Nunit V2.1 Beta 1 - 20 July, 2003

NUnit 2.1 Beta is finally out! It's taken a long time due to the large scope of the release and the pressure of our other commitments. Thanks again to all those who have helped.

This is the third major release of NUnit and the second since it was rewritten to take advantage of .NET custom attributes. Highlights release include support for the .NET framework 1.1, the ability to run test suites across multiple assemblies, improvements to the GUI interface, new command line switches for both GUI and Console runners, some degree of integration with Visual Studio and new TestFixtureSetup and TestFixtureTeardown attributes

With so many new features, we anticipate lively comment and discussion. We will continue with additional Beta iterations as needed leading to the final release.

Installation

By default the installation program places the all files in the directory:

```
C:\Program Files\Nunit V2.1
```

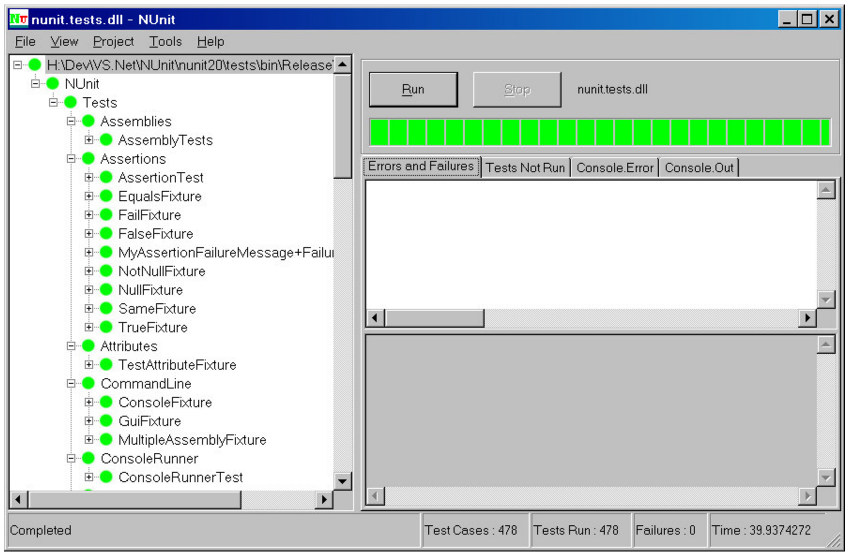
In the installation directory there are three sub-directories: bin, doc and src. The samples directory is located under src.

Start Menu

The installation program places a number of items in the Start menu. There is a shortcut to the Forms interface executable. (There is also a shortcut placed directly on the desktop). In addition to the executable file the menu items under Samples bring up a Visual Studio.NET solution file for the particular sample. The source shortcut brings up the Visual Studio.NET solution file for the entire project.

Installation Verification

You can verify that the installation has worked successfully by installing the program and running the tests that were used to build the program. Below is a screen shot of a successful run after loading nunit.tests.dll. There are 478 tests in all and they should all run successfully.



H:\Dev\VS.Net\NUnit\nunit20\tests\bin\Release
NUnit
Tests
Assemblies
AssemblyTests
Assertions
AssertionTest
EqualsFixture
FailFixture
FalseFixture
MyAssertionFailureMessage+Failu
NotNullFixture
NullFixture
SameFixture
TrueFixture
Attributes
TestAttributeFixture
CommandLine
ConsoleFixture
GuiFixture
MultipleAssemblyFixture
ConsoleRunner
ConsoleRunnerTest

Completed | Test Cases : 478 | Tests Run : 478 | Failures : 0 | Time : 39.9374272

Manual Installation

Those building NUnit 2.1 from source code may either build the install project and run the msi that is created or perform a manual installation by following these steps:

- Copy the following files to the target directory:
    - nunit.framework.dll
    - nunit.extensions.dll
    - nunit.uikit.dll
    - nunit.util.dll
    - nunit-console.exe
    - nunit-console.exe.config
    - nunit-gui.exe
    - nunit-gui.exe.config
- Run gacutil.exe to install nunit.framework.dll into the GAC.
- Create shortcuts as needed.
- If you want to be able to run the nunit tests, copy the following files to the same location as the others:
    - mock-assembly.dll
- **nonamespace-assembly.dll**
    - nunit.tests.dll
    - nunit.tests.dll.config
    - timing-tests.dll

  Note: Some of the nunit tests require access to the source to run successfully. The source tree should be copied into a directory named src with the same parent as the directory containing the assemblies so that the solution may be found at ..\src\nunit.sln.

Changes in V2.1

The following list includes most of the major changes from NUnit 2.0 to 2.1. **Text Like This** refers to more detail available in a later section of this document.

- The distributed version of NUnit 2.1 is now built using Visual Studio 2003 The solution and project files in this Beta can only be loaded by Visual Studio 2003. We may be able to make a Visual Studio 2002 version available to those who need it during the beta.
- NUnit can run against .NET framework versions 1.0 or 1.1. The supplied config file will allow use of either version, preferring version 1.1 if available. You may modify it to use only one version if you prefer.
- NUnit now supports loading and running tests across multiple assemblies in both the console and the GUI. This may be done on an adhoc basis or by creating NUnit test projects saved in files of type NUnit. For more information see **Multiple-Assembly Support** in this document.
- Information about one or more test assemblies may now be persisted as **NUnit Test Projects**. This is a file of type .nunit in XML format which supports the definition of multiple configurations each containing one or more assemblies.

- TestFixtureSetup and TestFixtureTearDown attributes are now recognized. Methods marked with these attributes will be run before and after the tests in a fixture. Unlike the standard Setup and TearDown attributes, these methods are invoked only once, before and after all the tests are run. See **Attributes** section below for more info.
- The Assertion class is now deprecated and has been replaced with the Assert class supporting a new set of static method names. See **Assert** section in this document.

Forms Interface

- There are a number of additional menu items. See **Main Menu** and **Context Menu**.
- The GUI interface now runs tests on a separate thread. This allows the tree display to update as execution proceeds. A Stop button now allows cancelling a test run. An option to cancel the run is also displayed if the user attempts to exit while a test is running.
- XML output from a test run may now be saved using the Tools | Save Results as XML… menu item. The format is the same as that used by the Console runner.
- The Tools | Options… menu item allows setting a number of options. See **Options Dialog** later in this document for more info.
- Automatic reloading of test assemblies may now be disabled. An alternate approach of reloading assemblies whenever a run begins is also available as an option.
- If Visual Studio support is enabled, Visual Studio project and solution files may be opened and Visual Studio projects may be added to an NUnit test project. Currently, C#, VB.NET, J# and managed C++ projects are supported. See **Visual Studio Support** below in this document for more information.
- When a Visual Studio or NUnit project is loaded, the user may switch between the available configurations causing the tests to be reloaded. A **Configuration Editor** allows adding, deleting or renaming configurations.
- The GUI allows creating and modifying **NUnit Test Projects** through the **Project Editor**.
- A Properties window is now availble to display information about any test in the tree. It may be pinned to allow quickly examining the results of different tests. See **Test Properties** in this document for more info.
- Window layout has been streamlined, eliminating one splitter and reducing the area above the tabs to the minimum size needed.
- The File | Recent Assemblies menu item is now called Recent Files since it may hold Visual Studio or NUnit project names. The user may now set the number of items kept in the list. Loading of the most recently used file may be disabled by a command line switch.
- Nested Classes are now shown in the TreeView using the format OuterClass+InnerClass. [Showing InnerClass under OuterClass was confusing when both classes contained tests.]
- The contents of the TreeView display are now sorted by name.
- Splitter positions are now saved in the registry and restored on startup.
- A View menu includes options for Expanding and Collapsing tree nodes, including Expand All, Collapse All, Expand Fixtures and Collapse Fixtures.

Console Interface

- The console interface command line parameters have been modified to support loading of multiple assemblies, Visual Studio projects and NUnit projects. Several new switches are available. See the **Command Line Parameters** section in this document for details.
- When run in debug mode, the console output is now sent to the Visual Studio Output window. Clicking on a test failure brings up the test file in the editor at location of the Assert.
- Formatting of console output has been improved so that redirected output may be examined conveniently in an editor that requires CRLF at the end of each line.

Upgrading from V2.0 to V2.1

You will need to recompile your test assemblies using the new nunit.framework assembly.

Because the old Assertion class is marked with the obsolete, you will receive warning messages until you have converted to the new Assert class.

Upgrading from V1.11 to V2.1

Upgrading from V1.11 to V2.1 requires a minimal amount of work. Since the framework still looks for test methods by name in addition to the attributes no test method will need to be modified to upgrade to the new version. The only source code change required is to remove the constructor with the string parameter in the class that inherits from TestCase. The only requirement is that you have a default constructor. If there was significant processing in the existing constructor than just move it to the default constructor. The only other change is to change the reference to nunit.framework.dll instead of the V1.x framework dll. This method has been marked using the Obsolete attribute so you will get warnings but they can be ignored for now!

Suite property

The existing Suite property will not be found by the new program. These must be changed to the “Suite” attribute for the test runners to find them. Another alternative is that these suites are no longer needed due to the automatic capability that is built in to the new version.

AssertionFailedError

If you have written code expecting the exception, AssertionFailedError this must be changed to AssertionException.

Multiple-Assembly Support

This release of NUnit allows loading suites of tests from multiple assemblies. A suite is constructed which contains as tests the root suite for each assembly. Tests are run and reported just as for a single assembly.

In the past, test writers have been able to rely on the current directory being set to the directory containing the assembly. For the purpose of compatibility, the current directory is set to the directory containing an assembly whenever any test from that assembly is being run.

Because some assemblies may rely on unmanaged dlls in the same directory, the current directory is also set to that of the assembly at the time the assembly is loaded. However, in cases where assemblies from multiple directories are loaded, this may not be sufficient and the user may need to place the directory containing the unmanaged dll on the path.

NUnit Test Projects

Running tests from multiple assemblies is facilitated by the use of NUnit test projects. These are files with the extension .nunit containing information about the assemblies to be loaded. The following is an example of a hypothetical test project file:

```
<NUnitProject>
  <Settings activeconfig="Debug"/>
  <Config name="Debug">
    <assembly path="LibraryCore\bin\Debug\Library.dll"/>
    <assembly path="LibraryUI\bin\Debug\LibraryUI.dll"/>
  </Config>
  <Config name="Release">
    <assembly path="LibraryCore\bin\Release\Library.dll"/>
    <assembly path="LibraryUI\bin\Release\LibraryUI.dll"/>
  </Config>
</NUnitProject>
```

The project contains two configurations, each of which contains two assemblies. The Debug configuration is currently active. By default, the assemblies will be loaded using the directory containing this file as the ApplicationBase. The PrivateBinPath will be set automatically to `LibraryCore\bin\Debug;LibraryUI\bin\Debug` or to the corresonding release path.

XML attributes are used to specify non-default values for the ApplicationBase, Configuration File and PrivateBinPath. The **Project Editor** may be used to create or modify NUnit projects.

Attributes

This section describes new or changed attributes supported in NUnit 2.1.

TestFixtureSetUp and TestFixtureTearDown

These two attributes are used inside a TestFixture to provide a set of functions that are performed once before any tests are run (TestFixtureSetUp) and after the last test (TestFixtureTearDown). A TestFixture can have only one TestFixtureSetUp method and only one TestFixtureTearDown method. If more than one of each type is defined the TestFixture will compile but it will not be run. Note that a fixture may have both a TestFixtureSetup and a Setup method and both a TestFixtureTearDown and a TearDown method.

TestFixtureSetup and TestFixtureTearDown are provided for use when performance considerations make it inconvenient to use Setup and TearDown, which are run for each test case. Users should consider that use of these attributes leads to greater interdependency between tests. In general, use Setup and TearDown in preference to these attributes.

TestFixtureSetUp/TestFixtureTearDown Example

This example shows the order in which the various methods are called.

```
[TestFixture]
public class TestFixtureSetUpAndTearDownTest
{
   [TestFixtureSetup]
   public void RunBeforeAllTests()
   {
      Console.WriteLine( "TestFixtureSetup" );
```

```
        }

        [TestFixtureTearDown]
        public void RunAfterAllTests()
        {
           Console.WriteLine( “TestFixtureTearDown” );
        }

        [Setup]
        public void RunBeforeEachTest()
        {
           Console.WriteLine( “Setup” );
        }

        [TestFixtureTearDown]
        public void RunAfterEachTest()
        {
           Console.WriteLine( “TearDown” );
        }
        [Test]
        public void Test1()
        {
           Console.WriteLine( “Test1” );
        }

        [Test]
        public void Test2()
        {
           Console.WriteLine( “Test2” );
        }
    }
```

The output from this test fixture would be:

```
TestFixtureSetup
Setup
Test1
TearDown
Setup
Test2
TearDown
TestFixtureTearDown
```

If Test2 were run alone the output would be:

```
TestFixtureSetup
Setup
Test2
TearDown
TestFixtureTearDown
```

Assert

The Assert class replaces the old Assertion class and contains static methods that replace each of the methods of that class:

```
Assert.IsTrue( bool );
Assert.IsFalse( bool );
Assert.IsNull( bool );
Assert.IsNotNull( bool );
Assert.AreSame( object, object )
Assert.AreEqual( object, object );
Assert.AreEqual( int, int );
Assert.AreEqual( float, float, float );
Assert.AreEqual( double, double, double );
Assert.Fail();
```

Each method also has overloads that take a message to be displayed as the last argument.

Command Line Parameters

Forms Interface

The forms interface may be run with or without the name of a file containing tests on the command line. If the program is started without any options it automatically loads the most recently loaded assembly.

Run without loading an Assembly

To supress loading of the most recent assembly, use the /noload switch:

```
nunit-gui.exe /noload
```

Specifying an Assembly

The other option is to specify an assembly or project file name on the command line. The following will start the forms interface with the assembly nunit.tests.dll:

```
nunit-gui.exe nunit.tests.dll
```

The following will start the forms interface loading the same assembly through it's Visual Studio project definition:

```
nunit-gui.exe nunit.tests.csproj
```

Assuming an NUnit test project has been created, the following will again load the nunit.tests.dll:

```
nunit-gui.exe nunit.tests.nunit
```

Specifying which Configuration to load

When loading a Visual Studio project or NUnit project, the first configuration found will be loaded by default. Usually this is Debug. The configuration loaded may be controlled by using the /config switch. The following will load the Release configuration of the nunit.tests.dll:

```
Nunit-gui.exe nunit.tests.csproj /config:Release
```

Specifying Multiple Assemblies

The Forms Interface does not provide for specifying more than one assembly on the command line. Multiple-assembly projects must be loaded by specifying the name of a Visual Studio solution file or an NUnit test project. See **Visual Studio Support** and **NUnit Test Projects** respectively.

Console Interface

The console interface has a few additional options compared to the forms interface. In addition, it supports entry of multiple assemblies on the command line. The console program must always specify an assembly or project file on the command line. The console interface always creates an

XML representation of the test results. This file by default is called TestResult.xml and is placed in the working directory.

Note: By default the nunit-console program is not added to your path. You must do this manually if this is the desired behavior.

Specifying an Assembly

The console program must always have an assembly or project specified. To run the tests contained in the nunit.tests.dll use the following command.

```
nunit-console.exe nunit.tests.dll
```

To run the tests in nunit.tests.dll through the Visual Studio project, use:

```
nunit-console.exe nunit.tests.csproj
```

To run the same tests through an NUnit test project you have defined, use:

```
nunit-console.exe nunit.tests.nunit
```

Specifying an Assembly and a Fixture

When specifying a fixture you must give the full name of the test fixture along with the containing assembly. For example, to run the NUnit.Tests.AssertionTests in the nunit.tests.dll assembly use the following command.

```
nunit-console /fixture:NUnit.Tests.AssertionTests  nunit.tests.dll
```

The /fixture option may be used with Visual Studio or NUnit projects as well.

Specifying the XML file name

As stated above the console program always creates an XML representation of the test results. To change the name to console-test.xml use the following command line option,

```
nunit-console /assembly:nunit.tests.dll /xml:console-test.xml
```

Specifying the Transform file

The console interface uses XSLT to transform the test results from the XML file to what is printed to the screen when the program executes. The console interface has a default transformation that is part of the executable. To specify your own transformation named myTransform.xslt use the following command line option.

```
nunit-console nunit.tests.dll /transform:myTransform.xslt
```

Note: For additional information see the XML schema for the test results. This file is in the same directory as the executable and is called Results.xsd. The default transform Summary.xslt is located in the nunit-console source directory.

Specifying which Configuration to run

When running tests from a Visual Studio project or NUnit project, the first configuration found will be loaded by default. Usually this is Debug. The configuration loaded may be controlled by using the /config switch. The following will load and run tests for the Release configuration of the nunit.tests.dll:

```
Nunit-console.exe nunit.tests.csproj /config:Release
```

Specifying Multiple Assemblies

You may run tests from multiple assemblies in one run using the console interface even if you have not defined an NUnit test project file. The following command would run a suite of tests contained in assemby1.dll, assembly2.dll and assembly3.dll:

```
nunit-console.exe assembly1.dll assembly2.dll assembly3.dll
```

Note: You may specify multiple assemblies but not multiple NUnit or Visual Studio projects on the command line. Further, you may not specify an NUnit or Visual Studio project together with a list of assemblies.

Note: The /fixture option may be used with multiple assemblies but it's obviously more efficient to simply specify the assembly that contains the fixture.

Additional Switches

- **/wait –** Require user input to exit program.
- **/xmlconsole –** Display XML output to the console.
- **/nologo –** Suppress display of the NUnit informational message.
- **/help –** Display help for the command

Main Menu

File

New Project…

Closes any open project, prompting the user to save it if it has been changed and then opens a FileSave dialog to allow selecting the name and location of the new project.

Open…

Closes any open project, prompting the user to save it if it has been changed and then opens a FileOpen dialog to allow selecting the name and location of an assembly, a test project or (if Visual Studio support is enabled) a Visual Studio project.

Close

Closes any open project, prompting the user to save it if it has been changed.

Save

Saves the currently open project. Opens the Save As dialog if this is the first time the project is being saved.

Save As…

Opens a FileSave dialog to allow specifying the name and location to which the project should be saved.

Reload

Reloads the currently loaded project using the latest saved copy of each assembly.

Recent Files…

Displays a list of recently opened files from which the user is able to select one for opening.

Exit

Closes and exits the application. If a test is running, the user is given the opportunity to cancel it and or to allow it to continue. If the open project has any pending changes, the user is given the opportunity to save it.

View

Expand

Expands the currently selected tree node.

Collapse

Collapses the currently selected tree node.

Expand All

Expands all nodes of the tree.

Collapse All

Collapses all nodes in the tree to the root.

Expand Fixtures

Expands all the tree nodes representing fixtures.

Collapse Fixtures

Collapses all the tree nodes representing fixtures.

Properties…

Displays the Properties Dialog for the currently selected test.

Project

Configurations (popup)

Debug, Release, etc.

Selects one of the available configurations

Add…

Opens the Add Configuration Dialog, which allows entry of the name of the new configuration and specifying an existing configuration to use as a template.

Edit…

Opens the **Configuration Editor**

Add Assembly…

Displays a FileOpen dialog to allow selecting an assembly to be added to the active configuration of the currently open project.

Add VS Project…

Only available if Visual Studio Support is enabled. Displays a FileOpen dialog to allows selecting a Visual Studio project to be added to the currently open project. Entries are added for each configuration specified in the VS project, creating new configurations in the test project if necessary.

Edit…

Opens the **Project Editor**.

Tools

Save Results as XML…

Opens a FileSave Dialog for saving the test results as an XML file.

Options

Displays the **Options Dialog**.

Help

Help

Not yet available.

About NUnit…

Displays info about your version of NUnit and a link to the nunit.org site.

Context Menu

The context menu displays when one of the tree notes is right-clicked.

Run

Runs the selected test. Disabled if a test is running.

Expand

Expands the selected test node – invisible if the node is expanded or has no children.

Collapse

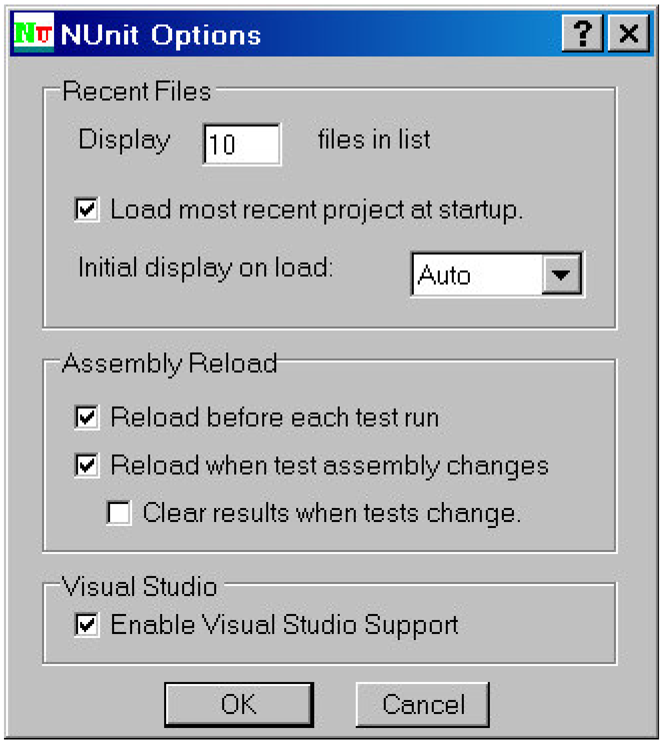
Collapses the selected test node – invisible if the node is collapsed or has no children.

Properties

Displays the **Test Properties** for the selected test node.

Options Dialog

The Options Dialog is displayed using the Tools | Options menu item and allows the user to control some aspects of NUnit's operation:



NUnit Options

Recent Files

Display 10 files in list

☑ Load most recent project at startup.

Initial display on load: Auto

Assembly Reload

☑ Reload before each test run

☑ Reload when test assembly changes

☐ Clear results when tests change.

Visual Studio

☑ Enable Visual Studio Support

Recent FIles

- The text box allows the user to choose the number of entries to display in the recent files list.
- If "Load most recent project at startup" is checked, the GUI will load the last file opened unless it is run with a specific filename or with the /noload parameter.
- The list box allows selecting the degree of expansion of the tree when tests are loaded:

  Expand – expands all tests

  Collapse – collapses all tests

  Hide Tests – expands all suites except for the fixtures themselves.

  Auto – selects one of the above based on the space available for the tree display.

Assembly Reload

- If "Reload before each test run" is checked, a reload will occur whenever the run button is pressed whether the assemblies appear to have changed or not.
- If "Reload when test assembly changes" is checked, assemblies are watched for any change and an automatic reload is initiated.
- If "Clear results when tests change" is checked, reloading an assembly will reinitialize all test nodes in the tree (grey display) – if it is not checked, result information for tests that do not seem to have changed will be retained.

Visual Studio

- If "Enable Visual Studio Support"is checked, the user will be able to open Visual Studio projects and solutions and add Visual Studio projects to existing test projects.

Visual Studio Support

Visual Studio support in this release is a sort of "poor man's integration." We have implemented a number of features while avoiding any that would require using an Addin or otherwise interacting with the Visual Studio extensibility model.

Using Console Interace to Debug Applications

When the nunit-console program is run in debug mode under Visual Studio, it detects that it is running in this mode and sends output to the Visual Studio output window. Output is formatted so that double clicking any error or failure entries opens the appropriate test file at the location where the failure was detected.

Opening Visual Studio Projects

When Visual Studio support is enabled, the File Open dialog displays the following supported Visual Studio project types: C#, VB.Net, J# and C++. The project file is read and the configurations and output assembly locations are identified. Since the project files do not contain information about the most recently opened configuration, the output assembly for the first configuration found (usually Debug) is loaded in the GUI. The tree shows the projedt as the top-level node with the assembly shown as its descendant.

When tests are run for a Visual studio project, they run just as if the output assembly had been loaded with one exception. The default location for the config file is the directory containing the project file and it's default name is the same as the project file with an extension of .config.

For example, the following command would load the tests in the nunit.tests assembly using the configuration file nunit.tests.dll.config located in the same directory as the dll.

```
nunit-gui.exe nunit.tests.dll
```

On the other hand, the following command would load the tests using the configuration file nunit.tests.config located in the same directory as the csproj file.

```
nunit-gui.exe nunit.tests.csproj
```

The same consideration applies to running tests using the console runner.

Opening Visual Studio Solutions

When Visual Studio support is enabled, solution files may be opened as well. All the output assemblies from contained projects of the types supported will be loaded in the tree. In the case where all contained projects are located in the subdirectories beneath the solution, it will be possible to load and run tests using this method directly.

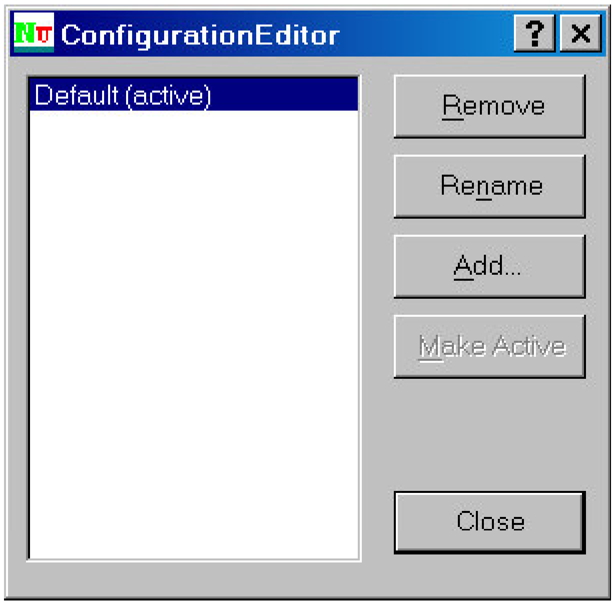
When a solution contains projects located elsewhere in the file system, it may not be possible to run the tests – although the solution will generally load without problem. In this case, the Project Editor should be use to modify and save the NUnit test project so that there is all referenced assemblies are located in or beneath the application base directory.

Adding Visual Studio Projects to the Open Test Project

When Visual Studio support is enabled, the Project menu contains an active entry to add a VS project to the loaded project. The output assembly will be added for each of the configurations specified in the VS project.

Configuration Editor

The Configuration Editor is displayed using the Project | Configuratino | Edit… menu item and supports the following operations:



Remove

Remove the selected configuration. If it was the active config, then the next one in the list is made active.

Rename

Rename the selected configuration.

Add…

Add a new configuration. The Add Configuration dialog allows specifying an existing configuration to use as a template.

Make Active

Makes the selected configuration active.

Close

Exits the configuration editor

Test Properties

The test properties dialog is displayed using either the View | Properties menu item on the main menu or the Properties item on the context menu. It shows information about the test and – if it has been run – about the results. The dialog contains a “pin” button in the upper right corner which causes it to remain open as the user clicks on different tests.

Test Tab

This tab gives general information about the test itself.

Full Name

The fully qualified name of the test. In the image below, a tooltip is displaying the complete name, which did not fit in the label.

Test Count

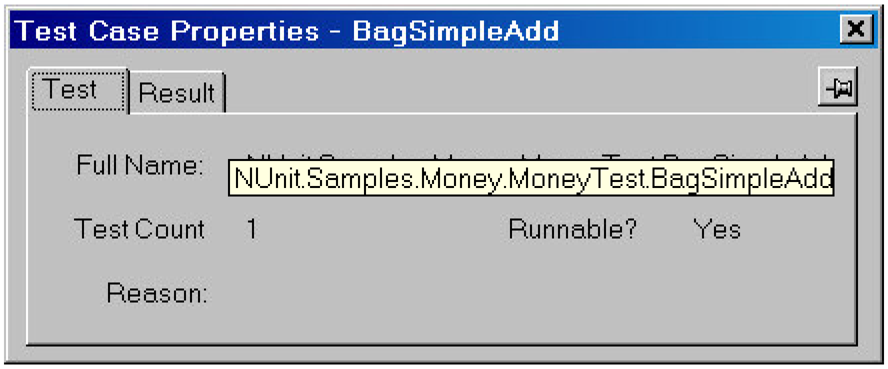
The number of test cases contained directly or indirectly in this test.

Runnable?

Indicates whether the test is able to be run or not.

Reason

If the test is not Runnable, indicates the reason. This can be the reason given on an ignore attribute or an error such as trying to apply the Test attribute to a private method.



Full Name: NUnit.Samples.Money.MoneyTest.BagSimpleAdd

Test Count 1 Runnable? Yes

Reason:

Result Tab

This tab is only visible if the user has run the selected test. It gives information about the success or failure of the test.

Success / Failure

Label indicating whether the test passed or failed.

Time

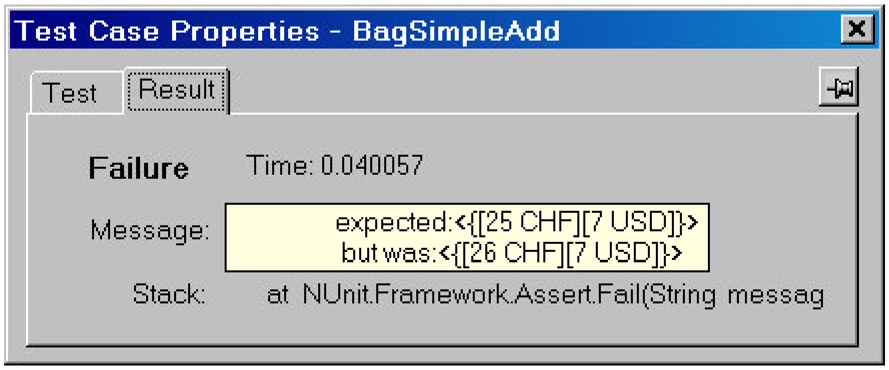
The elapsed time in seconds to run the test.

Message

If the test failed, the error message is shown here. In the image below, a tooltip is displaying the full text of the message.

Stack

If the test failed, the stack trace at the point of failure. In the image below, the stack display is truncated but could be displayed in a tooltip by the user hovering with the mouse.



Failure Time: 0.040057

Message: expected:<{[25 CHF][7 USD]}> but was:<{[26 CHF][7 USD]}>

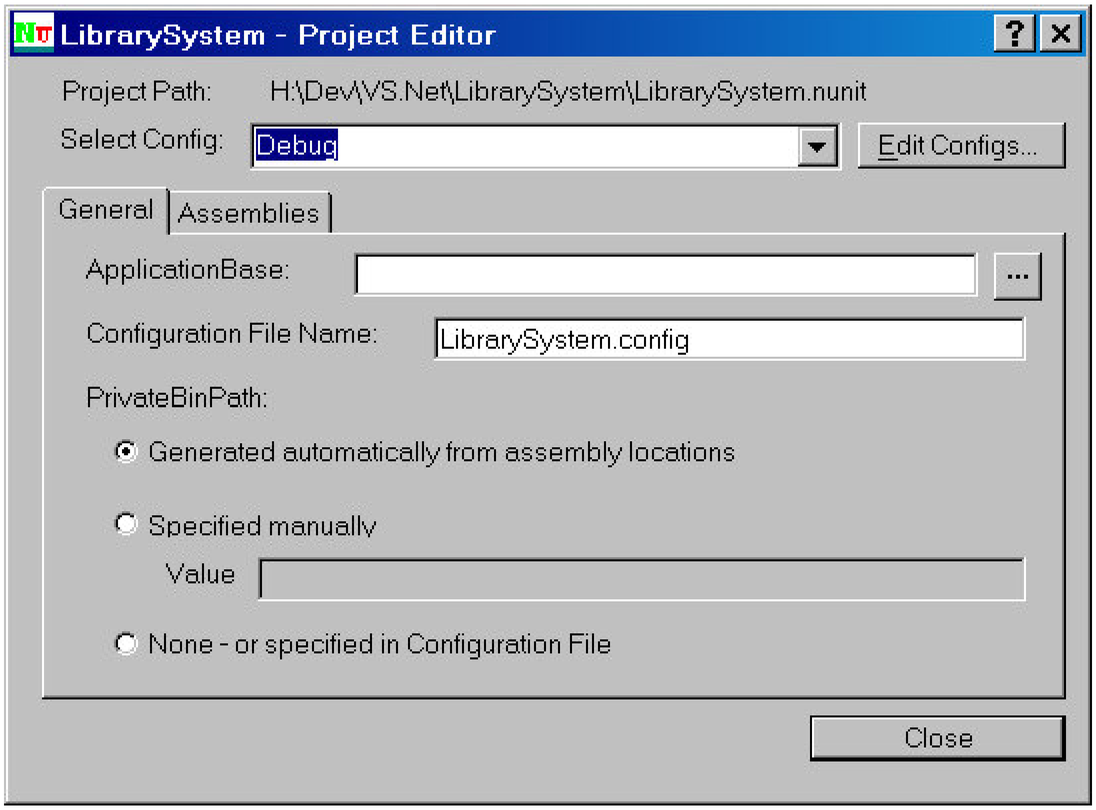
Stack: at NUnit.Framework.Assert.Fail(String messag

Project Editor

The Project Editor is displayed through the Project | Edit menu item and allows creating or modifying NUnit test projects. It should be noted that a Test Project is active whenever any tests have been loaded, even if no project was explicitly created or referenced. In the case of an assembly being loaded, an internal wrapper project is created. This allows the user to change settings and save the project directly without needing to perform any extra steps. The editor consists of a common area and two tabs, as seen in the image below.

Common Area

The common area of the Project Editor contains a label showing the full path to the project file, a dropdown combo box allowing selection of the configuration to be edited and a button, which opens the **Configuration Editor**, discussed elsewhere in this document.



LibrarySystem - Project Editor

Project Path: H:\Dev\VS.Net\LibrarySystem\LibrarySystem.nunit

Select Config: Debug

ApplicationBase:

Configuration File Name: LibrarySystem.config

PrivateBinPath:

[✓ Generated automatically from assembly locations]

Specified manually

Value

None - or specified in Configuration File

General Tab

The General tab allows setting a number of options pertaining to the selected configuration, all of which will be stored in the NUnit project file as attributes of the `<config>` xml node.

ApplicationBase

The ApplicationBase defaults to the directory containing the project file. The user may set it to another path provided that path is contained under the project file directory.

Configuration File Name

The configuration file defaults to the name of the test project with the extension changed from .nunit to .config. The user may substitute another name.

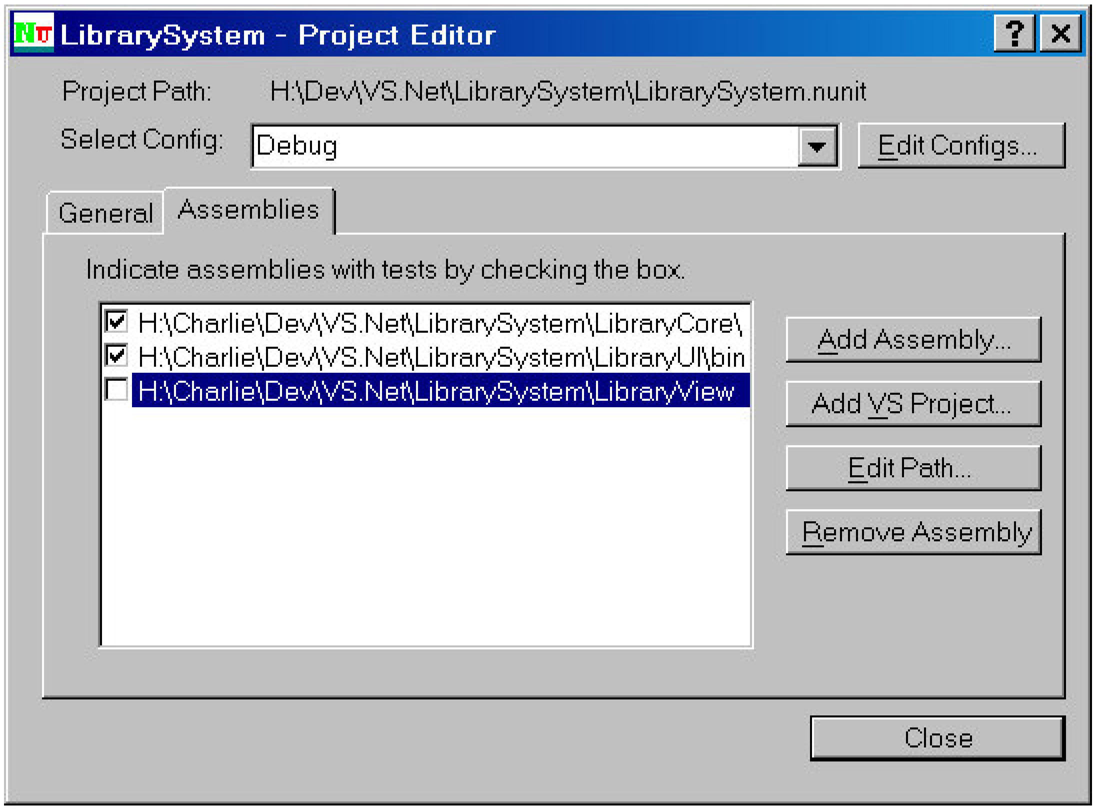
PrivateBinPath

By default, the PrivateBinPath is generated from the assembly locations specified on the Assemblies Tab. For those applications requiring a different level of control, it may be specified manually or using this editor or placed in the configuration file.

Results Tab

The results tab contains the list of assemblies that form part of this test project. A checkbox is used to indicate which assemblies contain tests. This allows use of the automatically generated private bin path by including assemblies that the tests depend upon but leaving the box unchecked. The GUI does not attempt to load these unchecked assemblies but does use their locations in generating the PrivateBinPath and also watches them for changes, if automatic reloading of tests is enabled.

Note: Although the dialog shows the location of assemblies as absolute paths, they are always persisted in the NUnit project file as paths relative to the application base. This allows moving projects as a whole to a different directory location.



Project Path: H:\Dev\VS.Net\LibrarySystem\LibrarySystem.nunit

Select Config: Debug

Indicate assemblies with tests by checking the box.

- [x] H:\Charlie\Dev\VS.Net\LibrarySystem\LibraryCore\
- [x] H:\Charlie\Dev\VS.Net\LibrarySystem\LibraryUI\bin
- [ ] H:\Charlie\Dev\VS.Net\LibrarySystem\LibraryView

Add Assembly

Opens a dialog allowing adding an assembly to this configuration.

Add VS Project

Opens a dialog allowing selction of a VS project to be added to this project.

Edit Path

Opens a dialog allowing the user to change the path to the selected assembly in this configuration.

Remove Assembly

Removes the selected assembly from this configuration.

Samples

- C# - This sample demonstrates 4 failing unit tests and 1 ignored test written in C#.
- VB.NET - This sample demonstrates what should be four failing unit tests and 1 ignored test.
- Managed C++ - This is the same example as the others with 4 failing unit tests. This is correct when compiled in Debug mode. In Release mode the divide by zero test succeeds.
- Visual J# - This has the same tests as the other samples, except written in Visual J#. If you have not installed this when you go to open the Visual Studio solution file it will inform you that the file solution file cannot be loaded.
- Money - This is C# version of the money example which is found in most xUnit dimplementations. Thanks Kent.
- Money-port - This is an example of the minimum amount of work that is needed to upgrade fromprevious versions of NUnit to this version.

Code contribution philosophy

First and foremost we have done our best to insure that this program was developed test-first. As such we will accept no code changes without associated tests. As for bug fixes we plan to follow the procedure that we write a failing unit test first and then fix it. In this fashion the number of tests that we have will grow over time. Lastly, if a change breaks an existing test it is the responsibility of the person making the change to first understand the ramification of the change and either fix the test or alter the modification that caused the problem. That said, if you are interested, so are we, please help make Nunit the best xUnit tool in any language.

License

Developers

- James Newkirk (jim@nunit.org)
- Michael C. Two (mike2@nunit.org)
- Alexei A. Vorontsov (alexei@nunit.org)
- Philip A. Craig (philip@nunit.org)
- Charlie Poole (cpoole@pooleconsulting.com)

V2.1 Detailed Corrections

Changes from V2.0 to Beta 1

Feature Requests Implemented

- 597609 – Support a cancel/stop operation
- 606393 – Maintain results in gui when reloading
- 603449 – Option to disable assembly watching
- 597626 – /wait gone from NunitConsole.exe
- 600119 – Per-test statistics in nunit-gui
- 616931 – Sort order of TestFixtures
- 594486 – Loading Switch
- 599120 – Show help when invalid GuiRunner options
- 598814 – Add TestFixtureSetup/TearDown
- 660750 – A class SetUp which runs each time Run is clicked
- 623090 – Warnings if fixtures not public
- 609643 – Compound structure
- 614602 – Load multiple test dlls in nunit-gui
- 660751 – Do not expand Tree by default
- 629138 – Add Class Setup/TearDown
- 516161 – Test Project Files
- 629136 – Remarks for TestFixture and Test Attributes
- 677192 – Recursive assembly load by reference
- 587855 – Need option to select default test case
- 660767 – Assert should indicate which source line
- 669203 – Save and restore splitter positions
- 660761 – Add shortcut key to Recently Used list
- 648492 – NUnit doesn't display in reference list
- 676195 – Increase the Number of Recent Assemblies
- 600182 – REPOST: Output to IDE Output Window
- 660764 – Set focus, when StdOut or Errors are reported
- 660745 – NUnit GUI should nest nested classes
- 757160 – Add Registry key to show assemblies in .NET Reference dialog

- 755787 – Realtime Tree updates
- 755791 – Stop button in GUI

Bugs Fixed or Closed

- 610910 – UI Problems while Test is running
- 624007 – Object has been disconnected
- 624935 – Splitter messes up Run Area
- 624943 – Disabled Run Button
- 610906 – Running Test Domain Unloaded
- 615750 – Test of unsafe C# code stack ovflw excpt
- 619823 – crash when specifying invalid assembly
- 630663 – Documentation
- 618673 – Installer fails with Interrupted.. error
- 640533 – NUnit crashes when assembly is updated
- 644877 – Assembly Load failure
- 599145 – Failed check of sign. For MS.VisualBasic
- 609440 – I have an error when I run nunit v2 rc2
- 618945 – installer fails
- 665558 – ConfigurationSettings not available
- 661571 – nunit task from nant = failure
- 660770 – ExpectedException doesn't work as documented
- 656401 – Missing dll reference in documentation
- 648152 – Click on failed test then STDOUT tab
- 668406 – ConfigurationSettings w/ networked assemblies
- 699661 – Nunit-gui exits when pressing space bar
- 699640 – Tests run in nunit-console fails
- 703096 – Run All Tests in All Assemblies
- 667664 – GUI updates
- 646933 – Assembly Load failure: possible solution
- 699641 – TestResult.xml written in wrong directory
- 675968 – Another open database bug – VS.NET 2003
- 706643 – Switching to StandardOut pane with too much text fails.
- 707671 – System.Net.IPAddress.Parse – Invalid Pinvoke metadata format
- 713681 – [STAThread] Attribute Missing in Console Runner

- 713388 – (BUG) Dependent assemblies aren’t reloaded
- 720225 – AssertEquals() incorrectly compares decimals in .NET 1.1
- 722037 – Infinite loop causes test harness to lock
- 730822 – Setup() should fail if an exception is thrown during execution
- 731422 – AssertEquals fails with two Decimals in .NET v1.1
- 731258 – suite not recognized
- 738425 – NUnit-console.exe runs tests from a MTA
- 730870 – Setup() should fail if an exception is thrown during execution
- 640467 – Newline not rendered in GUI
- 671642 – Multiline ToolTip messages improperly displayed
- 735111 – assembly loading fails if assembly name contains a dot char
- 743442 – timing-tests won’t compile
- 735138 – nunit.console
- 742163 – NUnit.Extensions.RepeatedTest.cs won’t compile
- 770942 – AppSettings not loaded when using nunit-console.exe
- 744649 – Lengthy COM calls cause failures in Console.WriteLine