

NUnit 2.2 Supplementary Documentation

With NUnit 2.2, much of the documentation is now distributed in HTML form. However, not all the existing documentation has been converted, so this file is distributed to fill in the gap until the process is complete.

Multiple-Assembly Support

Since version 2.1, NUnit has allowed loading suites of tests from multiple assemblies. A top-level suite is constructed, which contains 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 corresponding 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.

Configuration Files

NUnit uses configuration files for the test runner executable – either nunit-console.exe or nunit-gui.exe – as well as for the tests being run. Only settings that pertain to NUnit itself should be in the nunit-console.exe.config and nunit-gui.exe.config, while those that pertain to your own application and tests should be in a separate configuration file.

NUnit Configuration Files

The main purpose of the nunit-console and nunit-gui config files is to allow NUnit to run with various versions of the .NET framework. NUnit 2.2 is built using version 1.1 of the framework, but can be made to run against version 1.0 or 2.0. As delivered, the config file selects the first installed version of .NET it finds in this order: 1.1, 2.0, 1.0. You should edit the config files if you prefer a different order. Your tests will run using the same version of the framework that is used to run NUnit itself.

Test Configuration File

When a configuration file is used to provide settings or to control the environment in which a test is run, specific naming conventions must be followed.

If a single assembly is being loaded, then the configuration file is given the name of the assembly file with the config extension. For example, the configuration file used to run nunit.tests.dll must be named nunit.tests.dll.config and located in the same directory as the dll.

If an NUnit project is being loaded, the configuration file uses the name of the project file with the extension *changed* to config. For example, the project AllTests.nunit would require a configuration file named AllTests.config, located in the same directory as AllTests.nunit. The same rule is followed when loading Visual Studio projects or solutions.

Generally, you should be able to simply copy your application config file and rename it as described above. However, see the next section if you wish to run tests compiled for an earlier version of NUnit.

NUnit 2.0 or 2.1 Tests Under NUnit 2.2

To run NUnit tests built against the NUnit 2.0 or 2.1 framework without recompiling them, you must modify your test configuration file to cause the tests to use version 2.2 of the framework dll. The files `nunit20under22.config` and `nunit21under22.config` are provided as templates for this purpose. They are located in the src directory.

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.

Running From Within Visual Studio

The most convenient way to do this is to set up a custom tool entry specifying the path to NUnit-Gui as the command. For a C# project, you may wish to use `$(TargetPath)` for the arguments and `$(TargetDir)` for the initial directory. If you would like to debug your tests, use the Visual Studio Debug | Processes... menu item to attach to nunit-gui.exe after starting it and set breakpoints in your test code as desired before running the tests.

Using Console Interface 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 project 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.

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 allow 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.

Exception Details...

Displays detailed information about the last exception.

Options

Displays the **Options Dialog**.

Help

Help

Displays the HTML help information.

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:

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. This item is disabled on Windows 98 or ME.

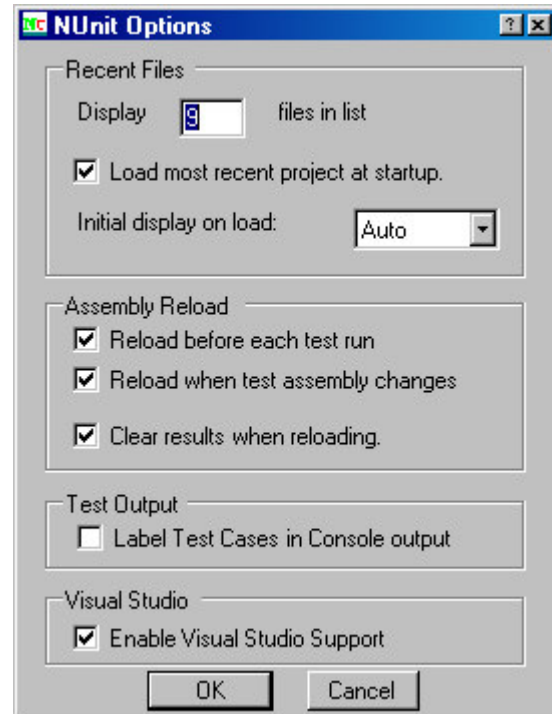
If "Clear results when tests change" is checked, an automatic or manual reload will reinitialize all test nodes in the tree (grey display) – if it is not checked, result information for tests that do not seem to have changed will be retained.

Test Output

Check this to precede text in the output window with the name of the test that produced it.

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.



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

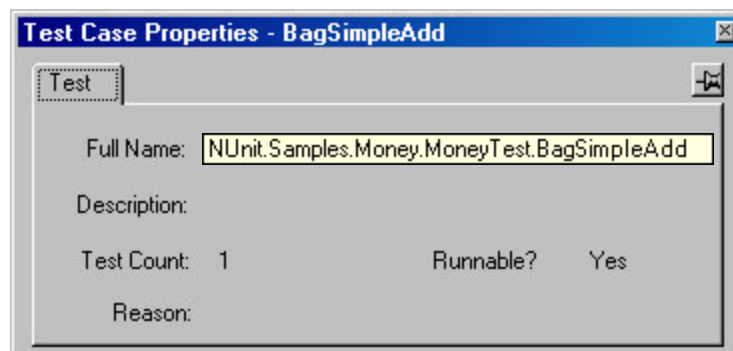
The number of test cases contained directly or indirectly in this test.

Runnable?

Indicates whether the test can 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.



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

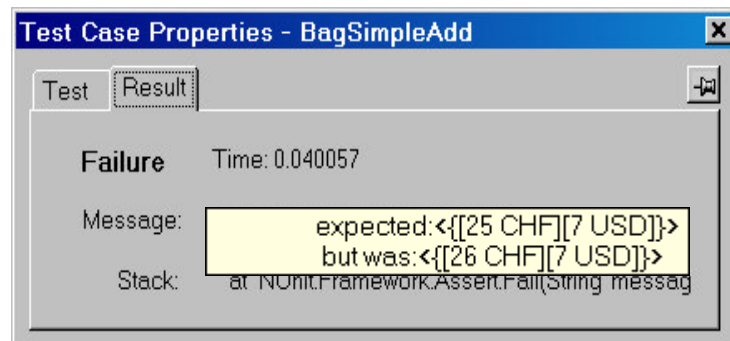
The elapsed time in seconds to run the test.

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.



Configuration Editor

The Configuration Editor is displayed using the Project | Configuration | 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...

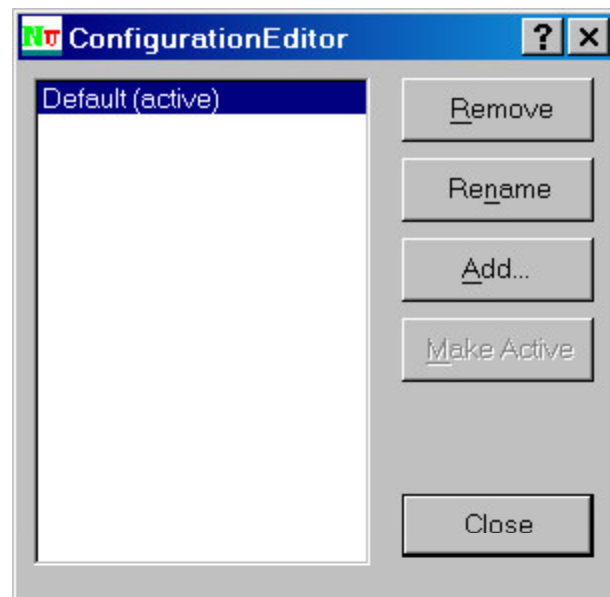
Add a new configuration. The Add Configuration dialog allows specifying an existing configuration to use as a template.

Make Active

Makes the selected configuration active.

Close

Exits the configuration editor

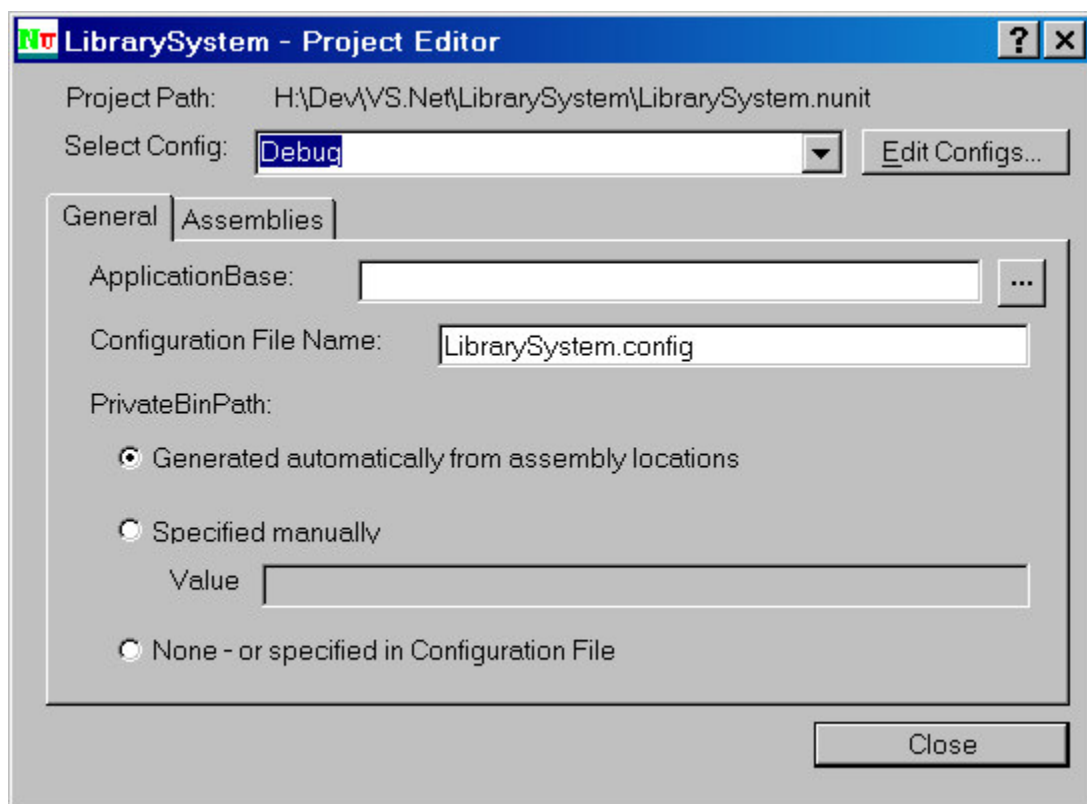


Project Editor

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.



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.

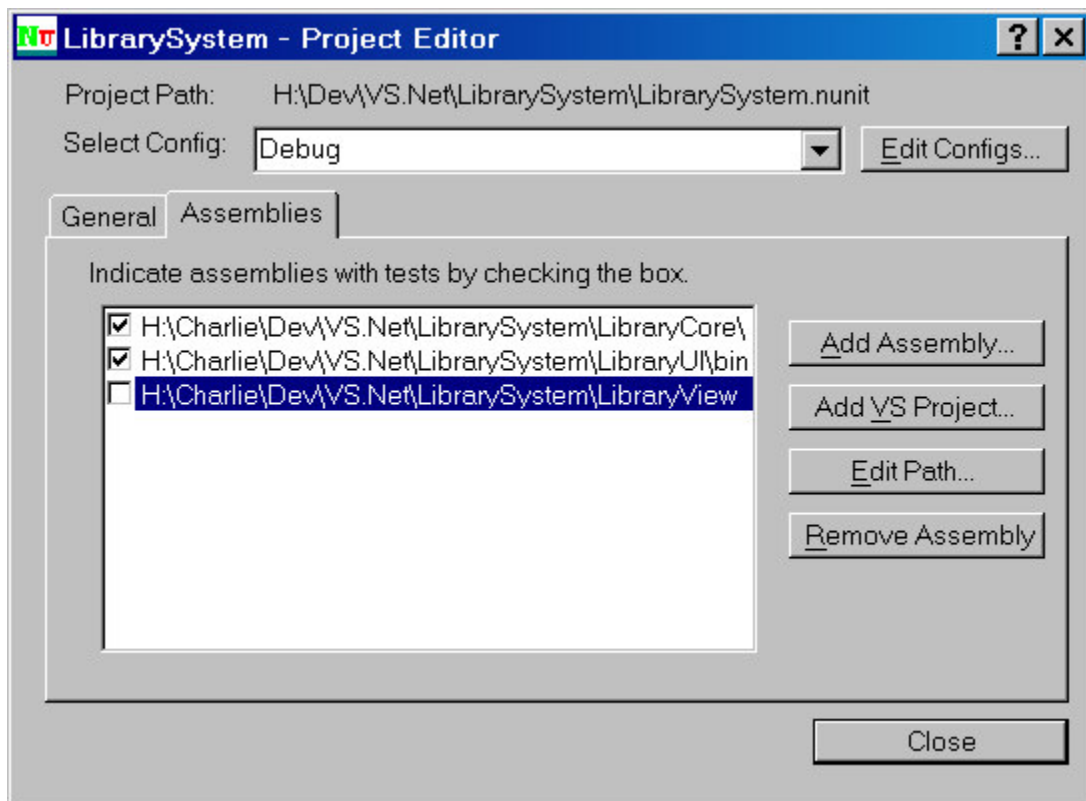
PrivateBinPath

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.



Add Assembly

Opens a dialog allowing adding an assembly to this configuration.

Add VS Project

Opens a dialog allowing selection 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.