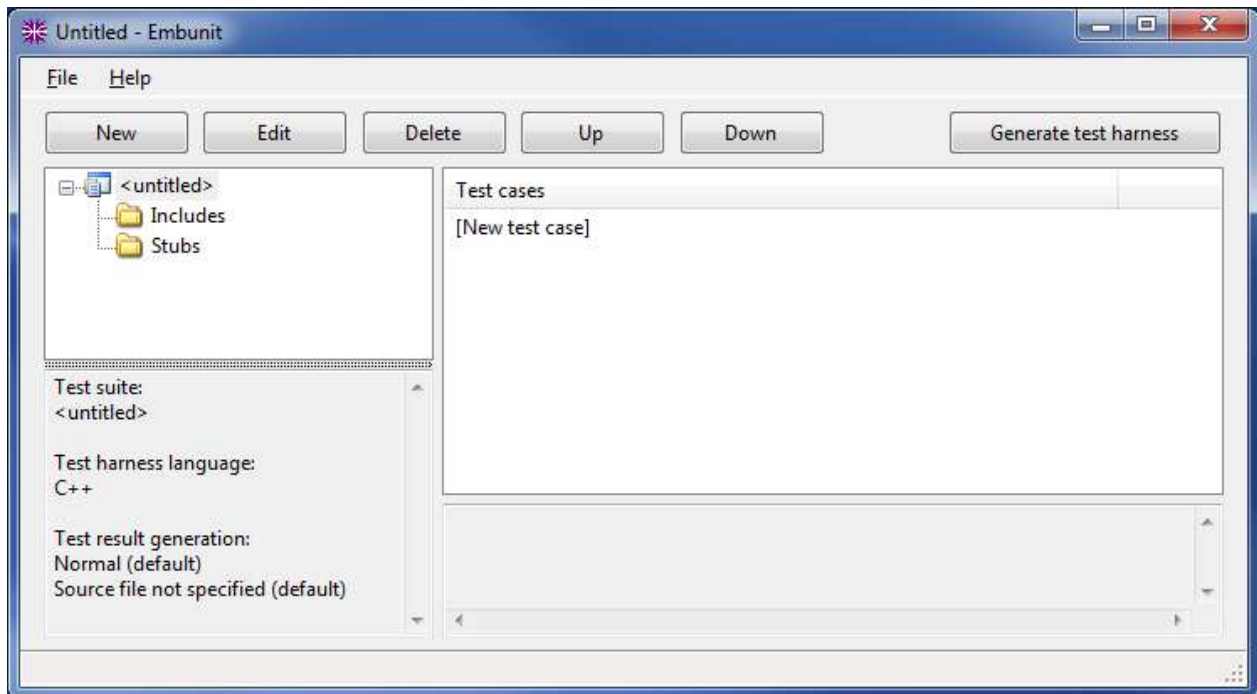
The GUI has four re-sizeable panels:

- The tree view (top left) shows the hierarchical organisation of the test suite.
- The list view (top right) shows a list of elements at the current node in the tree.
- The details box (bottom left) displays more detail about the current node.
- The output box (bottom right) displays messages when generating the test harness.

Use the **New**, **Edit**, and **Delete** buttons to create and edit the various elements of the test suite.

Use the **Up** and **Down** buttons to change the order of the elements.

Use the **Generate test harness** button to generate the source code for your unit tests.



Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Installing Embunit

You must have admin rights in order to install Embunit (see Pre-requisites below).

Embunit is distributed as a self-extracting .exe file.

1. Run the self-extracting .exe file and follow the prompts to unzip the files to a suitable location.
2. Locate the **setup.exe** file that has just been unzipped and run it.
3. If necessary, the .NET 4.0 Client Profile is installed and you are prompted to restart your computer.
4. Following the restart the main Embunit installer starts automatically. If it does not, run setup.exe again.
5. Follow the prompts to install Embunit.

Pre-requisites

Embunit uses the .NET 4.0 Client Profile.

Setup.exe checks whether this is present and installs it if necessary.

Admin rights are required in order to install the .NET 4.0 Client Profile.

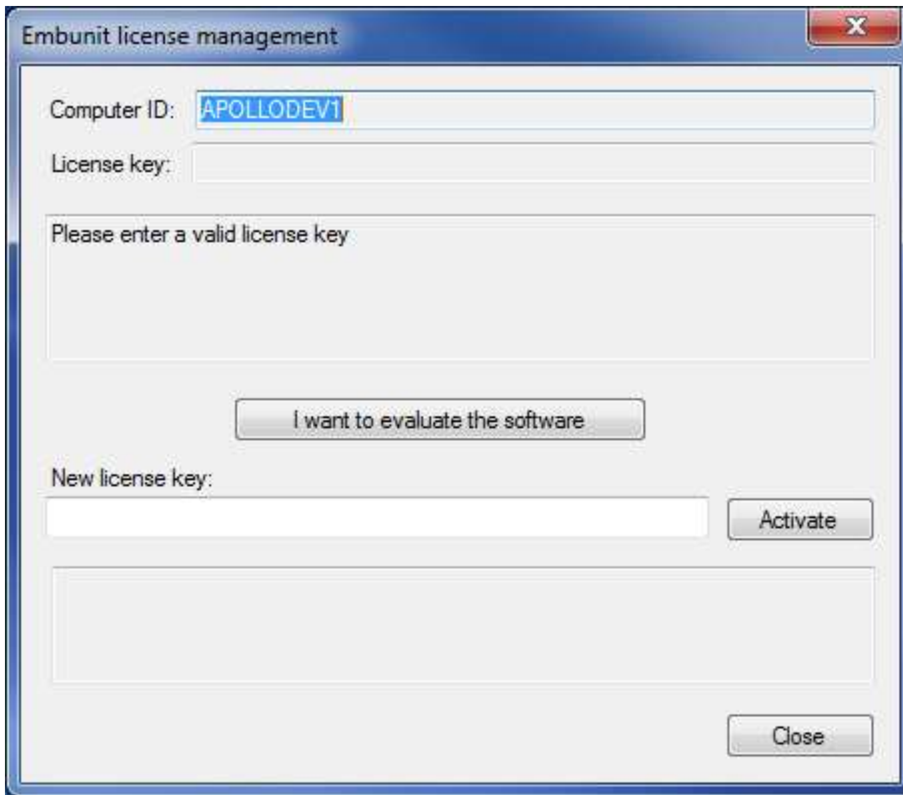
If the .NET 4.0 Client Profile is already installed Embunit can be installed by users without admin rights.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

License management

The license management form is displayed whenever you run Embunit without a valid license.

You can also access the form from the **Help** menu.



To start the 30-day evaluation of Embunit, click the button labelled **"I want to evaluate the software"**.

The number of days left for evaluation is displayed in the box above the button.

Once the evaluation has started the button is disabled.

If you have been supplied with a license key; enter it in the **"New license key:"** edit box, and click the **Activate** button.

If the license is accepted it moves to the **"License key:"** box at the top of the form, otherwise an error message is displayed in the box at the bottom of the form.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Building unit tests

When you click the **Generate test harness** button Embunit generates the source code for your unit tests.

This comprises a header and source file for the test suite (containing the main() function), and a separate source file for each test case.

The files are created in the Output folder.

To build a fully functioning test harness the generated source files need to be compiled and linked with:

- The test framework source files (see below)
- Your source files (including any stub files)
- Any library files that are required

The test framework source files contain common functions for indicating test results, counting failures, and so on. The functions are spilt into two groups; those that need to be customised for each target, and those that do not. Furthermore, there are different versions for C and C++, so you need to build with the appropriate version:

- C:\Program Files\Apollo Systems\Embunit\Lib\Embunitc.h
- C:\Program Files\Apollo Systems\Embunit\Lib\Embunitc.c
- C:\Program Files\Apollo Systems\Embunit\Lib\TestResult.c (customise for each target)

OR

- C:\Program Files\Apollo Systems\Embunit\Lib\Embunit.h
- C:\Program Files\Apollo Systems\Embunit\Lib\Embunit.cpp
- C:\Program Files\Apollo Systems\Embunit\Lib\TestResult.cpp (customise for each target)

Test framework functions

Configuration

There are currently no configuration options for Embunit.

The only thing you have to do is customise a small set of functions to suit each of your target systems.

Examples are provided that work straight out of the box if your target supports printf or std::cout.

Target-specific functions

Getting help

You can access this user guide from the **Help** menu. It is also supplied in PDF and Word format in the Doc folder:

- C:\Program Files\Apollo Systems\Embunit\Doc

You can obtain the latest version of this guide and further information at:

http://www.embunit.com

If you are still having problems, send an email to support@embunit.com

We aim to respond to emails within one working day.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Examples

Example projects in C and C++ can be found in the Samples folder:

- C:\Program Files\Apollo Systems\Embunit\Samples

Uninstalling Embunit

To uninstall Embunit; go to Control Panel and then, depending on your operating system, select:

- Add/Remove Programs (XP)
- Programs and Features (Vista)
- Programs (Windows 7)

If necessary (Windows 7), click the "Uninstall a program" link to obtain a list of programs.

Find Embunit in the list of programs, select it and click the appropriate button (**Uninstall** or **Remove**).

System requirements

The minimum system requirements for Embunit are:

- Windows 7, Windows Vista, or Windows XP SP3

- 512MB of RAM

- 80MB of free disk space

- 800 x 600 screen resolution

Embunit uses the .NET 4.0 Client Profile; this is installed automatically if not already present.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

3 Test suite

All the unit tests, includes, and stub file references that together make up your test specification are known as a 'test suite'.

Embunit allows you to create, save, and open existing test suites using the File menu.

Test suites are stored as text files in XML format.

File extensions

Save your test suites with the file extension .eut.

The installer associates files with the extension .eut with Embunit.

GUI

The test suite is at the root of the tree in the tree view.

When you click the test suite a list of test cases is displayed in the list view, and information about the test suite appears in the details box.

To edit information about the test suite select it in the tree view and click the **Edit** button.

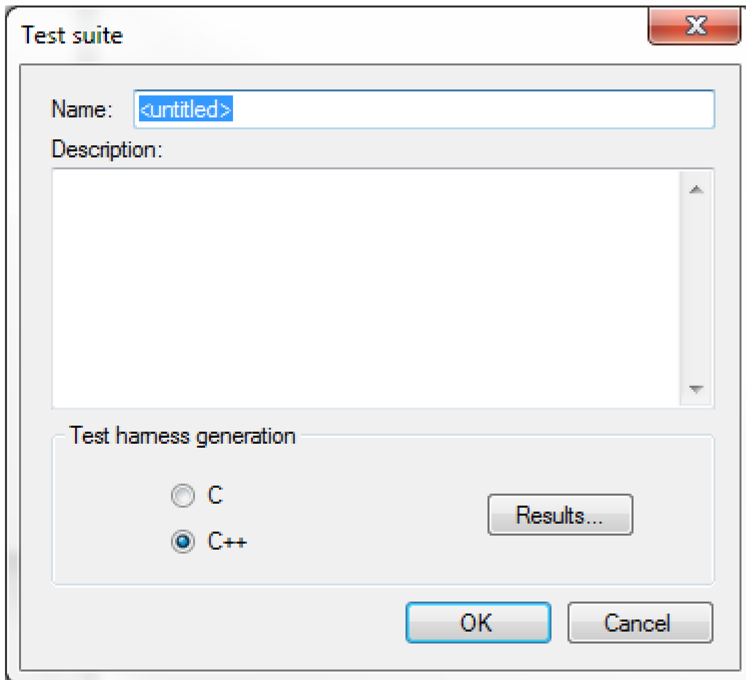
Edit test suite

Note: The New and Delete buttons have no effect on the test suite. If you want to create a new test suite use the File menu.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Edit test suite

The **Test suite** dialog is displayed when you edit the test suite.



Name

The name you choose for the test suite is used during test harness generation to create a name for the test suite header and source files, and in the case of C++ the test suite class name as well.

File names

Description

The description is inserted in the test suite header and source files during test harness generation.

Warning: Do NOT insert newlines in the test case description (using ctrl-return) or the generated files may not compile.

Test harness generation

Use the radio buttons to select whether you want to generate C or C++ code.

Click the **Results** button to edit the test result generation options.

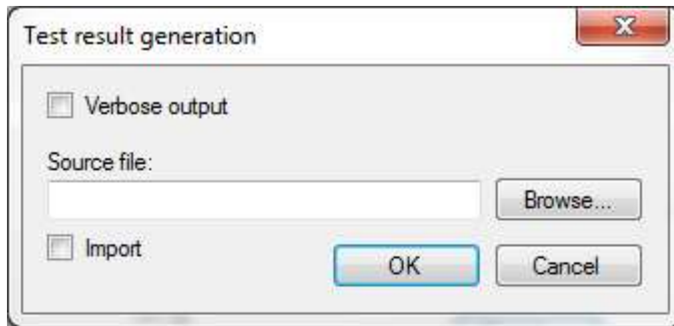
Test result generation

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Test result generation

The **Test result generation** dialog is displayed when you click the **Results** button in the test suite dialog.

[Edit test suite]



Verbose output

Check this box if you want the test harness to generate a string containing the parameters of each test.

This can be useful to quickly identify which test has failed, but does increase the size of the test harness.

If this box is checked a call to embunit_Info() is inserted for every test during test harness generation (see Target-independent functions).

Source file

Specify the source file that contains your target-specific test framework functions.

Target-specific functions

This entry is optional and can be used in two ways:

- Simply as a reference to remind you which file to include in your build, or
- To import the code into the test suite source file during test harness generation.

Import

Check this box if you want to import the code during test harness generation.

Test harness generation:

- If the code is not being imported, Embunit inserts a comment specifying the path and file name in the test suite source file.

- If the code is being imported, Embunit inserts a comment and then inserts the entire contents of the file in the test suite source file, followed by another comment to indicate the end of the code. If the file cannot be opened Embunit inserts a comment to that effect.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

4 Includes

The Includes 'folder' contains a list of files that need to be included in the test case source code.

Typically these are the header files for the classes and functions you are testing.

There is an Includes 'folder' at the top (test suite) level where you can add files that need to be included in every test case.

Each test case also has its own Includes 'folder' where you can add files that are specific to that particular test case.

List view

When you click the Includes 'folder' a list of file names (without any path information) is displayed in the list view.

If you expand the Includes 'folder' and click on an individual file (in the tree view) the path and file name is displayed in the list view.

Test harness generation

Embunit inserts a #include statement in the test case source code for each file in the list.

The #include statement contains the path and file name.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

5 Stubs

Create test-specific stub code

The Stubs 'folder' contains a list of files containing stub code that need to be built with the test harness.

You can use these entries in two ways:

- Simply as a reference to remind you which files to include in your build, or
- To import the stub code into the test harness source files.

There is a Stubs 'folder' at the top (test suite) level where you can add files that need to be imported into the test suite source file.

Tip: Use this if you have stub code that is common to all test cases.

Each test case also has its own Stubs 'folder' where you can add files that only need to be imported into one particular test case source file.

List view

When you click the Stubs 'folder' a list of file names (without any path information) is displayed in the list view.

If you expand the Stubs 'folder' and click on an individual file (in the tree view) two columns are displayed in the list view. The first contains the path and file name; the second indicates whether the stub file will be imported when the test harness is generated.

Test harness generation

- If the stub file is not being imported, Embunit inserts a comment specifying the path and file name in the source file.

- If the stub file is being imported, Embunit inserts a comment and then inserts the entire contents of the stub file in the source file, followed by another comment to indicate the end of the stub code. If the stub file cannot be opened Embunit inserts a comment to that effect.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

6 Test case

Test cases are central to Embunit; they are where you define what your unit tests should do.

The core of a test case is a list of operations known as test case items. These are the building blocks from which you construct your tests.

Each test case is a separate function/method in your test harness, so its name must be unique (within the test suite).

Each test case also has its own Includes and Stubs 'folders'.

Includes
Stubs

Viewing test cases

Test cases are contained within the test suite.

When you click on the test suite (in the tree view) a list of test cases is displayed in the list view.

When you expand the test suite 'node' the test cases are displayed in the tree view.

When you click a test case in the tree view a list of its test case items is displayed in the list view, and information about the test case appears in the details box.

The order of test cases and test case items can be changed using the Up/Down controls.

Create a test case

To create a test case; select the test suite and click the **New** button.

A dialog box is displayed allowing you to enter the test case name etc.

Edit test case

*The new test case is appended to the end of the list. To insert a new test case at a particular position; select the test suite as before and then select the desired insertion point in the list view. Finally, click the **New** button.*

Add a test case item

To add a new test case item; select the test case in the tree view and click the **New** button.

Test case items

*The new test case item is appended to the end of the list. To insert an item at a particular position; select the test case as before and then select the desired insertion point in the list view. Finally, click the **New** button.*

Editing the test case

To edit a test case item; select the item and click the **Edit** button.

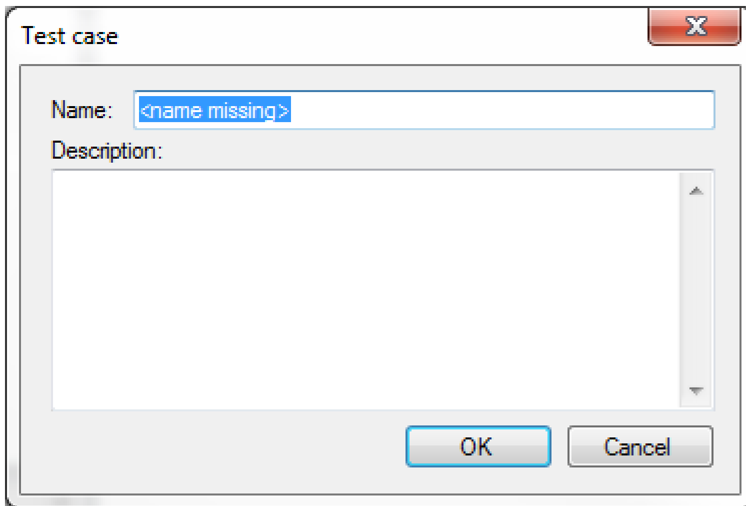
To edit the information about the test case; select the test case and click the **Edit** button.

Edit test case

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Edit test case

The **Test case** dialog is displayed when you create or edit an existing test case.



Name

The name you choose for the test case is used during test harness generation to create a name for the test case source file, and the test case function/method.

File names

Description

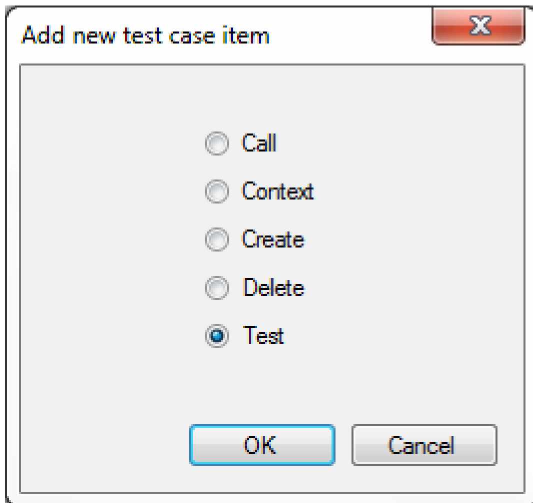
The description is inserted in the test case source file during test harness generation.

Warning: Do NOT insert newlines in the test case description (using ctrl-return) or the generated files may not compile.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Test case items

This dialog is displayed when you add a new item to a test case.



Test case items are the building blocks of your test cases.

They comprise a simple set of operations that can be used to define your test specification.

Certain items can be nested within other test case items.

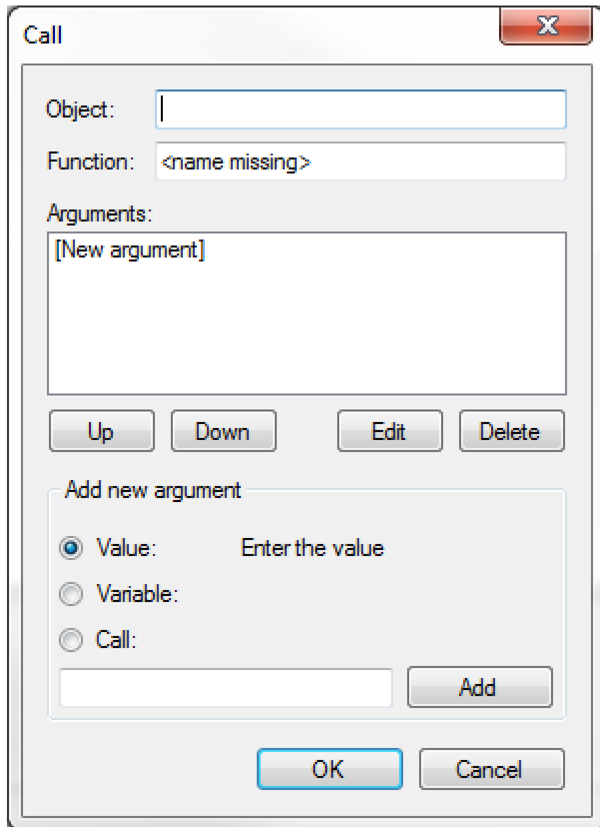
Select the type of test case item you want to create, and click **OK**.

- Call
- Context
- Create
- Delete
- Test

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Call

The **Call** dialog is displayed when you create or edit an existing 'Call' test case item.



Use this item to call a function or method in your test case.

Note: Calls can be nested in other test case items.

Object

If you are calling a method of an object, enter the name of the object. Otherwise leave this field blank.

Do not add a member access operator. Embunit automatically generates the correct operator ('.' or '->') depending on how the object is created (see Create).

Function

Enter the name of the function or method.

Arguments

The arguments are shown in a list, and are passed to the function/method in the order they are listed.

Use the **Up** and **Down** buttons to change the order.

Use the **Edit** and **Delete** buttons to edit or delete existing arguments.

Add new argument

Select the position in the list where you want to insert a new argument.

By default arguments are added at the end of the list.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Arguments take one of three forms:

- Value
- Variable
- Function/method call

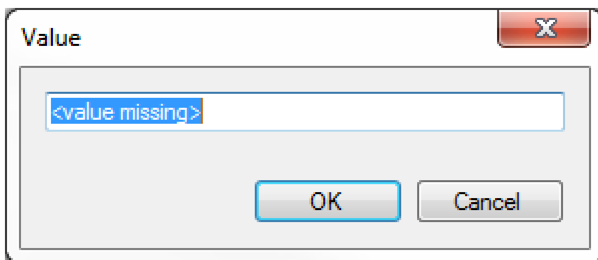
Use the radio buttons to specify whether the new argument is a value, a variable, or a function/method call.

Using the text box, enter the **value**, the **name** of the **variable**, or the **name** of the **function/method**, and click the **Add** button.

You should now edit the argument and enter any further information that may be required.

Value

This dialog is displayed when you edit a 'Value'.



Values are used as:

- Arguments to function/method calls
- Initializers
- Operands in tests

Embunit inserts values in your test harness source code exactly as you specify them, without any checks or special formatting.

This makes values very flexible, but also means that you have to take care of any quotes, brackets, or other formatting that may be necessary.

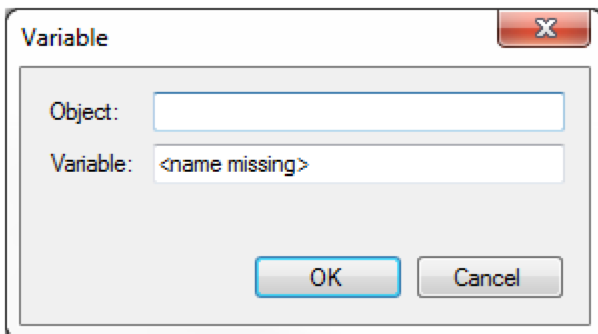
Examples:

- "A string constant"
- 'Z'
- (3 * 4)

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Variable

This dialog is displayed when you edit a 'Variable'.



Variables are used as:

- Arguments to function/method calls
- Initializers
- Operands in tests

Object

If the variable is a member of an object, enter the name of the object. Otherwise leave this field blank.

Note: If you are using C the object can be an instance of a struct.

Do not add a member access operator. Embunit automatically generates the correct operator ('.' or '->') depending on how the object is created.

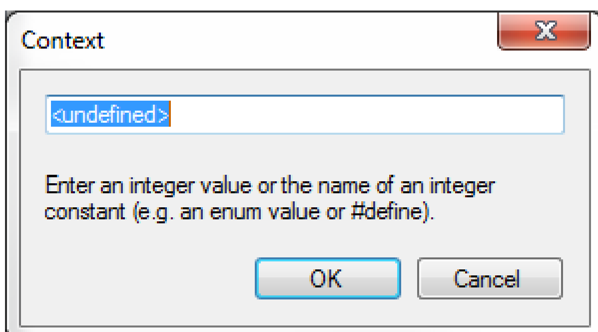
Create

Variable

Enter the name of the variable.

Context

The **Context** dialog is displayed when you create or edit an existing 'Context' test case item.



The context is an integer value that can be accessed from stub code.

It can be set anywhere in a test case and retains its value until set by another context test case item.

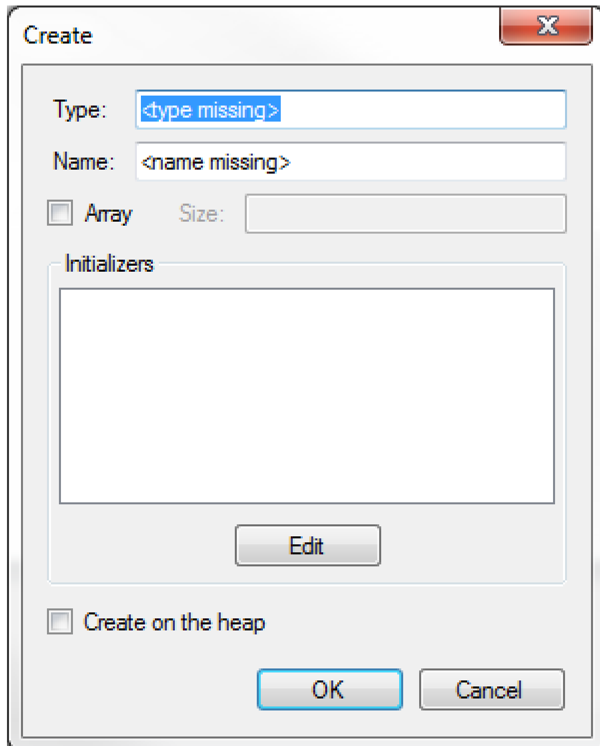
Its value is preserved across test cases.

For readability it is recommended that you use an enum or a set of #define statements to specify the context values. These should be put in a header file and included in the test suite (see Includes).

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Create

The **Create** dialog is displayed when you create or edit an existing 'Create' test case item.



Use this item to create an object in your test case.

Note: A 'Create' item can be nested in a 'Test' item for the purpose of testing constructor exceptions.

Type

Enter the type of the object. You can use built in types or your own classes, structures, and enums.

Name

Enter a name for the object.

Note: You cannot use the same name to create different objects in the same test case.

Array

Check the box if you want to create an array, and enter the **Size** of the array in the text box.

Note: If you have already specified initialization values before checking the box, Embunit converts them to a comma separated list of values. If any initializer is a 'Variable' or function/method 'Call' the conversion is aborted and Embunit displays an error message.

Initializers

You can specify initialization values for the object.

Use the **Edit** button to create or edit initialization values.

Initializers
Array initializers

*Note: Arrays created on the heap cannot have initializers, Embunit warns you about this situation when you click the **OK** button.*

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Create on the heap

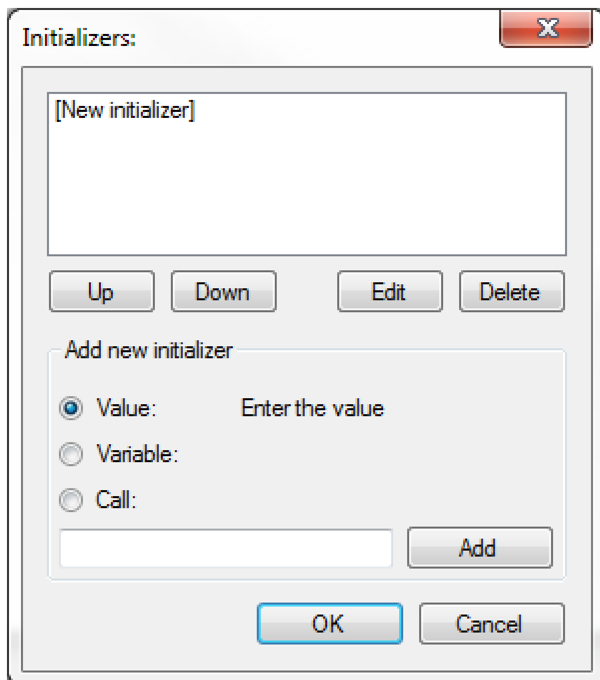
By default objects are created as local variables on the stack. Check the **Create on the heap** box if you want the object to be created on the heap and accessed using a pointer.

All objects created on the heap must have a corresponding 'Delete' test case item, otherwise a memory leak will occur when running the test harness.

Note: Embunit automatically generates the correct member access operator, so you can switch between creating on the stack and the heap without affecting any other test case items.

Initializers

This dialog is displayed when you edit the initializers of a 'Create' test case item.



Note: This dialog is not used for Array initializers.

The initializers are shown in a list.

If you generate C code, only the first initializer is used; any others are ignored.

If you generate C++ code, the initializers are passed to the constructor in the order they are listed.

Use the **Up** and **Down** buttons to change the order.

Use the **Edit** and **Delete** buttons to edit or delete existing initializers.

Add new initializer

Select the position in the list where you want to insert a new initializer.

By default initializers are added at the end of the list.

Initializers take one of three forms:

- Value
- Variable

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

- Function/method Call

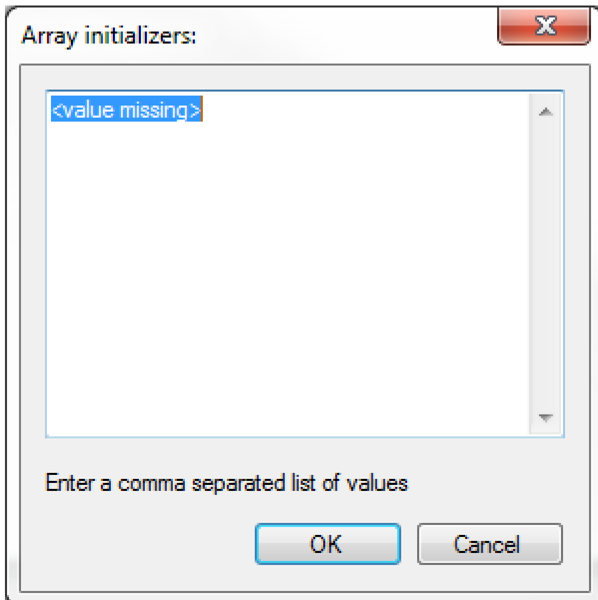
Use the radio buttons to specify whether the new initializer is a value, a variable, or a function/method call.

Using the text box, enter the **value**, the **name** of the **variable**, or the **name** of the **function/method**, and click the **Add** button.

You should now edit the initializer and enter any further information that may be required.

Array initializers

This dialog is displayed when you edit the initializers of a 'Create' test case item.

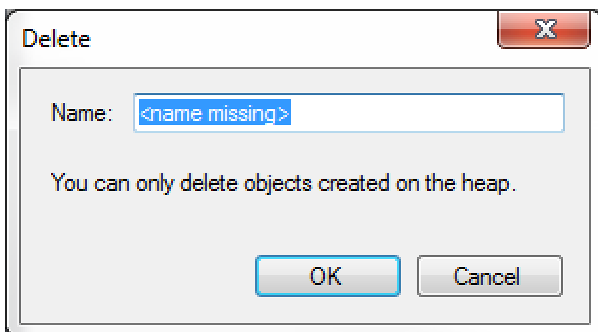


Note: This dialog is only used for arrays, for single objects see Initializers.

Enter a comma separated list of initializers.

Delete

The **Delete** dialog is displayed when you create or edit an existing 'Delete' test case item.



All objects created on the heap must have a corresponding 'Delete' test case item (see Create).

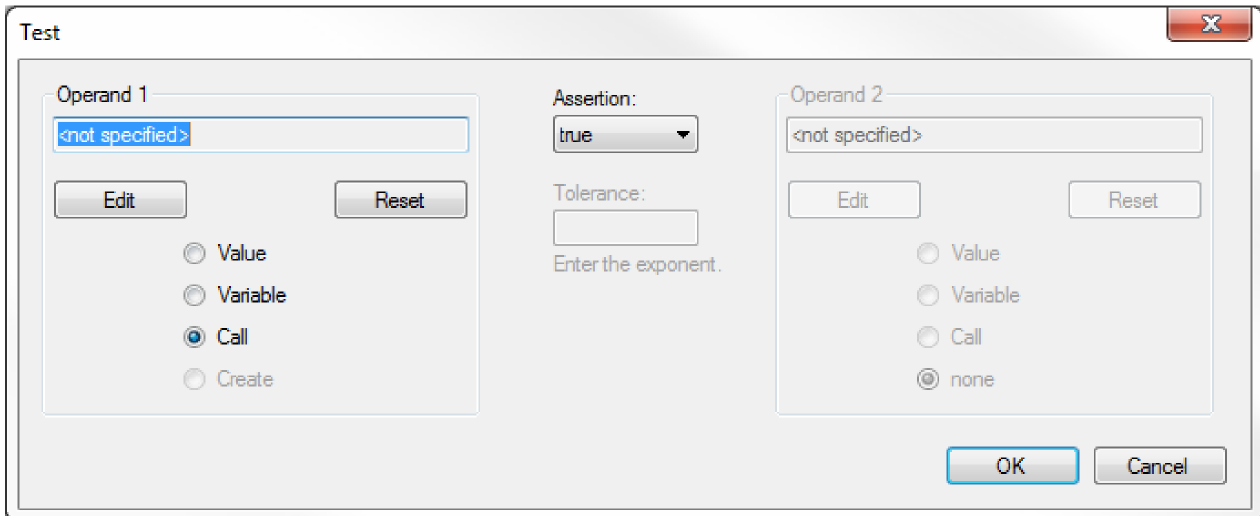
Name

Enter the name of an object created on the heap.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Test

The **Test** dialog is displayed when you create or edit an existing 'Test' test case item.



This is the key building block of your unit tests, it is where you specify what you expect from the software under test.

All tests involve one or two operands, and an assertion.

If the assertion is true the test has passed, if it is false the test has failed.

Embunit displays error/warning messages if the operands and assertion you select are not compatible.

Operand 1

The operand can be one of four types:

- Value
- Variable
- Function/method Call
- Create

Note: The Create type can only be used to test constructor exceptions, and is greyed out unless the assertion is 'throws'.

Use the radio buttons to select the type of operand then click the **Edit** button.

If the type of operand has already been defined the radio buttons are greyed out.

To **change** the **type** of operand click the **Reset** button.

You are prompted to confirm that you want to reset the operand.

Assertion

Select the assertion from the drop down list.

Assertions

Tolerance

The tolerance is used when testing that two floating point numbers are equal (or not equal). It is greyed out at all other times.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

[Tolerance](#)

Operand 2

Operand 2 is greyed out if the selected assertion only takes one operand.

The operand can be one of four types:

- [Value](#)
- [Variable](#)
- Function/method [Call](#)
- none

Note: The default operand type 'none' cannot be used with assertions that require two operands.

Use the radio buttons to select the type of operand then click the **Edit** button.

If the type of operand has already been defined the radio buttons are greyed out.

To **change** the **type** of operand click the **Reset** button.

You are prompted to confirm that you want to reset the operand.

Assertions

You can choose from the following assertions:

Assertion	# operands	Test
true	1	Test that Operand 1 is true
false	1	Test that Operand 1 is false
==	2	Test that Operand 1 equals Operand 2
!=	2	Test that Operand 1 does not equal Operand 2
>	2	Test that Operand 1 is greater than Operand 2
>=	2	Test that Operand 1 is greater than or equal to Operand 2
<	2	Test that Operand 1 is less than Operand 2
<=	2	Test that Operand 1 is less than or equal to Operand 2
≈≈	2	Test that Operand 1 is almost equal to Operand 2 (**see Note**)
!≈	2	Test that Operand 1 is not almost equal to Operand 2 (**see Note**)
throws	1 or 2	Test that Operand 1 throws an exception. Operand 2 is optional and is used to specify the exception type.

***Note**: These assertions compare two floating point numbers for (non-)equivalence using the tolerance value. This is achieved by calling the embunit_IsAlmostEqual() test framework function (see [Target-specific functions](#)).*

Comparing C-style strings and arrays

To test whether two C-style strings are the same call the standard library function strcmp(), passing the two strings, and test the return value.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Example:

Operand 1: strcmp(string1, string2)
Assertion: ==
Operand 2: 0

Remember to add string.h to the Includes.

Arrays can be compared in a similar way using memcmp().

Tolerance

The tolerance is used when testing that two floating point numbers are equal (or not equal).

This is achieved by calling the embunit_IsAlmostEqual() test framework function (see Target-specific functions).

The default implementation of embunit_IsAlmostEqual() uses the value in the tolerance box as a power of ten (exponent).

This is best illustrated using some examples:

Value entered in GUI	Tolerance
-3	0.001
-1.301	0.05
-1	0.1
0	1
1	10
1.30103	20
3	1000

The default implementation of embunit_IsAlmostEqual() returns false (not equal) if:

Operand_1 > (Operand_2 + Tolerance)

OR

Operand_2 > (Operand_1 + Tolerance)

You are free to change the implementation of embunit_IsAlmostEqual() and the interpretation of the value entered in the GUI if you so wish.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

7 New

New items can be created by selecting the tree view or the list view and clicking the **New** button.

Tree View

To create a new item, select a node in the tree view and click the **New** button.

In general, the new item is created at the end of the corresponding list in the list view.

The exception to this is when you have selected an individual file in the Includes or Stubs 'folder'. In this case the selected file serves as the insertion point and the new item is inserted above it.

List view

To create a new item in the list view, select the insertion point and click the **New** button.

The new item is inserted above the insertion point.

The last row in the list is a dummy item containing text in square brackets (e.g. [New test case]). To add a new item at the end of the list select this row and click the ***New*** *button.*

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

8 Edit & Delete

There are Edit and Delete buttons on the main GUI, and on some of the test case item dialog boxes (Call, Create-Initializers).

On the main GUI the **Edit** and **Delete** buttons operate on items in the tree view and the list view.

Select the required item and click the appropriate button.

Note: Includes and Stubs 'folders' cannot be edited or deleted. The test suite cannot be deleted, but it can be edited.

Delete control

You can delete more than one item at a time by selecting multiple items using the **Shift** or **Ctrl** keys (does not apply to the tree view).

When you delete something on the main GUI you are given the option to cancel the operation.

When you delete something in a test case item dialog box you do not get an option to cancel the operation.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

9 Up/Down

There are Up/Down buttons on the main GUI, and on some of the test case item dialog boxes (Call, Create-Initializers).

On the main GUI the **Up** and **Down** buttons operate on items in the list view. They do not operate on items in the tree view.

Select the required item in the list and use the appropriate button to move it up or down.

Note: Only one item is moved, if you select multiple items only the top item is moved.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

10 Generate test harness

You can generate the source code for your unit tests by clicking the **Generate test harness** button on the GUI, or using the Command line interface.

Embunit automatically saves the test suite before generating the source code, or prompts you to enter a file name if it is a new test suite that has never been saved.

The results of the code generation operation are displayed in the output box and also saved in a log file.

The log file is overwritten each time you generate the test harness.

Embunit generates a header and source file for the test suite (containing the main() function), and a separate source file for each test case.

The files are created in the Output folder.

If you generate C code, each test case is a separate function called by main().

If you generate C++ code, a test suite class is created and each test case is a separate method called by main().

Command line interface

You can use the command line interface to generate a test harness:

thgen [/c] "test_suite_file" ["output_folder"]

Parameter	Description
/c	(optional) generate C code (the default is C++)
"test_suite_file"	an XML test suite file (created using Embunit)
"output_folder"	(optional) path of the output folder

If "output_folder" is not specified the files are created in the Projects folder (see Output folder).

Note: Quotation marks are only necessary where the path or filename contains spaces.

Where is thgen?

The executable thgen.exe is located in the Bin folder (below the main installation folder).

- C:\Program Files\Apollo Systems\Embunit\Bin\thgen.exe

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Output folder

The output folder is a **folder** that Embunit creates **in the same folder as your test suite file.**

The name of the folder is the same as your test suite file, **minus the file extension**. If there is no file extension Embunit appends an underscore to differentiate between the two names.

If Embunit fails to create the output folder it uses the Projects folder instead:

- C:\Program Files\Apollo Systems\Embunit\Projects

If the Projects folder does not exist (and cannot be created) Embunit uses the local application folder:

- C:\Program Files\Apollo Systems\Embunit\Bin

File names

The test suite header and source file name is generated from the test suite name by replacing any invalid characters with underscores, and appending "_ts".

The test case source file name is generated from the test case name by replacing any invalid characters with underscores, and appending "_tc".

The name of the log file is the same as your test suite file, but with the extension .log.

If the original extension is .log Embunit appends another .log to differentiate between the files.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

11 Test framework functions

The test framework functions are common functions for indicating test results, counting failures, and so on. The functions are spilt into two groups; those that are target-specific, and those that are not.

Both groups can be customised to suit your particular needs.

The source files are located in the Lib folder. Separate versions are supplied for C:

- C:\Program Files\Apollo Systems\Embunit\Lib\Embunitc.h
- C:\Program Files\Apollo Systems\Embunit\Lib\Embunitc.c
- C:\Program Files\Apollo Systems\Embunit\Lib\TestResult.c (target-specific)

and C++:

- C:\Program Files\Apollo Systems\Embunit\Lib\Embunit.h
- C:\Program Files\Apollo Systems\Embunit\Lib\Embunit.cpp
- C:\Program Files\Apollo Systems\Embunit\Lib\TestResult.cpp (target-specific)

Note: If you have a number of different targets, you may prefer to keep your customised files with the rest of your project files, rather than the Lib folder.

Header files

- embunitc.h
- embunit.h

The test framework functions that are called by the generated source files are declared in these header files. The interfaces should not be changed.

Customisation

- Target-specific functions
- Target-independent functions
- Memory allocation in C

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Target-specific functions

The functions that need to be customised to suit your target hardware are described in the table below:

Function	Called	Notes
embunit_ReportBegin()	By embunit_Begin()	Reports the test suite name
embunit_ReportEnd()	By embunit_End()	Reports the test suite name and number of test failures
embunit_ReportPass()	By embunit_Test()	Reports the result
embunit_ReportFail()	By embunit_Test()	Reports the result
embunit_ReportAborted()	By embunit_AbortTest()	Reports the result
embunit_IsAlmostEqual()	When testing floating point numbers for (non-)equivalence	

embunit_IsAlmostEqual() is declared in the header files embunitc.h and embunit.h. You can change the implementation of the function but you cannot change the interface.

Note: The C and C++ versions have different return types.

As far as the other functions are concerned, you can change both the implementation and the interface to suit your requirements. The functions are declared in embunitc.c and embunit.cpp using the **extern** keyword.

Example implementations of all the functions are provided:

- TestResult.c (uses printf to report results)
- TestResult.cpp (uses std::cout to report results)
- TestResulttoFile.c (uses fprintf to send results to a file)
- TestResultToFile.cpp (uses std::ofstream to send results to a file)

You must tailor the functions to suit the I/O available on your target system.

Report test results

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Target-independent functions

These functions do not need to be customised, but you may wish to modify them to collect and report additional metrics (such as the total number of tests executed).

- embunitc.c

Function	Called	Notes
embunit_Begin()	At the start of the test suite	Sets the test suite name and calls embunit_ReportBegin()
embunit_End()	At the end of the test suite	Calls embunit_ReportEnd()
embunit_Case()	At the start of each test case	Sets the test case number
embunit_Step()	At the start of each test step	Sets the test step number
embunit_Context()	For each 'Context' item in the test suite	Sets the context
embunit_Info()	Before each test (verbose output only)	Sets the test info string
embunit_GetCase()	By stub code	Gets the test case number
embunit_GetStep()	By stub code	Gets the test step number
embunit_GetContext()	By stub code	Gets the context
embunit_Failures()	By main() at the end of the test suite	Value returned by main()
embunit_Test()	After each test	Calls embunit_ReportPass() or embunit_ReportFail()
embunit_AbortTest()	If the test case is aborted	Calls embunit_ReportAborted()

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

- embunit.cpp

Note: The C++ get/set functions embunit_Case(), embunit_Step(), and embunit_Context() are overloaded.

Function	**Called**	**Notes**
embunit_Begin()	At the start of the test suite	Sets the test suite name and calls embunit_ReportBegin()
embunit_End()	At the end of the test suite	Calls embunit_ReportEnd()
embunit_Case()	At the start of each test case	Sets the test case number
embunit_Step()	At the start of each test step	Sets the test step number
embunit_Context()	For each 'Context' item in the test suite	Sets the context
embunit_Info()	Before each test (verbose output only)	Sets the test info string
embunit_Case()	By stub code	Gets the test case number
embunit_Step()	By stub code	Gets the test step number
embunit_Context()	By stub code	Gets the context
embunit_Failures()	By main() at the end of the test suite	Value returned by main()
embunit_Test()	After each test	Calls embunit_ReportPass() or embunit_ReportFail()
embunit_AbortTest()	If the test case is aborted	Calls embunit_ReportAborted()

Memory allocation in C

When an object is created on the heap the C code generator calls the macro **embunit_calloc** (defined in embunitc.h).

Similarly, when the object is deleted the C code generator calls the macro **embunit_free** (also defined in embunitc.h).

These macros are implemented using the standard library functions malloc() and free().

If you have a custom memory allocator you can redefine these macros to suit.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

12 How do I?

This section contains useful tips to help you get the most out of Embunit.

Report test results

The simplest way to report test results from an embedded system is via a serial port.

If your system does not have a serial port there are several other options (depending on the I/O that is available to you) including:

- Ethernet
- Saving to a file
- Storing in non-volatile memory
- Toggling a digital I/O line

If none of these approaches are suitable you may have to store the results in memory and use your debugger to retrieve them.

As an alternative to running on the target hardware, some tools provide a simulator that allows you to run your code on the PC while still interacting with simulated on-chip peripherals. These usually have a console window to which you can send your test results.

As a last resort, algorithms that do not interact with hardware can be tested by creating a console application to run on the PC, but watch out for portability issues concerning word sizes and endianness.

Save test results to a file

To save the test results to a file you need to modify the target-specific functions as follows:

- Modify embunit_ReportBegin() to create the file.
- Modify embunit_ReportPass(), embunit_ReportFail(), and embunit_ReportAborted() to update the file.
- Modify embunit_ReportEnd() to close the file.

Example implementations are given in TestResultToFile.c and TestResultToFile.cpp, which can be found in the Lib folder.

Create test-specific stub code

You can determine which test is currently running by calling test framework functions from your stub code. This makes it easy to specify the behaviour for an individual test, or a group of tests.

If your test harness is in C you need to include **embunitc.h** and use these functions:

- `int embunit_GetCase() - the test case number`
- `int embunit_GetStep() - the step number within a given test case`
- `int embunit_GetContext() - the test` Context

If your test harness is in C++ you need to include **embunit.h** and use these functions:

- `int embunit_Case() - the test case number`
- `int embunit_Step() - the step number within a given test case`
- `int embunit_Context() - the test` Context

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Create a whitebox test in C++

Add a friend declaration to the class being tested.

By making the test suite class a friend of the class under test you gain access to its private members.

Example:

```
class person    // Class being tested
{
    friend class person_ts;    // Test suite class
    ...
};
```

Disable exception handling

If your C++ compiler does not support exception handling you can disable the automatically generated exception handling code by re-defining try and catch as shown below.

```
#define try if(1)
#define catch(x) if(0)
```

These should be put in a header file and included in the test suite (see Includes).

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

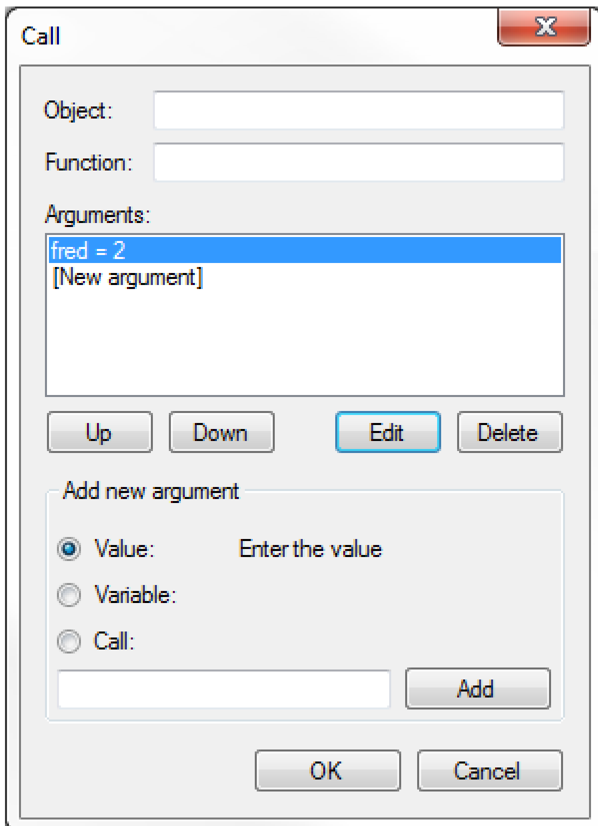
Assign or increment a value

There is no test case item specifically for assigning values to variables.

However this can be achieved by adding a Call test case item and leaving the **Object** and **Function** fields blank.

Add a Variable argument to the call and put the code to do the assignment in the Variable field.

An example is shown below:



The same technique can be used for increment and decrement operators.

Structure my tests

When starting out it can be difficult to judge whether to have big test cases with lots of tests, or lots of smaller test cases.

You could have one test case per function/method, or you might decide to group several tests together. Ultimately the answer depends on your own circumstances and preferences.

The main thing to bear in mind when making your choice is the scope of objects.

Each test case is a separate function/method in a separate source file, so any objects that are created are local to that test case.

Insert a copyright block

There are two ways to insert a copyright block; using the test suite (or test case) description, or using a stub file.

Note: You cannot insert a copyright block in the test suite header file using a stub file.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

Using the test suite (or test case) description

Simply add the copyright text to the test suite and test case descriptions.

Using a stub file

You need to create a stub file that contains the copyright text (as a comment).

To insert the copyright block in the test suite source file; add the stub file at the top level and check the Import box.

To insert the copyright block in a test case source file; add the stub file at the test case level and check the Import box.

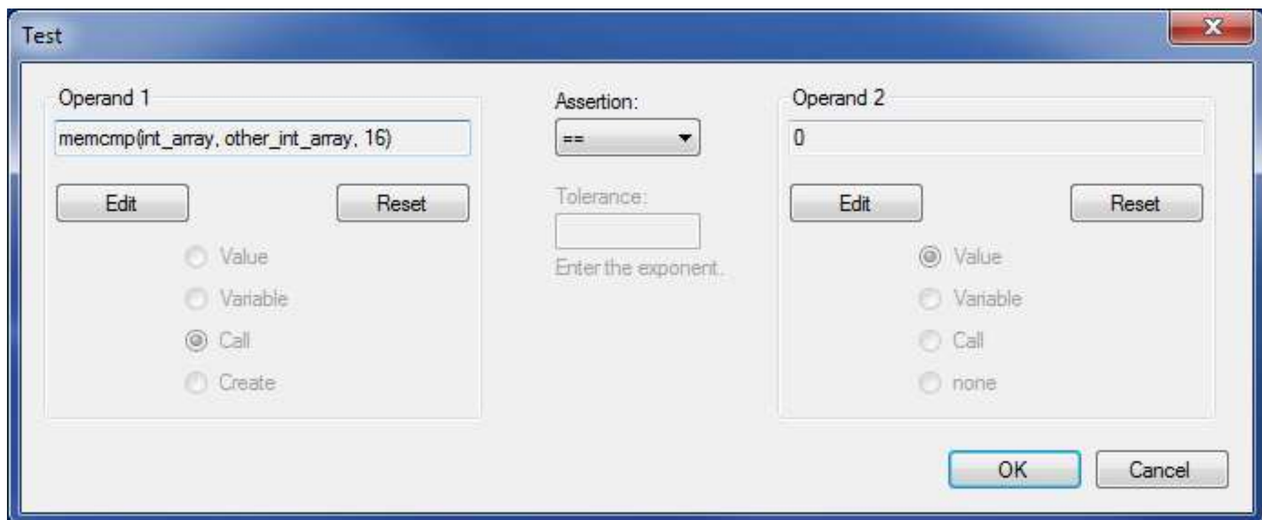
Compare two arrays

You can compare the values of the bytes in two arrays using the standard library function memcmp(). The steps to do this are as follows:

1. Create an Include for string.h.
2. Create a Test where the first Operand is a function call.
3. Set memcmp as the Function name.
4. Add three Arguments.
5. The first two arguments should specify the arrays you wish to compare.
6. The third argument should specify the size of the arrays (number of bytes).
7. Set the Assertion as '=='.
8. Set the second Operand to the value 0.

Note: The steps described above compare the arrays for equality. Testing for inequality can be achieved by changing the assertion to "!=". Refer to documentation on memcmp() for more information.

An example, which compares the first 16 bytes of two arrays called int_array and other_int_array, is shown below:



Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

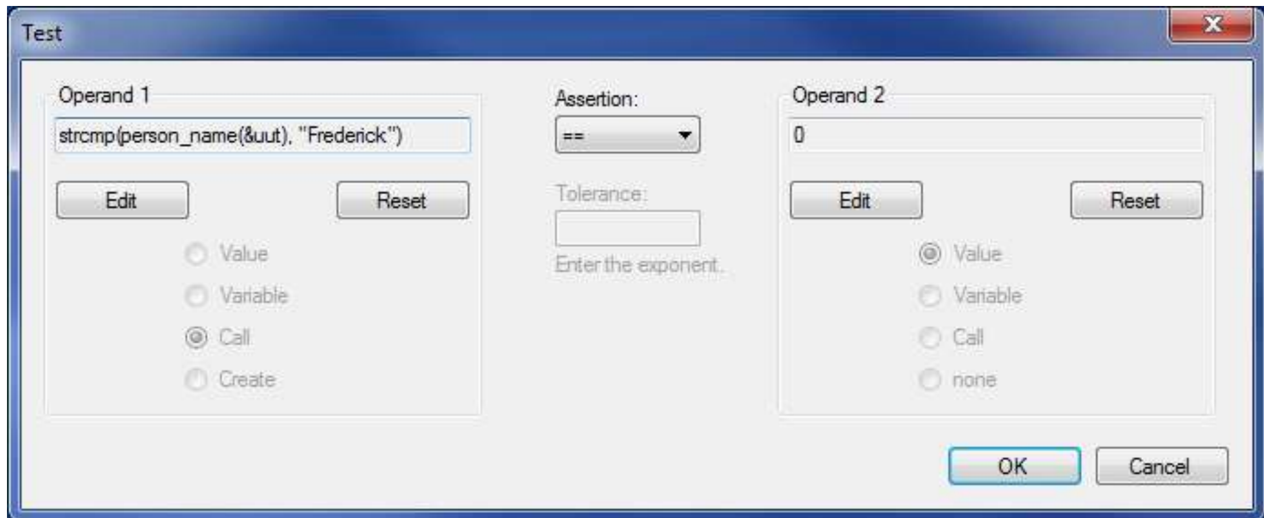
Compare two strings

You can compare two strings using the standard library function strcmp(). The steps to do this are as follows:

1. Create an Include for string.h.
2. Create a Test where the first Operand is a function call.
3. Set strcmp as the Function name.
4. Add two Arguments.
5. The arguments should specify the strings you wish to compare.
6. Set the Assertion as '=='.
7. Set the second Operand to the value 0.

Note: The steps described above compare the strings for equality. Testing for inequality can be achieved by changing the assertion to "!=", and more specific tests can be performed using the "<" and ">" assertions. Refer to documentation on strcmp() for more information.

An example, which compares the string returned by the function call person_name(&uut) with the string "Frederick", is shown below:



Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

13 Error messages

This section lists the error and warning messages you may encounter when using Embunit.

Error messages generally prevent you from continuing with the current operation, whereas warning messages do not.

Code generation errors are stored in the log file and displayed in the output box or on the command line, as appropriate.

Errors

Error opening file
The file does not contain a test suite

The test suite file you are trying to open does not conform to the Embunit XML DTD. Make sure you are using a file that was generated by Embunit.

Error opening file
Unrecognised file format

The test suite file you are trying to open does not contain XML. Make sure you are using a file that was generated by Embunit.

Embunit - Array size not specified
You must enter a value for the size of the array.

You have a Create item that specifies an array, but you have not specified the size of the array.

Embunit - Initializers cannot be converted to array format
Only 'Value' initializers can be used with arrays. Delete the affected initializers then try again.

You have a Create item and have already specified the initializers. You are now changing it to be an array, but at least one of the initializers is a function call or variable.

Embunit attempts to convert the initializers to a comma separated list of values.

Embunit - Operand 1 is missing
Operand 2 cannot be edited unless Operand 1 also exists. Select one of the Operand 1 radio buttons then try again.

You have a Test item with no operands, and are trying to edit Operand 2. You must define Operand 1 before you can edit Operand 2.

Embunit - Operand 1 is missing
All tests require at least one operand. Select one of the Operand 1 radio buttons.

You have a Test item with no operands, and have clicked the **OK** button. You must either define an operand, or click the **Cancel** button.

Embunit - Operand 2 is 'none'
The assertion you have chosen requires two operands.

The only Test item assertions that take a single operand are 'true', 'false', and 'throws' (optional second operand). All other assertions require two operands.

Embunit - Operand 2 is a function call
The exception type cannot be a function call. Reset Operand 2 and select a different radio button.

You have a Test item that checks that an exception is thrown. Operand 2 specifies the exception type,

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.

which cannot be a function call.

Operand 2 is optional in this case.

Embunit - Tolerance is not defined
Enter a value for the Tolerance.

You have a Test item that is comparing two floating point numbers for (non-)equivalence. You must specify a tolerance for the comparison.

If you want to check whether two floating point numbers are exactly the same use the '==' assertion.

Warnings

Embunit - Array initializers will be ignored
Arrays created on the heap cannot have initializers.

You have a Create item that specifies an array with initializers. Initializers can only be used with arrays that are created on the stack. When you generate the test harness the initializers will be ignored.

Embunit - Operand 2 is not 'none'
The assertion you have chosen only requires one operand. Operand 2 will be discarded when you save the file.

You have a Test item assertion that only requires Operand 1, but you have also defined Operand 2. Check that you have specified the correct assertion.

*You can delete Operand 2 by clicking the **Reset** button, and then selecting the 'none' radio button.*

Code generation errors

Could not find test_suite tag

The test suite file does not conform to the Embunit XML DTD. Make sure you are using a file that was generated by Embunit.

Could not create output path: <path>

A folder could not be created. Make sure the application has the right permissions to create the folder.

Could not create default output path: <path>

A folder could not be created. Make sure the application has the right permissions to create the folder.

Copyright © 2012 by Apollo Systems Contracts Ltd. All Rights Reserved.