Clover.NET User Manual

Version 1.2.2053

1. Introduction

1.1. Introduction

Clover.NET is a code coverage tool to help you understand and improve the quality of your .NET software testing. This, in turn, allows you to deliver quality .NET software and updates with confidence and reduced costs.

1.1.1. Getting Started

To install Clover.NET, please refer to the Installation Guide. This covers installation of the command-line tools, the Visual Studio.NET 2003 plugin, and the NAnt tasks

If you are having problems and need support please refer to our Frequently Asked Questions section covering common questions with links to our support forums.

1.1.2. System Requirements

Clover.NET requires Version 1.1 of the .NET Framework. Instrumentation is currently supported on Windows operating systems. The Visual Studio .NET plugin requires Visual Studio .NET 2003. At this time Clover.NET only supports the C# programming language.

All necessary third party libraries are packaged with Clover.NET. The Clover.NET runtime component has no external dependencies.

1.1.3. Acknowledgments

Clover.NET makes use of the following excellent 3rd party libraries.

Elkhound	A GLR parser generator
NVelocity	Templating engine used for HTML report generation.
Apache Logging Log4Net	Logging system

The Clover.NET Visual Studio plugin contains portions Copyright 2004, Microsoft Corporation. All rights reserved

2. Code Coverage

2.1. Code Coverage

2.1.1. What is Code Coverage?

Code coverage measurement simply determines those statements in a body of code have been executed through a test run and those which have not. In general, a code coverage system collects information about the running program and then combines that with source information to generate a report on test suite's code coverage.

Code coverage is part of a feedback loop in the development process. As tests are developed, code coverage highlights aspects of the code which may not be adequately tested and which require additional testing. This loop will continue until coverage meets some specified target.

2.1.2. Why Measure Code Coverage?

It is well understood that unit testing improves the quality and predictability of your software releases. Do you know, however, how well your unit tests actually test your code? How many tests are enough? Do you need more tests? These are the questions code coverage measurement seeks to answer.

Coverage measurement also helps to avoid test entropy. As your code goes through multiple release cycles, there can be a tendency for unit tests to atrophy. As new code is added, it may not meet the same testing standards you put in place when the project was first released. Measuring code coverage can keep your testing up to the standards you require. You can be confident that when you go into production there will be minimal problems because you know the code not only passes its tests but that it is well tested.

In summary, we measure code coverage for the following reasons:

- To know how well our tests actually test our code
- To know whether we have enough testing in place
- To maintain the test quality over the lifecycle of a project

Code coverage is not a panacea. Coverage generally follows an 80-20 rule. Increasing coverage values becomes difficult with new tests delivering less and less incrementally. If you follow defensive programming principles where failure conditions are often checked at many levels in your software, some code can be very difficult to reach with practical levels of testing. Coverage measurement is not a replacement for good code review and good programming practices.

In general you should adopt a sensible coverage target and aim for even coverage across all of the modules that make up your code. Relying on a single overall coverage figure can hide large gaps in coverage.

2.1.3. How Code Coverage Works

There are many approaches to code coverage measurement. Broadly there are three approaches, which may be used in combination:

Source Code Instrumentation	This approach adds instrumentation statements to the source code and compiles the code with the normal compile tool chain to produce an instrumented assembly.
Intermediate code Instrumentation	Here the compiled assemblies are instrumented by adding new intermediate language statements to the code and a new instrumented assembly generated.
Runtime Information collection	This approach collects information from the runtime environment as the code executes to determine coverage information

As the code under test executes, code coverage systems collect information about which statements have been executed. This information is then used as the basis of reports. In addition to these basic mechanisms, coverage approaches vary on what forms of coverage information they collect. There are many forms of coverage beyond basic statement coverage including conditional coverage, method entry and path coverage.

3. Installation

3.1. Installation Guide

3.1.1. Command-Line

The Clover.NET command line tools are packaged as a Windows MSI installer. Run this installer by double clicking on the Installer file, CloverNET.msi. The installation process is straight forward. You may change the default location to suit your local conventions or simply accept the default.

Once installed, the command-line tools are available directly in the installation directory. You may want to add this directory to your Path environment variable to make these commands available

The command-line install includes the following command line tools

CloverInstr	The Clover.NET instrumenter
HtmlReporter	The HTML Report generator
XmlReporter	The XML Report Generator
CloverRuntime.dll	The Clover Runtime library. Your Clovered builds must add a reference to this assembly
Clover DLLs	Various DLLs used by the Instrumenter and report generators

Please refer to the Licensing section for information on obtaining and installing a Clover.NET license. All Clover.NET installations require a license file to operate.

For information on the options and the operation of the command line tools please refer to the Command Line Tools Guide.

3.1.2. Visual Studio .NET 2003

The Clover.NET Visual Studio plugin is packaged as two separate MSI installers. The first, VSIP Interop Assembly Redist.msi, is supplied by Microsoft corporation and provides the necessary managed execution environment for Visual Studio plugins. Other plugins from other vendors may also install this component. It does not cause any harm to run the installer again to be sure.

The second installer, CloverVSNET.msi, provides the Clover command line tools and the package for the Visual Studio.NET plugin. Before installing, ensure Visual Studio is not currently running. The installer dialogs are straight forward. The installer will register the plugin with Visual Studio automatically so it will be available the next time you restart Visual Studio.

For the plugin to operate you must have a valid Clover license installed. Please refer to the Licensing section for information on obtaining and installing a Clover.NET license.

For information on the operation of the plugin please refer to the Plugin documentation. As the plugin also installs the command line tools, it is worth also reading the Command Line Tools Guide.

3.2. Clover.NET licensing

3.2.1. Clover.NET licensing

All Clover.NET installations require a valid Clover.NET license file to operate. This file is not included with the standard installers for Clover. It must be obtained from the Clover.NET website. The license file determines what operations you are licensed to use.

Installing the license file, clovernet.license, is simply a matter of placing the file into the same directory into which you installed Clover.NET.

4. Command Line Tools

4.1. Clover.NET Command Line Tools

4.1.1. Command Line Tools

Clover.NET provides a set of command line tools to perform the operations required to instrument and report on Code Coverage. If your build process uses `make`, `nmake` or similar tools, you can integrate Clover.NET into that build process with the command line tools. The Clover.NET NAnt tasks also use the command line tools to perform required Clover.NET operations.

The following sections describes the operation and supported options for each of the command line tools

CloverInstr	The Clover.NET instrumenter
CloverSolution	Visual Studio Solution Instrumenter
HtmlReporter	The HTML Report generator
XmlReporter	The XML Report Generator
CloverCheck	Coverage Target Checker
CloverRuntime.dll	The Clover Runtime library. Your Clovered builds must have a reference to this assembly.

A typical build process involving the command line tools would be

1. Instrument the code using CloverInstr
2. Compile the code with your existing compiler tool chain
3. When compiling add a reference to the CloverRuntime.dll assembly
4. Run your unit tests with your unit test framework such as NUnit
5. Generate a set of reports using HtmlReporter

You may want to add the Clover.NET install directory to the Windows Path environment variable. The plugin and Clover.NAnt tasks do not require this to operate.

4.1.2. Response Files

All command line tools accept response files. These are indicated by and argument starting with a '@' character. The options in the response file can be on multiple lines or multiple options may be on a single line. Where spaces are desired in values, they should be enclosed in double quot (") characters. A response file can be combined with command line options.

4.2. Clover.NET Instrumenter

4.2.1. CloverInstr - The Clover.NET instrumenter

CloverInstr is the Clover Instrumenter. It takes your source code and adds coverage instrumentation. The output of CloverInstr is a new set of source files which need to be compiled using your preferred compiler tool chain.

To perform its instrumentation, CloverInstr parses the source code to determine where instrumentation is required. CloverInstr will detect syntax errors in the source code, although this error reporting is less comprehensive than a compiler. It is recommended, therefore, that you ensure that your code compiles before attempting to Clover with CloverInstr.

As CloverInstr analyzes your code, it builds a coverage database recording information about the classes, methods and statements that make up your code. This information is combined with coverage recordings made during test runs to build the various coverage reports.

Once instrumented, you should build the Clovered code using your preferred compiler tool chain. The resulting assembly will generate coverage recordings when the code is run. Normally the Clovered code would be run during unit testing but there is no dependency of the Clovered code on the unit test framework. Coverage recordings can be gathered in other test scenarios such as functional and user-interface testing.

4.2.2. Usage

```
CloverInstr [options] [response_files] [source_files]
```

4.2.3. Options

CloverInstr supports a number of options which control how it instruments your code. These are detailed below. If you just type CloverInstr, it will show a summary of all options. Most options have both a long and short format. Use of the long format is recommended for use in build systems such as make since the system is more readable and maintainable. For command line usage, the short options may be more convenient

Long Name	Short Name	Required	Description
--initstring	-i	Yes	This option specifies the location of the coverage database maintained by CloverInstr. If the

			database does not exist, CloverInstr will create it.
--srcdir	-s	No	The --srcdir option gives CloverInstr the location of a source directory. If you do not specify a source directory you must provide one or more individual source files as arguments to CloverInstr. You may provide multiple -s options to Clover more than one set of Source files. CloverInstr will instrument all source files it finds and copy to the destination directory
--destdir	-d	Yes	The --destdir option tells CloverInstr where to place the instrumented source code. All files are placed into the destination directory relative to their source directory or to the source root, if one is specified. If any filename collisions occur, due to multiple source directories being specified, CloverInstr will rename the instrumented source file.
--flushinterval	-f	No	Normally Clover.NET writes out coverage information when the containing AppDomain is unloaded. In most cases that means that

			the coverage recording is written out when the application closes. In some circumstances you may not want to shut down the application being tested. By selecting a flush interval (in milliseconds), instrumentation will include additional code to write out the coverage recording at regular intervals.
--recursive	-r	No	By default when you specify a source directory with the --srcdir option, CloverInstr will pick up the source files in that directory. You can pick up all source files in sub-directories by using the --recursive option. CloverInstr will then scan all subdirectories for Source files.
--srcroot		No	Normally when picking up sources from multiple directories, CloverInstr uses the directory specified with --srcdir as to root directory. The file's name relative to that root directory is used to create the instrumented file relative to the --destdir directory. Files which are specified individually are instrumented to the destination directory directly.

			The --srcroot option allows you to specify which directory to use as the root for computing the relative filenames. This is useful where you wish to specify exactly which files to instrument but wish to retain the source directory structure in the instrumented code. It can also be used to select a subset of source directories to Clover.
--define		No	Defines a preprocessor symbol
--encoding		No	Specify the encoding of source files. By default, CloverInstr uses the system's default encoding.
--codepage		No	Specify the codepage for the source files. This is an alternative method to --encoding of specifying the character encoding used for the source files. It takes precedence over the --encoding option.
--noassign	-na	No	Do not instrument boolean expressions which contain assignments. Ordinarily Clover.NET instruments all boolean expressions. In some conditions that instrumentation can affect the compiler's ability to determine whether an uninitialized variable

			has been used. With this option Clover.NET will not attempt to instrument such expressions.
--rootnamespace		No	When instrumenting Visual Basic.NET code, this option controls the root namespace, which is used to enclose all type definitions in Visual Basic files. You should ensure this option matches the value you give to the vbc command line compiler.
--verbose	-v	No	Turns on verbose level logging. This gives more information about the progress of the Instrumenter
--debug		No	Turns on debug level logging. This will give more information than verbose level

4.2.4. Arguments

The arguments to the CloverInstr tool are individual files to be Clovered. If you have specified one or more --srcdir options, the source file arguments are optional.

4.2.5. Examples

```
CloverInstr -i CloverBuild\clover.cdb -s Source -d CloverBuild -r
```

This example creates, or updates, a coverage database at CloverBuild\clover.cdb. All of the source files in Source and its subdirectories are Clovered and the result is written out to the CloverBuild directory.

```
CloverInstr --define DEBUG -i CloverBuild\clover.cdb -s Source\Common --srcroot Source
```

This example is similar to the previous one except that only the Common source directory is Clovered. The use of --srcroot means that the directory structure of the Clovered code in

CloverBuild will be the same as the previous example. This example also sets the DEBUG preprocessor symbol

4.3. Clover.NET Solution Instrumenter

4.3.1. CloverSolution - Visual Studio solution instrumenter

CloverSolution is the Clover.NET instrumenter for Visual Studio solutions. It works by creating a Clovered version of your Visual Studio solution. It can also be used to Clover individual Visual Studio projects.

Any projects which are not Cloverable (i.e. any non C# and non VB projects) are dropped from the Clovered version of the solution. Once Clovered, you can build the Clovered version of the solution from the command line using devenv.com. You may optionally have the outputs from the Clovered version of your solution appear in the approriate location within you normal solution directory structure.

The coverage database and recordings generated by CloverSolution and the Clovered code are compatible with all other methods provided by Clover.NET for instrumenting code. This means that you may, for example, view the code coverage results from a Clovered solution within your IDE using the Clover.NET Visual Studio plugin.

4.3.2. Usage

```
CloverSolution [options]
```

4.3.3. Options

The CloverSolution control options are detailed below. If you just type CloverSolution, it will show a summary of all options. Most options have both a long and short format. Use of the long format is recommended for use in build systems such as make since the system is more readable and maintainable. For command line usage, the short options may be more convenient

Long Name	Short Name	Required	Description
--destdir	-d	No	The --destdir option tells CloverSolution where to place the instrumented solution. The solution directory structure is retained in the destination directory. If this option is not specified, it

			defaults to a directory CloverBuild.
--initstring	-i	No	This option specifies the location of the coverage database maintained by CloverSolution. If the database does not exist, CloverSolution will create it. This parameter is optional If you do not provide it, Clover.NET will create the coverage database as clover.cdb in the output directory.
--solution	-s	No	This options specifies the solution file you wish to Clover. If you do not specify a solution you must specify at least one Visual Studio project file.
--project	-p	No	This options specifies a Visual Studio project file. You may specify more than one project file and may do so in addition to a solution file. If CloverSolution detects that a dependent project is not included in the list of projects which are to be Clovered, it will give a warning.
--preserve		No	This option instructs CLoverSolution to generate Clovered versions of your project file so that the build outputs are written to the same location as

			the normal outputs of the corresponding project. This means that the Clovered output will overwrite your normal build output.
--config		No	Select the config you wish to build. Although solutions support multiple configs, CloverSolution needs to know which config to use when instrumenting code so that the appropriatre preprocessor defines are used to determine which code is to be Clovered. This defaults to Debug if not specified.
--flushinterval	-f	No	Normally Clover.NET writes out coverage information when the containing AppDomain is unloaded. In most cases that means that the coverage recording is written out when the application closes. In some circumstances you may not want to shut down the application being tested. By selecting a flush interval (in milliseconds), instrumentation will include additional code to write out the coverage recording at regular intervals.
--noassign	-na	No	Do not instrument boolean expressions

			which contain assignments. Ordinarily Clover.NET instruments all boolean expressions. In some conditions that instrumentation can affect the compiler's ability to determine whether an uninitialized variable has been used. With this option Clover.NET will not attempt to instrument such expressions.
--additional	-a	No	Specifies an additional file or directory to be copied to the Clovered solution. In some instances a solution may require files which are not listed in the solution projects. These files need to be specified manually. CloverSolution will determine where these files occur relative to the current solution and copy them to the corresponding location in the Clovered solution.
--include		No	The include option specifies a directory or file which is to be Clovered when generating the Clovered solution. If you do not specify any includes, all eligible files are Clovered. If you specify any includes, only files listed in the include

			specifications will be Clovered. Other elgible files will be copied across but will not be instrumented. Some files may still be modified to update items such as key file references, etc. When a directory is specified, all files under that diretcory are included.
--exclude		No	Exclude a file or directory from Clovering. When a source file is excluded from Clovering, a copy will still be made in the Clovered solution area. This copy will not have any instrumentation and will not appear in any Clover.NET reports. This option is useful where, for example, you solution contains a Test project and you do not want to see tests in your coverage results.
--clean	-c	No	Performs a full reClovering aof all source code. Normally when Clovering a solution, Clover.NET will not copy reclover files which have not changed.
--verbose	-v	No	Turns on verbose level logging. This gives more information about the progress of the Instrumenter
--debug		No	Turns on debug level

			logging. This will give more information than verbose level

4.3.4. Examples

```
CloverSolution -s Test.sln
```

This example creates, or updates, a coverage database at CloverBuild\clover.cdb. All of the source files in the Test solution's projects are Clovered and the result is written out to the CloverBuild directory.

```
CloverSolution -p src\Clover\Clover.csproj -p src\Tests\Tests.csproj --exclude src\Test
```

This example Clovers the Test and Clover projects and excludes the Test code from instrumentation.

4.4. Clover.NET XML Report Generator

4.4.1. XMLReporter - The Clover.NET XML Report Generator

The XML Report generator, XmlReporter, combines all of the coverage recordings generated by the Clovered code during execution, the coverage database created by instrumentation and the source files to generate an XML report on the coverage results.

4.4.2. Usage

```
XMLReporter [options]
```

4.4.3. Options

Long Name	Short Name	Required	Description
--initstring	-i	Yes	This option specifies the location of the coverage database generated during instrumentation. This is used to load the coverage database and any associated coverage recordings.
--output	-o	Yes	The name of the XML file which will contain the report
--title	-t	Yes	The report title.

--lineinfo	-l	No	Include information about each coverage counter for each file in the report.
--verbose	-v	No	Turns on verbose level logging. This gives more information about the progress of the report generation
--debug		No	Turns on debug level logging. This will give more information than verbose level

4.4.4. Examples

```
XMLReporter -i CloverBuild\clover.cdb -o report -t Test
```

This example generates a report in report.xml, titles Test.

```
XMLReporter -i CloverBuild\clover.cdb -o report -t Test -l
```

This example produces a more comprehensive report by including information about each coverage counter in the report.

4.5. Clover.NET HTML Report Generator

4.5.1. HtmlReporter - The Clover.NET HTML Report Generator

The HTML Report generator, HtmlReporter, combines all of the coverage recordings generated by the Clovered code during execution, the coverage database created by instrumentation and the source files to generate a comprehensive HTML report showing the coverage metrics and covered source.

4.5.2. Usage

```
HtmlReporter [options]
```

4.5.3. Options

Long Name	Short Name	Required	Description
--initstring	-i	Yes	This option specifies the location of the coverage database generated during

			instrumentation. This is used to load the coverage database and any associated coverage recordings.
--outputdir	-o	Yes	The location where the HTML report should be written. This area will contain all of the generated HTML files.
--title	-t	Yes	The Report title. This will appear in the browser window's title bar and in the coverage information header at the top of each generated page. If it contains spaces it should be quoted.
--bw		No	The bw option (Black and White) removes colour syntax highlighting from the generated reports. This reduces the size of the HTML files.
--showempty	-e	No	This option causes the HTMLReporter to include classes which have no content (no methods, etc) to be included in the report.
--nocache	-n	No	This option directs the reporter to include HTML meta directives to prevent caching of the reports by HTML browsers.
--hidesrc	-h	No	This option eliminates the rendering of source and just provides the overall, namespace, and class summary

			pages.
--hidebars	-b	No	This option prevents the rendering of the red-green coverage bar graphs
--tabwidth	-tw	No	This option controls the rendering of tabs characters in the source. All tabs are converted to spaces in the generated HTML. When a tab character is encountered it is replaced by spaces up to the next tab stop. This option controls where those tab stops occur. The default rendering is a tab stop every 4 characters.
--orderby	-c	No	Control the ordering of elements in the generated reports. The following values are supported • **alpha** - Sort classes and namespaces by name • **asc** - Sort by ascending coverage values • **desc** - Sort by descending coverage values
--verbose	-v	No	Turns on verbose level logging. This gives more information about the progress of the report generation
--debug		No	Turns on debug level logging. This will give more information than verbose level

4.5.4. Examples

```
HtmlReporter -i CloverBuild\clover.cdb -o report -t Test
```

This example generates a report into the report directory, titled Test. The report will include all source, with color syntax highlighting and coverage bar graphs

```
HtmlReporter -i CloverBuild\clover.cdb -o report -t Test -h -b
```

This example produces a more compact report by eliminating source code and coverage bars.

4.6. Clover.NET Coverage Checker

4.6.1. CloverCheck

CloverCheck allows you to quickly access coverage values from your coverage recording.

4.6.2. Usage

```
CloverCheck [options] [response_files]
```

4.6.3. Options

Long Name	Short Name	Required	Description
--initstring	-i	Yes	This option specifies the location of the coverage database generated during instrumentation. This is used to load the coverage database and any associated coverage recordings.
--overall	-o	No	Displays the overall coverage of the coverage recordings.
--namespace	-n	No	Allows you to specify the names of particular namespaces whose coverage is to be displayed.
--class	-c	No	Allows you to specify the names of particular class whose coverage is to be displayed.

--all	-a	No	Display all coverage values including overall, all namespaces and all classes.

4.6.4. Examples

```
CloverCheck -i CloverBuild\clover.cdb -o
```

This example prints the overall coverage values. The output would be like this:

```
        Loaded database at 'CloverBuild\clover.cdb'
        Processing 70 coverage recordings

        Coverage Results
        ----------------
        Overall: Total:19.7% Method:21.7% Statement:18% Conditional:25%
```

4.7. Clover.NET Runtime Assembly

4.7.1. CloverRuntime - The Clover.NET Runtime Assembly

The Clover Runtime assembly provides the necessary support functions for instrumented code. To compile and run the instrumented code, the CloverRuntime.dll assembly must be added as a reference to the compilation process. The NAnt tasks and the Visual Studio plugin take care of this process automatically. for command line driven builds, such as when using make, you need to adjust your build process to include the runtime as a reference. The CloverRuntime assembly is available in the Clover.NET installation directory.

5. Visual Studio Plugin

5.1. Clover.NET Visual Studio Plugin

5.1.1. Visual Studio Plugin

The Clover.NET Visual Studio plugin provides code coverage facilities for Visual Studio.NET 2003 users. These facilities are available within the Visual Studio environment and consist of the following:

1. A Clover options tool window to control the Clovering process
2. A Coverage Viewer which gives a tree view of the solution's namespaces of the coverage
3. A set of coverage markers which display coverage within the main Visual Studio editor display

The two tool windows are standard Visual Studio tool windows and can be docked in the normal way. Please refer to the following sections for more information

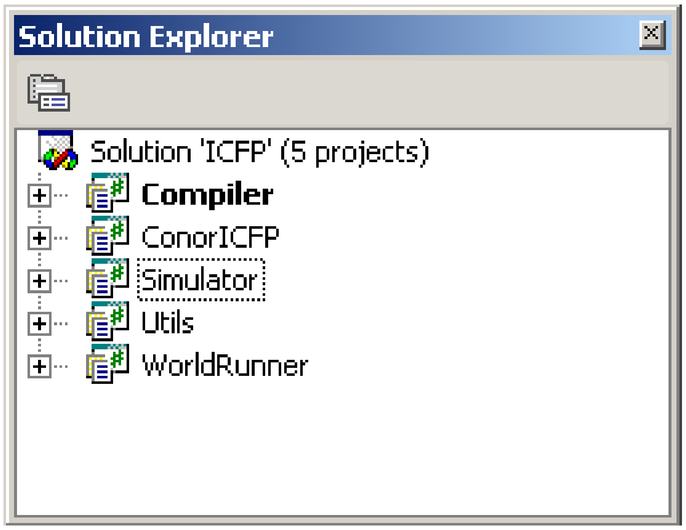
- Clovering with the Plugin
- Options Tool Window
- Coverage View Tool Window
- Coverage Display in Visual Studio Editor
- Report Generation
- How the Plugin Operates

5.2. Clovering with the Plugin

5.2.1. Clovering with the plugin

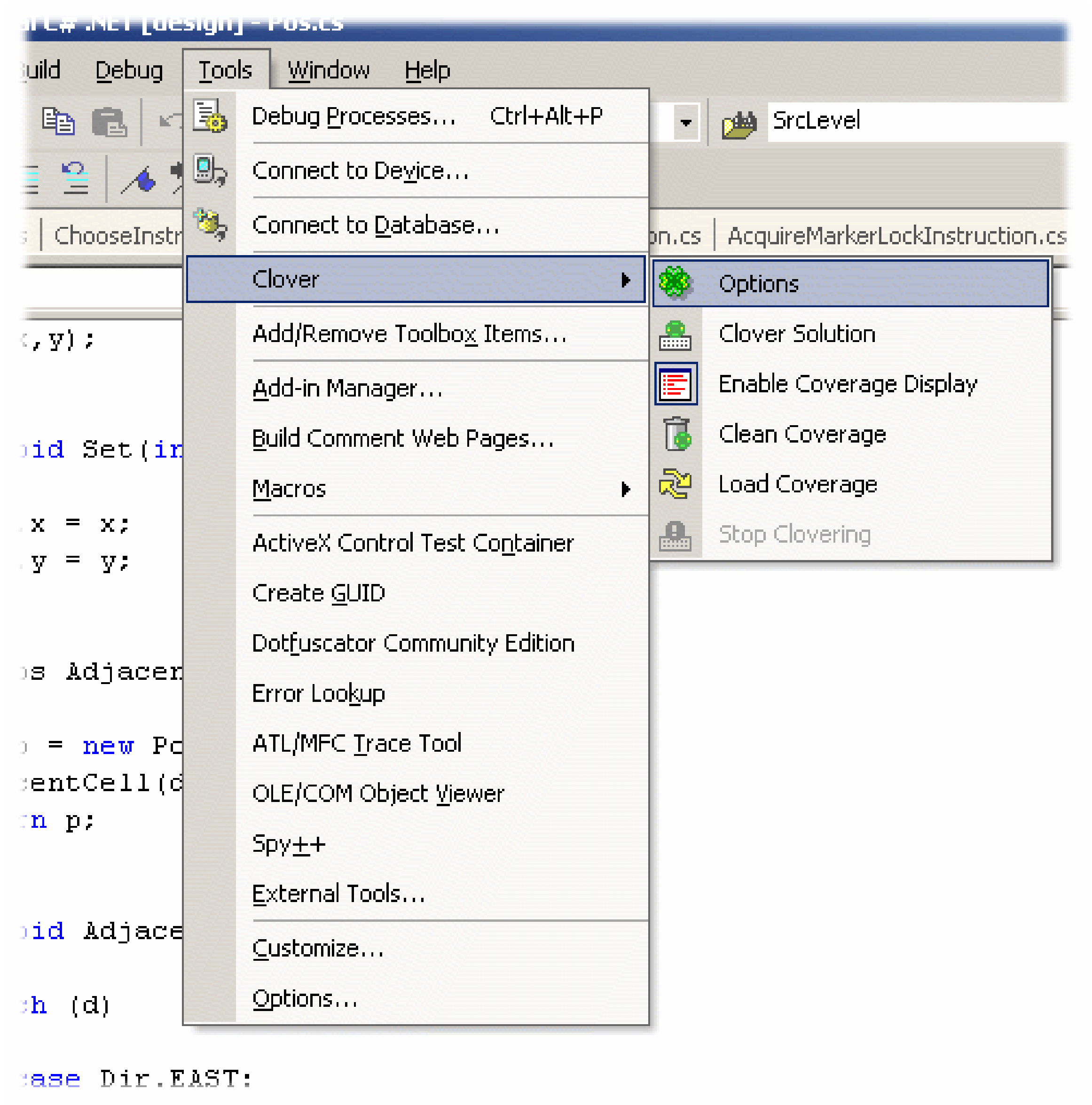
This section describes how you can use the Clover.NET Visual Studio plugin to Clover your projects. In this section we will walk through the steps necessary to Clover a small multi-project solution.

1. The first step you should take to Clover your solution is to ensure your solution builds normally. In this solution walkthough, there are 5 projects as shown:



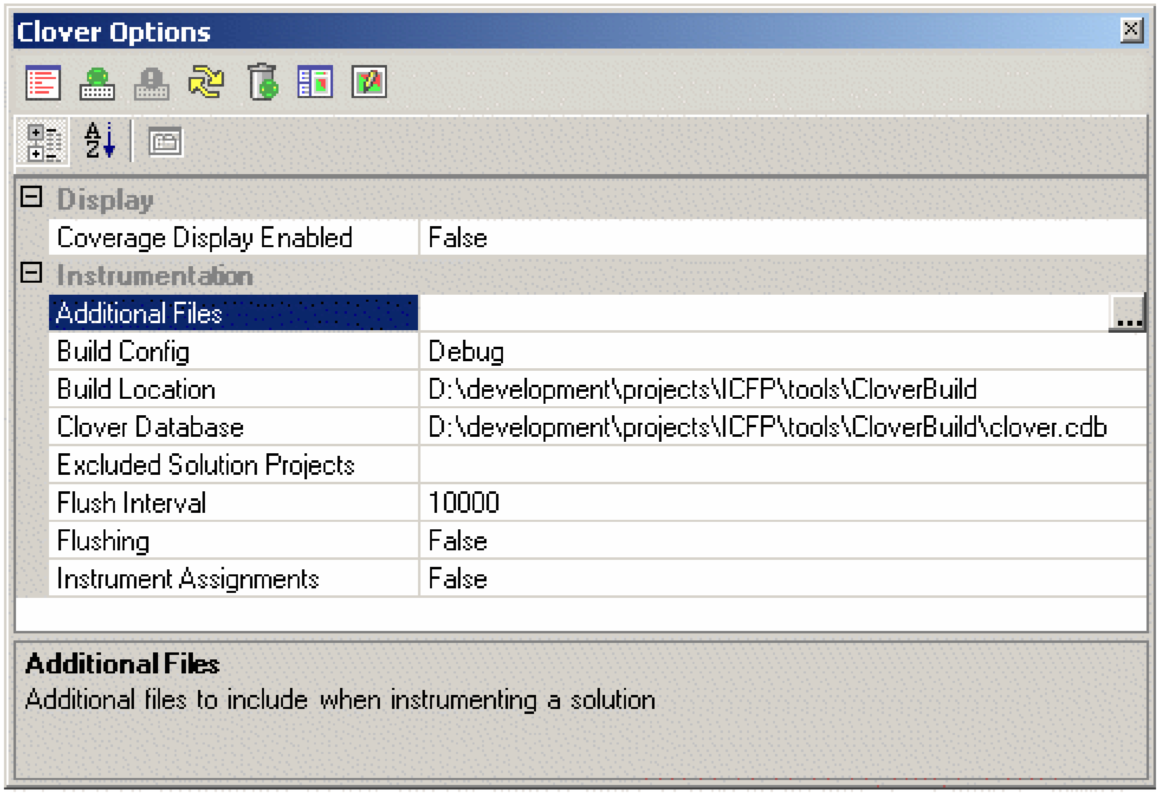
Example Solution

2. Once the solution builds, the Clover options should be setup for Clovering. Select the Clover Options Tool Window from the Clover menu as shown:



Clover Menu

An example Options window is shown below. Please refer to the section on the Options Tool Window for detailed information on the components of the Clover options window.



Display

Coverage Display Enabled	False

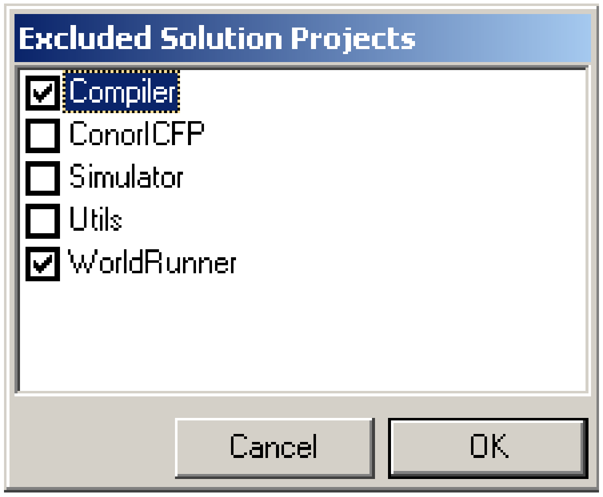
Instrumentation

Additional Files	
Build Config	Debug
Build Location	D:\development\projects\ICFP\tools\CloverBuild
Clover Database	D:\development\projects\ICFP\tools\CloverBuild\clover.cdb
Excluded Solution Projects	
Flush Interval	10000
Flushing	False
Instrument Assignments	False

Additional Files
Additional files to include when instrumenting a solution

Clover Options Window

3. In this walkthrough, two of the projects will not be tested and so these will be excluded from Clovering. Bring up the Excluded Projects Dialog by clicking on the Property in the Options tool window.



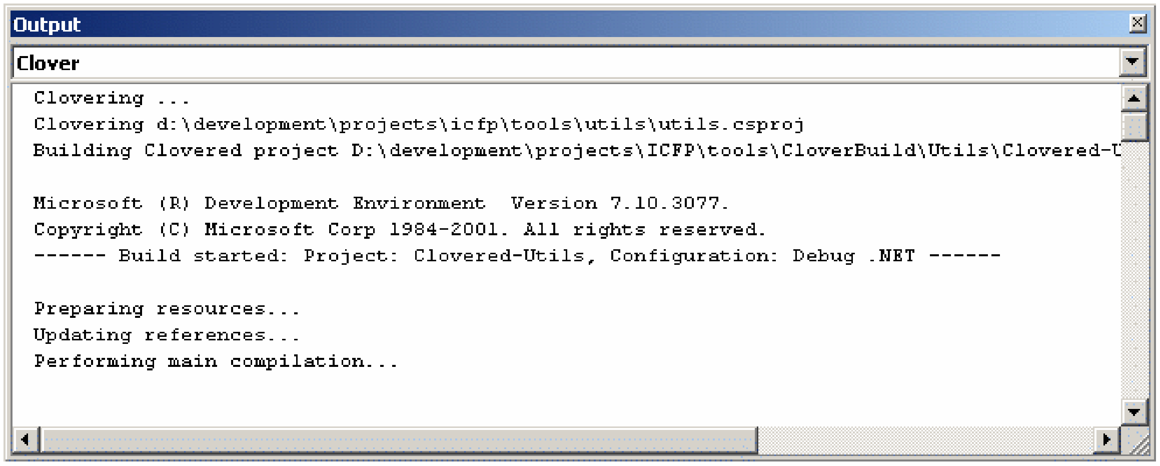
- [✓] Compiler
- [ ] ConorICFP
- [ ] Simulator
- [ ] Utils
- [✓] WorldRunner

Example Exclude

The Compiler and WorldRunner projects are excluded in this walkthrough. The default values are used for the remaining option settings.

4. It is now time to Clover the project. Click on the build button to start the Clovering process. Clover.NET creates a copy of all of the solution projects in the CloverBuild area, Clovering the sources files of those which are not specifically excluded.

   Clover.NET creates and uses a Clover specific pane in the Visual Studio output window where it writes information about the progress of the build. The output window at the start of Clovering is shown:



```
Clovering ...
Clovering d:\development\projects\icfp\tools\utils\utils.csproj
Building Clovered project D:\development\projects\ICFP\tools\CloverBuild\Utils\Clovered-U

Microsoft (R) Development Environment  Version 7.10.3077.
Copyright (C) Microsoft Corp 1984-2001. All rights reserved.
------ Build started: Project: Clovered-Utils, Configuration: Debug .NET ------

Preparing resources...
Updating references...
Performing main compilation...
```

Output Window

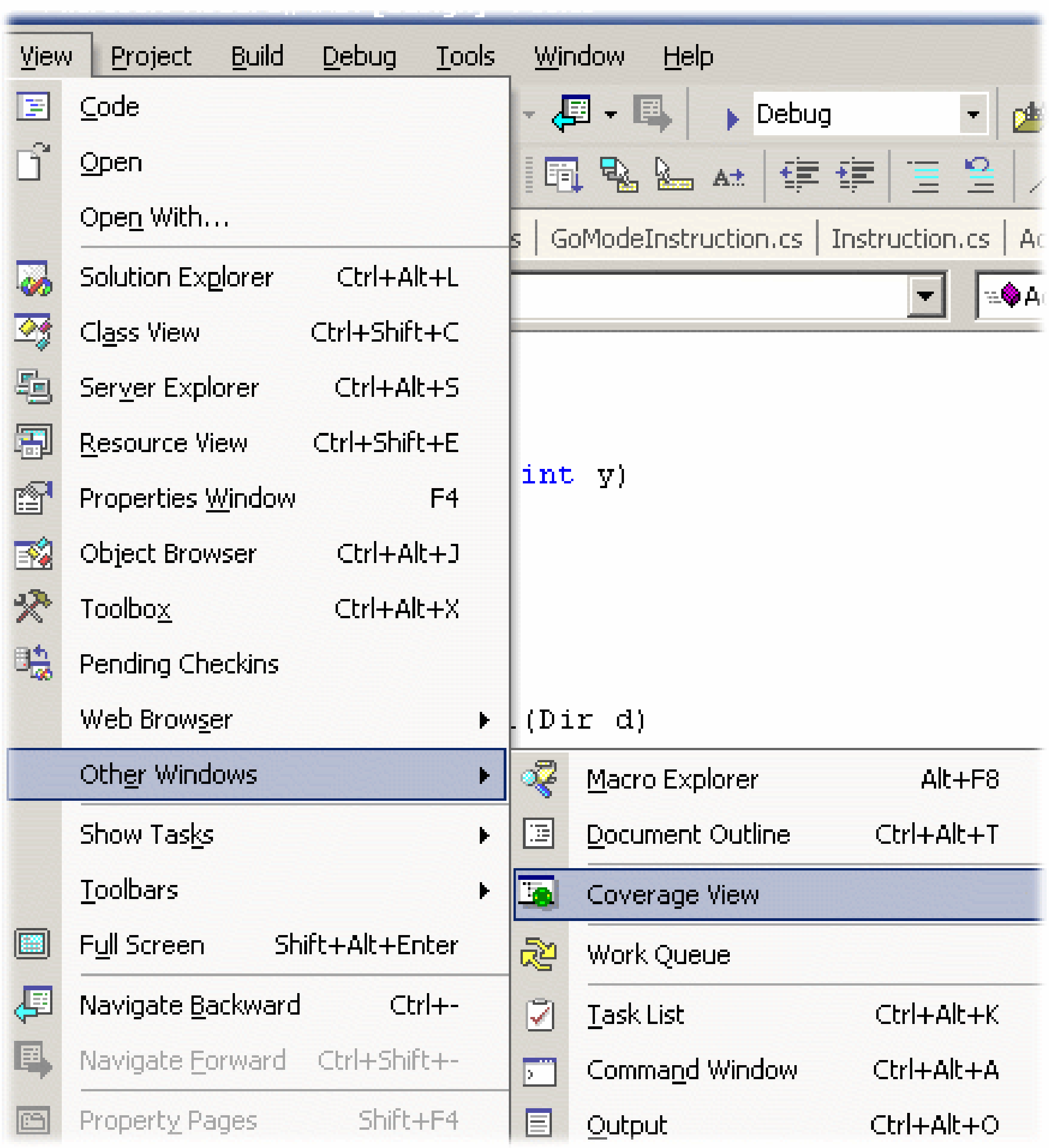
The Clover.NET plugin launches an independent, non-interactive instance of Visual Studio to build the Clovered project. Most of the output will be the familiar Visual Studio build output. At the end of the build process, Clover.NET will report a summary of the Clovering process

```
---------------------- Done ----------------------

  Clover: 5 projects, 5 Clovered, 5 built, 0 failed
```

Clovering is done in the background and Visual Studio can be used as normally in the foreground. If you wish to stop a Clover build, use the Stop Clover Button on the toolbar. This button only becomes active during a Clover build.

5. At this point the project has been Clovered and it is now ready to be tested. The unit test for this project is run external to Visual Studio using the components built under the CloverBuild directory

6. Once the tests have been run, Coverage can be displayed. Open the Coverage View tool window from the Visual Studio View menu as shown:

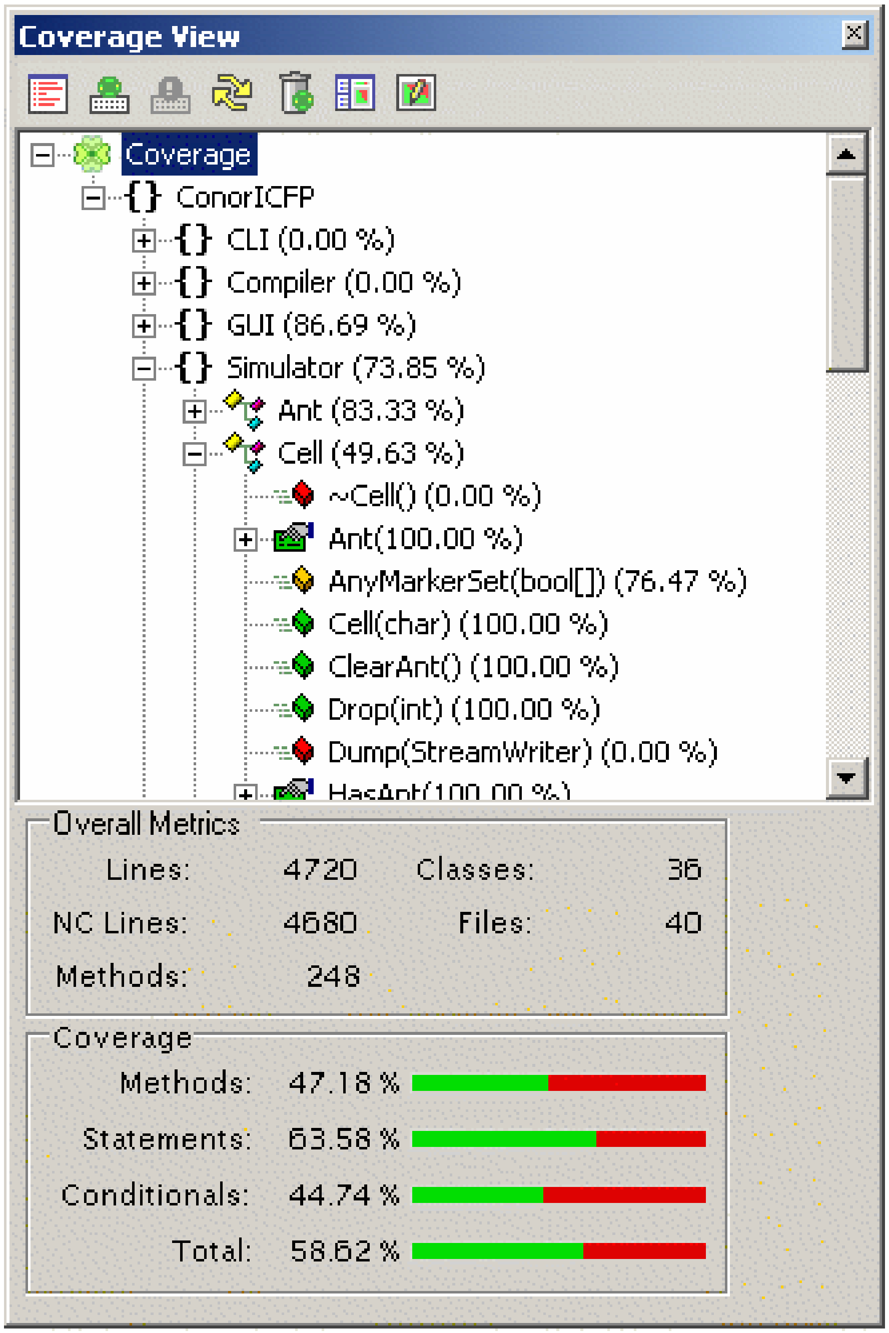


View Menu

7. The Coverage View tool Window provides a tree view of the project namespace. Initially the coverage tree will show no coverage since no coverage information has been loaded. You may load the latest coverage information using the Load Button. If you perform

subsequent test runs, you can load the additional coverage recordings by using this button.

An example is shown below:



```
Coverage
  ConorICFP
    CLI (0.00 %)
    Compiler (0.00 %)
    GUI (86.69 %)
    Simulator (73.85 %)
      Ant (83.33 %)
      Cell (49.63 %)
        ~Cell() (0.00 %)
        Ant(100.00 %)
        AnyMarkerSet(bool[]) (76.47 %)
        Cell(char) (100.00 %)
        ClearAnt() (100.00 %)
        Drop(int) (100.00 %)
        Dump(StreamWriter) (0.00 %)
        HasAnt(100.00 %)
```

Overall Metrics

Lines:	4720	Classes:	36
NC Lines:	4680	Files:	40
Methods:	248		

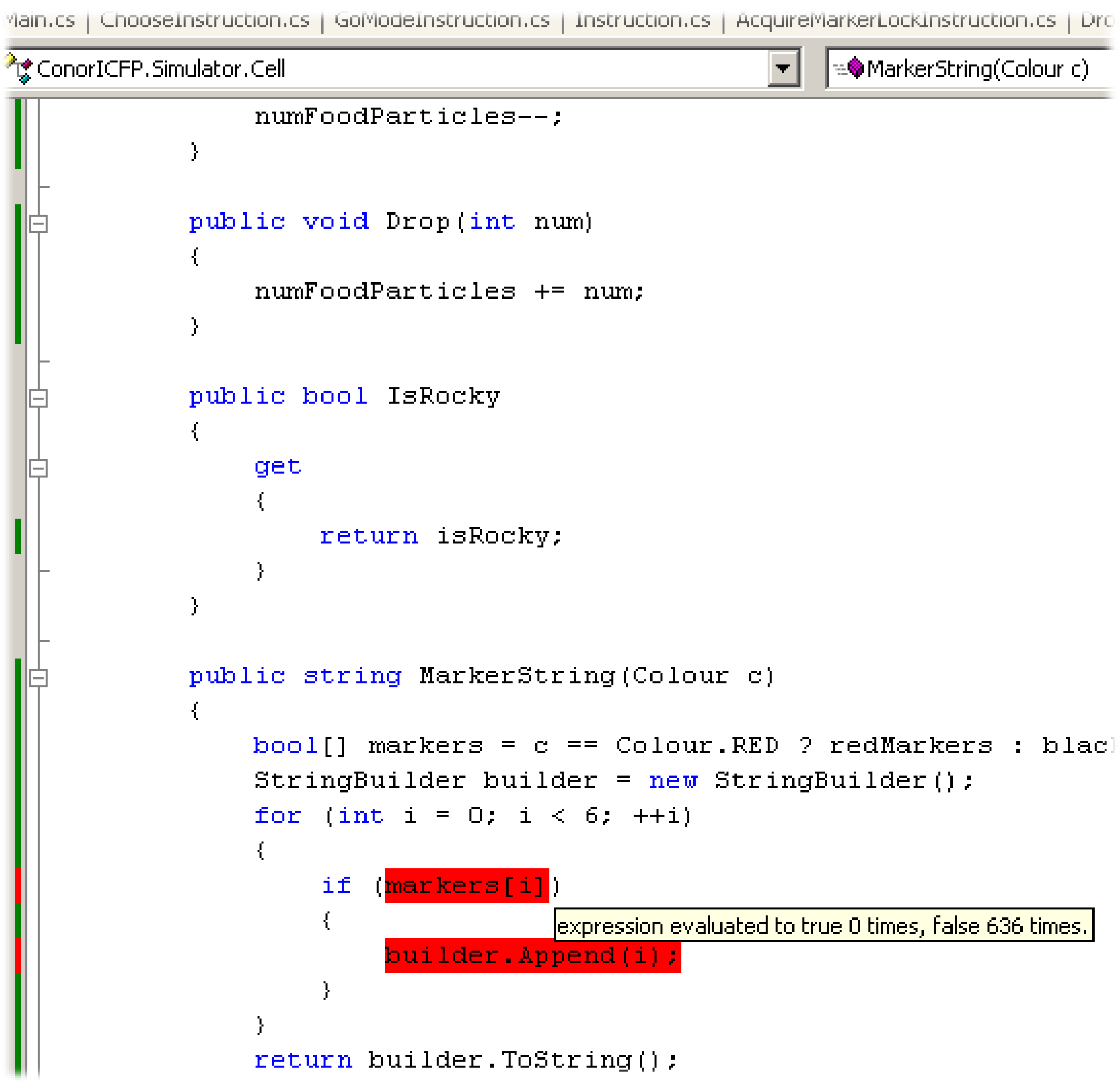
Coverage

Methods:	47.18 %
Statements:	63.58 %
Conditionals:	44.74 %
Total:	58.62 %

Coverage Tree

Please refer to the section on the Coverage View Tool Window for detailed information on this tool window.

8. The other way to view coverage information is to highlight code in the Visual Studio editor according to its coverage. To do this, select the Display button on the toolbar. The coverage information will then be displayed in the editor as shown:



Main.cs | ChooseInstruction.cs | GoModeInstruction.cs | Instruction.cs | AcquireMarkerLockInstruction.cs | Dro
ConorICFP.Simulator.Cell
MarkerString(Colour c)

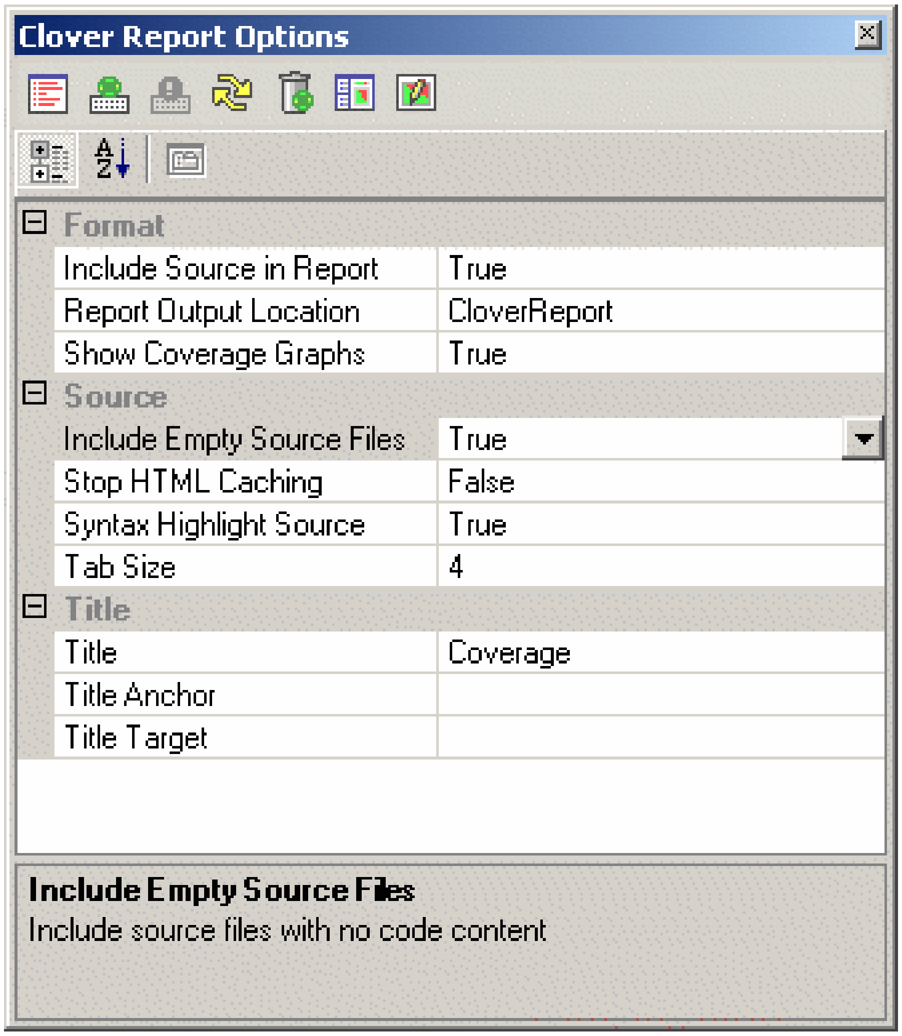
```
            numFoodParticles--;
        }

        public void Drop(int num)
        {
            numFoodParticles += num;
        }

        public bool IsRocky
        {
            get
            {
                return isRocky;
            }
        }

        public string MarkerString(Colour c)
        {
            bool[] markers = c == Colour.RED ? redMarkers : blac
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < 6; ++i)
            {
                if (markers[i])
                {
                    builder.Append(i);
                }
            }
            return builder.ToString();
```

expression evaluated to true 0 times, false 636 times.

Editor Display

Where code elements are not covered, they will be highlighted. The Visual Studio editor's gutter will also contains a marker where coverage has been recorded. This will be present even when the element is covered by testing. Please refer to the section on the Coverage Editor display for more information on this

9. If your Clover.NET license supports the generation of HTML reports you may configure and generate HTML reports directly in the Visual Studio IDE. You may use the Report Options button to configure your reports as shown:



Format	
Include Source in Report	True
Report Output Location	CloverReport
Show Coverage Graphs	True
Source	
Include Empty Source Files	True
Stop HTML Caching	False
Syntax Highlight Source	True
Tab Size	4
Title	
Title	Coverage
Title Anchor	
Title Target	

Include Empty Source Files
Include source files with no code content

Report Options

The HTML Report button can then be used to generate the report. The report generation occurs in the background. Once the report has been generated, you may ctrl-click on the URL in the output window to view the HTML reports in the IDE.

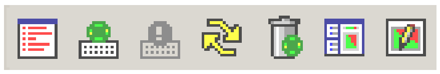
Please refer to the section on the Report Generation for detailed information.

10. Your solution has now been Clovered. You should now have some idea of the quality of your testing and what you can do to improve your testing efforts.

5.3. Clover.NET toolbar

5.3.1. Clover.NET toolbar

Both the Clover.NET Options and Coverage View tool windows provide a toolbar with buttons for common Clover.NET functions. These buttons also correspond to operations on the Clover.NET menu. The toolbar is shown below:


Clover.NET toolbar

5.3.2. Toolbar Functions

Display Enable	This button controls display of Coverage Information within the Visual Studio editor. Clicking this toggles coverage info display on and off in all editor tabs.
Clover Build	This button starts a Clover build. A Clover build involves Instrumenting the source code and then building the code externally. While a build is in progress, this button will be disabled.
Stop Clover	This button is normally disabled and becomes enabled when a Clover Build is in progress. It is used to stop the Clover build. It may take some time for the build to actually stop if it is currently waiting for an external build process to complete
Load Coverage	This button will reload coverage information. You should use this to load your initial set of coverage recordings and whenever you generate any new recordings by, for example, running additional tests.
Clean	This button removes the Coverage database and all coverage recordings. It ensures that all source will be reinstrumented at the next Clover build.

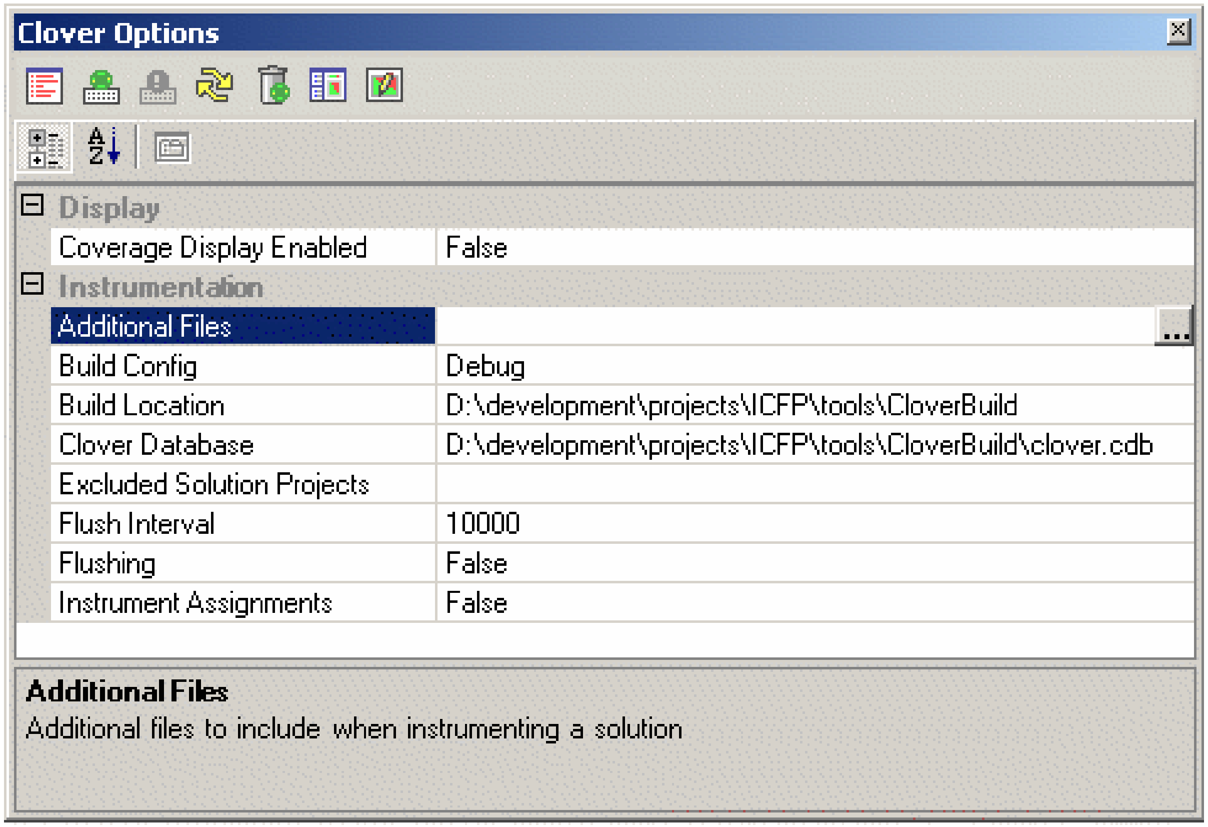
Generate Report	Generates an HTML report. This button will only be visible if your Clover.NET license includes HTML report generation.
Generate Report	Brings up the Report Options tool window allowing you to configure how HTML coverage reports are to be rendered. This button will only be visible if your Clover.NET license includes HTML report generation.

5.4. Options Tool Window

5.4.1. Clover Options Tool Window

The Clover Options tool window is activated by selecting the Options entry from the Clover menu.

The Clover.NET plugin only loads when required, so you may notice that initially all menu items except for the Options item are disabled. This is normal. Once the Options item is selected, Visual Studio will load the Clover.NET plugin and the remaining items will become enabled as appropriate. The layout of the Clover.NET options tool window is shown below


Clover Options

Display	
Coverage Display Enabled	False
Instrumentation	
Additional Files	
Build Config	Debug
Build Location	D:\development\projects\ICFP\tools\CloverBuild
Clover Database	D:\development\projects\ICFP\tools\CloverBuild\clover.cdb
Excluded Solution Projects	
Flush Interval	10000
Flushing	False
Instrument Assignments	False

Additional Files
Additional files to include when instrumenting a solution

Clover Options Window

The options tool window consists of a Clover.NET toolbar with buttons for most common Clover.NET functions and a properties window to control the operation of Clover.NET. The toolbar also appears on the Coverage View tool window and corresponds with the items on the Clover menu. The following table details the function of each of the Clover options properties.

Property	Function
Coverage Display Enabled	This option controls the display of coverage information in the Visual Studio editor. It corresponds to the Display button on the toolbar. When this property is set to True and there is coverage information loaded, Clover.NET will display the current coverage information within the Visual Studio Editor. Refer to the Coverage Display section for more details.
Additional Files	Clover.NET determines the files it must

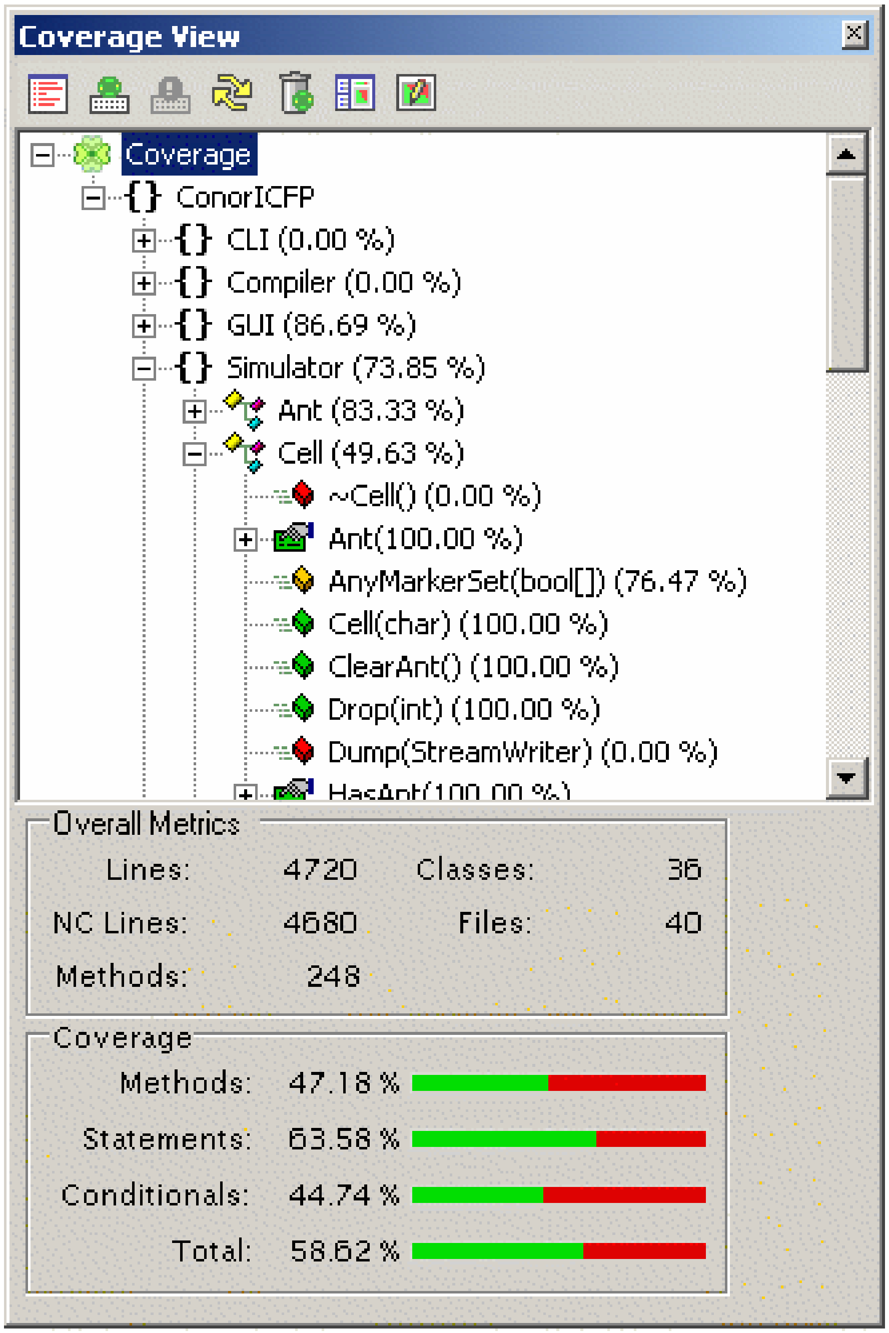
	instrument and copy by using the project information maintained by Visual Studio. If a file is required which is not included in the project hierarchy, you must add it explicitly using this option. You can type the file names in directly or use the associated dialog to select and add files or folders interactively. All files are added relative to the Solution file. Clover.NET will not handle key file references for assembly signing automatically. The key files do not need to be added explicitly.
Build Config	This is the build configuration that Clover.NET will use when building the Clovered projects. This defaults to the DEBUG configuration. This setting will affect which preprocessor definitions are used when instrumenting the source.
Build Location	This option controls where Clover.NET will create the Clovered versions of the projects within your solution. By default this is set to the directory CloverBuild relative to the Solution file. When Clover.NET has finished Clovering and building your projects, you will find the output assemblies here. These are the assemblies you need to use in testing to generate coverage recordings.
Clover Database	This determines the location of Clover.NET's coverage database. The default is to create the database at CloverBuild\clover.cdb relative to the solution. This approach keeps all Clover.NET related files in one area. You may, however change this to any location you like. In particular you can select an existing database if you wish to aggregate coverage recording across a number of solutions.
Excluded Solution Projects	In many solutions you may not want to Clover all projects. For example, you may have your unit tests as a separate project within the solution and not want coverage information on the tests themselves. The option provides a dialog where you can select which projects to exclude from Clovering. Excluded projects are still copied to the CloverBuild area but are not instrumented. The excluded projects are also built to satisfy inter-project dependencies, etc. If you are

	testing and the test project is excluded from Clovering, you should use the version built in the CloverBuild area as it will run with its Clovered dependencies.
Flush Interval and Flushing	These two properties control the insertion of flushing code into the instrumented code generated by the plugin. Normally coverage information is written out when the AppDomain is unloaded. This usually occurs when the application closes. Flushing is used to write out coverage information without needing to have the application close.
Instrument Assignments	Instrument boolean expressions which contain assignments. In some conditions the instrumentation of boolean expressions can affect the compiler's ability to determine whether an uninitialized variable has been used. With this set to false, Clover.NET will not attempt to instrument such expressions.

5.5. Coverage Tree View Tool Window

5.5.1. Clover.NET Coverage Tree View

The Coverage Tree View is activated from the View -> Other Windows menu in Visual Studio. An example of the tool window is shown below:


Coverage View

Coverage
- ConorICFP
  - CLI (0.00 %)
  - Compiler (0.00 %)
  - GUI (86.69 %)
  - Simulator (73.85 %)
    - Ant (83.33 %)
    - Cell (49.63 %)
      - ~Cell() (0.00 %)
      - Ant(100.00 %)
      - AnyMarkerSet(bool[]) (76.47 %)
      - Cell(char) (100.00 %)
      - ClearAnt() (100.00 %)
      - Drop(int) (100.00 %)
      - Dump(StreamWriter) (0.00 %)
      - HasAnt(100.00 %)

Overall Metrics

Lines:	4720	Classes:	36
NC Lines:	4680	Files:	40
Methods:	248		

Coverage

Methods:	47.18 %	🟩🟥
Statements:	63.58 %	🟩🟥
Conditionals:	44.74 %	🟩🟥
Total:	58.62 %	🟩🟥

Coverage Tree

There are three components to the tool window. At the top is the standard Clover toolbar to control Clover.NET operations. This toolbar also appears on the Options tool window and corresponds with the operations in the Clover menu.

The main component of the Coverage View window is the Coverage Tree. This is a namespace tree of your .NET Clovered code. It is similar to the Visual Studio Class View. Each node of the namespace that contains code shows the coverage figure for classes at that node of the namespace hierarchy. It does not include coverage for nodes lower in the hierarchy.

Below the coverage tree is a metrics pane which displays metrics and Coverage information. As you select a node in the coverage tree, this panel will update to show that node's metrics. The metrics pane can show metrics and coverage at the overall, namespace, class and file levels. Not all fields are relevant at all levels.

In the coverage tree, at the method level, the icons displayed for methods are color coded. A green icon means that all statements in the method have been executed, red indicates that the method was not entered and amber means that some of the statements in the method were not executed. You can double click on the class and method nodes in the tree and Clover.NET will bring up the source code for that item in the editor. This makes it easy to navigate around your code from a Coverage point of view and find areas which need attention in your testing strategy.

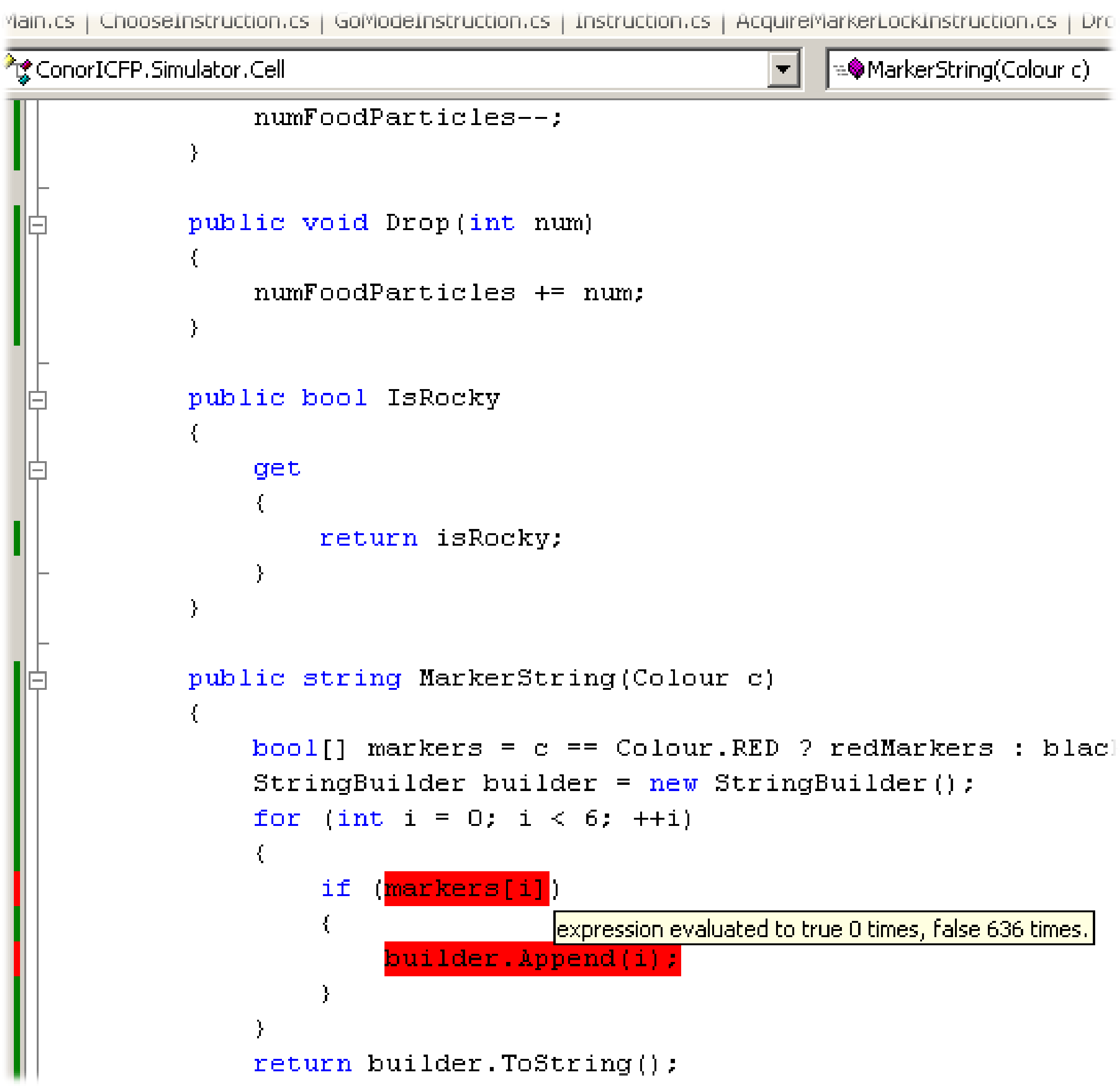
5.6. Coverage Display

5.6.1. Coverage Display

Clover.NET is able to display Coverage information directly in the Visual Studio Editor using text markers. You may be familiar with text markers when using breakpoints in the debugger. Clover.NET has four different types of markers:

Method coverage marker	Indicates that a Method has not been entered
Conditional coverage marker	Indicates that a conditional expression has not taken on both true and false values.
Statement coverage marker	Indicates that a statement has not been executed
Covered marker	Indicates that the code has been covered

An example of some of the markers in use is shown below:


Main.cs | ChooseInstruction.cs | GoModeInstruction.cs | Instruction.cs | AcquireMarkerLockInstruction.cs | Dro
ConorICFP.Simulator.Cell
MarkerString(Colour c)

```
        numFoodParticles--;
    }

    public void Drop(int num)
    {
        numFoodParticles += num;
    }

    public bool IsRocky
    {
        get
        {
            return isRocky;
        }
    }

    public string MarkerString(Colour c)
    {
        bool[] markers = c == Colour.RED ? redMarkers : blac
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < 6; ++i)
        {
            if (markers[i])
            {
                builder.Append(i);
            }
        }
        return builder.ToString();
```

expression evaluated to true 0 times, false 636 times.

Editor markers

All markers display tooltips to give more information about the reason why the coverage is not shown. This is especially useful for understanding why coverage of a conditional expression is highlighted. When you hover over the conditional, you will be able to see whether the expression has never evaluated to true, false, or neither value. In the above example we can see that the markers array never contained a true value.

Visual Studio renders the tooltips at the end of the marker area so in some cases the tooltip may not be visible. For methods, you may collapse the method display to more easily view the tooltip.

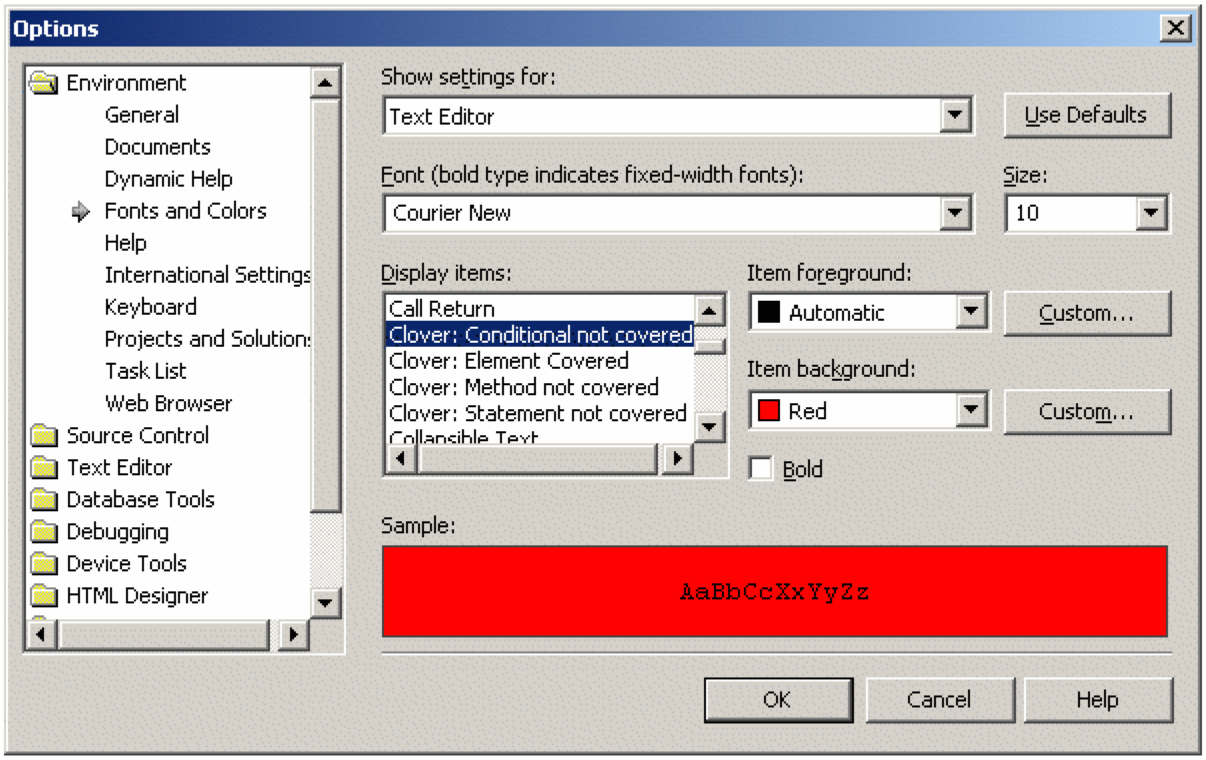
Since markers may overlap, Clover.NET will not create markers which are unnecessary. When a method has not been entered, for instance, the statement and conditional markers for the method's code will not be displayed since that information would be redundant. When a method has been entered but contains statements that have not executed, the Covered marker is used for the method, with the statement marker having a higher priority and rendering for the appropriate statements. You can see this in the gutter of the example above where the conditional and statement markers override the method's Covered marker

The "covered" marker will indicate how many times a statement or method has been executed or entered. This number is aggregated from all coverage recordings from the current version of the code on display. This information can be handy in some cases to confirm that tests are executing as you expect but you must be careful to interpret this correctly when there are multiple coverage recordings relevant to the current code.

The covered marker only renders in the gutter, it does not affect the normal display of text in the editor window, although such code will display a tool tip.

Since Visual Studio uses markers for other purposes, such as breakpoints it is possible to turn coverage display on and off quickly with the Display button on the toolbar.

By default the coverage markers use a standard Visual Studio red color as the background to source elements which have not been covered. This can be rather garish but can be customized by Visual Studio just like any other marker. To do this, select the Options item from the Tools menu. In the dialog, select the "Fonts and Colors" entry from the Environment folder. The Clover related markers all appear as entries prefixed by "Clover :" as shown in this image:



Show settings for: Text Editor

Font (bold type indicates fixed-width fonts): Courier New

Size: 10

Display items:
Call Return
Clover: Conditional not covered
Clover: Element Covered
Clover: Method not covered
Clover: Statement not covered
Collapsible Text

Item foreground: Automatic

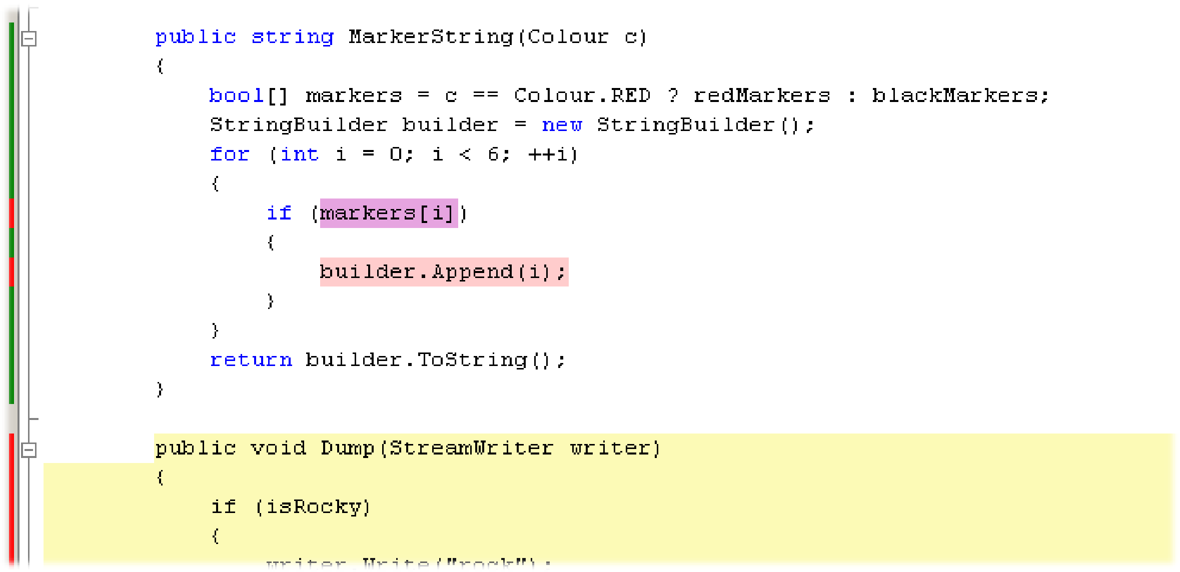
Item background: Red

Bold

Sample: AaBbCcXxYyZz

Marker Colors

You can choose any colors you like for the markers, either standard Visual Studio colors or a custom color. You can select different colors for each marker if you prefer. These colors do not affect the colors used in gutter coverage icons. Although its color can be customized, the covered marker will not display colored text. The following shows the display of all markers customized to different colors.



```
public string MarkerString(Colour c)
{
    bool[] markers = c == Colour.RED ? redMarkers : blackMarkers;
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < 6; ++i)
    {
        if (markers[i])
        {
            builder.Append(i);
        }
    }
    return builder.ToString();
}

public void Dump(StreamWriter writer)
{
    if (isRocky)
    {
```

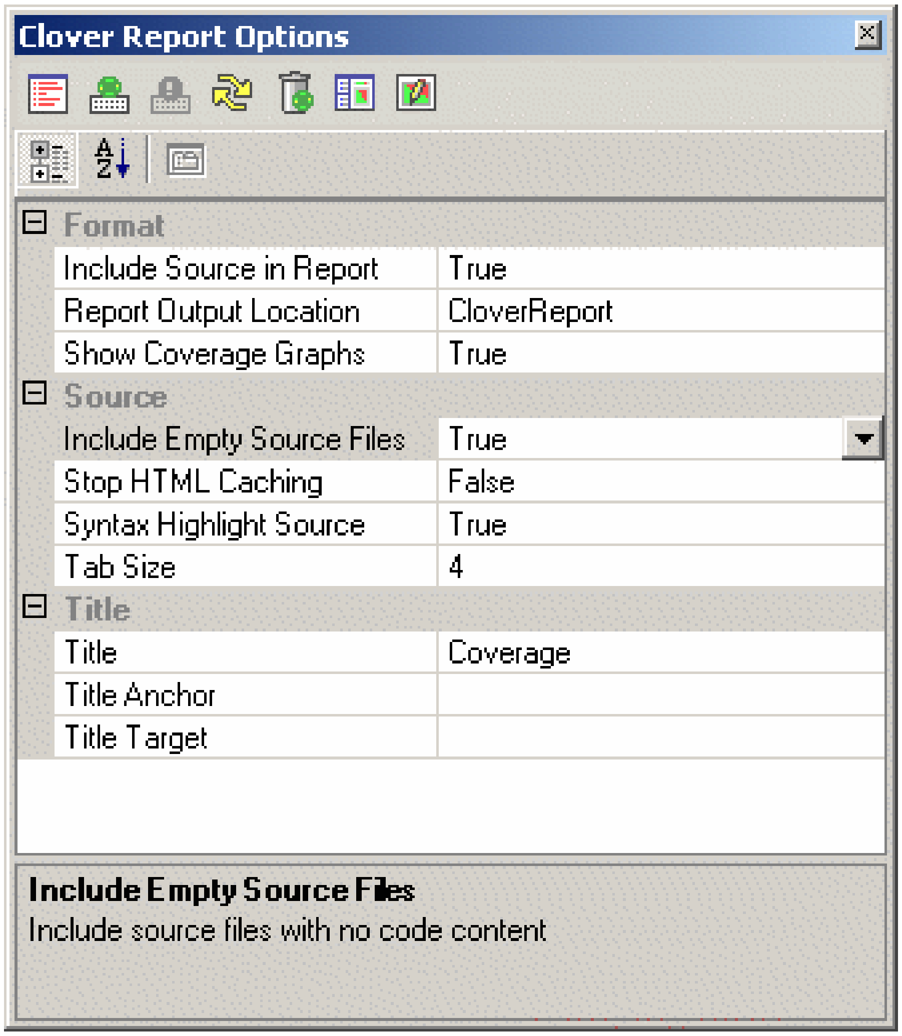
Customized colors

5.7. Report Generation with the Plugin

5.7.1. Report Generation with the plugin

If your Clover.NET license supports HTML report generation, the plugin provides options to configure and generate reports within the IDE.

You may use the Report Options button to configure your reports as shown:



Format	
Include Source in Report	True
Report Output Location	CloverReport
Show Coverage Graphs	True
Source	
Include Empty Source Files	True
Stop HTML Caching	False
Syntax Highlight Source	True
Tab Size	4
Title	
Title	Coverage
Title Anchor	
Title Target	

Include Empty Source Files
Include source files with no code content

Report Options

The report options tool window consists of a property grid with the properties you can use to configure your HTML reports.

Property	Function
Include Source in Report	When true, coverage information, such as coverage counts and indications of uncovered elements, is rendered for each source file. If false, only the summary pages showing the coverage values at the project and namespace levels are generated.

Report Output Location	This is the location where the report will be rendered. If a relative name is given, the report location will be relative to the Visual Studio solution file.
Show Coverage Graphs	When true, this option generates the green/red color bar indicating the relative coverage for project elements (project, namespaces, files, classes). If this is false, only the numeric coverage values are included in the report.
Include Empty Source Files	This option controls whether files with no code content are included in the report. Such files may consist of non-code elements such as comments and whitespace.
Stop HTML Caching	When true, this option included directives in the generated HTML to prevent the HTML data being cached in the browser. This would be used where you are publishing the reports many times in the course of a browser session.
Syntax Highlight Source	When true, code elements, such as keywords, string literals, etc. are highlighted in different colors. This increases the size of the generated HTML. To generate a smaller HTML report, set this option to False.
Tab Size	This option controls the rendering of tabs characters in the source. All tabs are converted to spaces in the generated HTML. When a tab character is encountered it is replaced by spaces up to the next tab stop. This option controls where those tab stops occur. The default rendering is a tab stop every 4 characters.
Title	The Report title. This will appear in the browser window's title bar and in the coverage information header at the top of each generated page. If it contains spaces it should be quoted.
Title Anchor	The Report title may be rendered as a hyperlink. This option gives the href of that hyperlink.
Title Target	Since the HTML reports are part of a frameset, this option allows you to specify a target for the title hyperlink. By default the special _top target is used to open the link in a new window.

Once you have configured the report, you can generate the report using the HTML Report button on the toolbar. The report is generated in the background and you may perform other Visual Studio operations while the report is being generated. The Clover.NET pane of the output window displays the progress of the report generation. When the report is completed, the URL of the report is displayed. This can be control-clicked to view the report directly in the IDE's browser window as shown:

5.8. How the Plugin Operates

5.8.1. How the Plugin Operates

This section describes how the Clover.NET plugin for Visual Studio works. The main objective of the plugin is to build a Clovered version of your solution's projects using their current settings without impacting on the current settings or operation of the Solution.

When Clovering a solution, the Clover.NET plugin performs the following steps:

1. The first step taken by the plugin in Clovering a solution is to create a copy of each solution project. As these are created, they are renamed to have a "Clover-" prefix. This is to distinguish these project files from the original project files.
2. As the plugin does not use a solution file, it builds each project in isolation. To do that it needs to adjust each project's assembly references. In addition to adding a reference to the Clover runtime assembly, the plugin will adjust any inter-project references to direct references to the other project's output assembly.
3. The projects which have not been explicitly excluded are then Clovered so that the source files that go with the Clovered projects are suitably instrumented. Any additional files specified in the Clover options are copied to the build area at this time.
4. Based on project dependencies, the plugin determines the required build order. For each project, the plugin launches a separate non-interactive instance of Visual Studio to build the Clovered project. This approach ensures that the current solution is not disturbed by building a project outside its context. Clover.NET always builds the Debug configuration when building the Clovered projects.

The Clovered project files are standard Visual Studio project files and if you wish you may open them in a separate instance of Visual Studio.

6. NAnt tasks

6.1. Clover.NET NAnt tasks

6.1.1. Clover.NET NAnt Tasks

Clover.NET supplies a number of NAnt tasks to integrate Clover.NET into existing NAnt based build environments. There are presently two versions of these tasks - one for NAnt 0.84 and one for NAnt 0.85. Two versions are required due to changes in the NAnt internal APIs.

In general it is easiest to install the Clover.NET NAnt tasks into the same install directory as your main Clover.NET installation. The NAnt tasks may **not** be installed into your NAnt install's bin directory.

To make the tasks available in your build file you need to load the tasks manually. The following code in the top of your build file will achieve this:

```
        <property name="clover.home" value="C:\Program Files\Cenqua\Clover.NET"/>
        <loadtasks assembly="${clover.home}\CloverNAnt-0.84.dll"/>
```

There tasks are described in the following sections:

<clover-setup>	Sets up Clover.NET to be used when the <csc> task is executed
<clover-report>	Generates a Clover.NET coverage report

Please refer to the section on Adding Clover.NET to you NAnt project for some suggestions and tips.

6.2. Clover Setup Task

6.2.1. Description

The <clover-setup> task initializes Clover.NET for your project. In addition to telling Clover.NET where to find the coverage database, the setup task tells NAnt to use Clover.NET instrumentation when compiling C# code with the <csc> task when Clover is enabled.

6.2.2. Attributes

Attribute	Description	Required
initstring	The initString describes the location of the Clover.NET coverage database. Typically this is a relative or absolute file reference.	Yes
buildir	The location where the Instrumented source is written. If you do not specify a location a temporary location is used and the instrumented source is deleted after compilation.	No
enabled	This controls whether Clover.NET will instrument code during code compilation. This attribute provides a convenient control point to enable or disable Clover.NET from the command line	No
flushinterval	When set to a value, this enables flushing and the value is the minimum period between flush operations (in milliseconds)	No
flatten	Instructs Clover.NET to flatten the directory structure of the source tree in the CloverBuild area. If not set or set to false the source directory structure is replicated in the build area	No
instassigns	If false, Clover.NET will not instrument boolean expressions containing assignment statements as this instrumentation may cause the compiler to not be able to decide if an uninitialized variable has been used. Defaults to true.	No
keyfile	Location of the key file for signing assemblies. Normally Clover.NET can determine this	No

	without any additional infromation. In some cases, where the location is a constant, for example, it can be specified here to allow the Clovered build to be signed correctly.	

6.2.3. Nested Filesets

The <clover-setup> task supports one or more nested filesets which can be used to control which source files are Clovered during the compilation process. Files which are excluded are copied to the build area but are not instrumented.

6.2.4. Examples

This example shows a straight forward use of <clover-setup>

```
        <clover-setup initstring="CloverBuild\clover.cdb"
                      builddir="CloverBuild"
                      enabled="${clover.enabled}"
                      flushinterval="1000"
                      flatten="true"/>
```

In this example, the use of Clover.NETis controlled through the clover.enabled property. This allows the use of Clover to be controlled from the NAnt command line. Note also the use of a flushinterval to enable flushing every second. Finally, the use of the flatten attribute causes the Clover.NET instrumenter to write all the instrumented files into the CloverBuild directory regardless of their location in the source hierarchy

This example shows the use of nested filesets to control which classes are instrumented

```
        <clover-setup initstring="CloverBuild\clover.cdb"
                      flatten="false">
          <fileset basedir=".">
            <include name="**/*"/>
            <exclude name="**/nunit-gui/**/*"/>
            <exclude name="**/uikit/**/*"/>
            <exclude name="**/tests/**/*"/>
          </fileset>
        </clover-setup>
```

6.3. Clover Report Task

6.3.1. Description

Generates current reports in HTML or XML formats. The report task has no parameter attributes and all configuration occurs in the nested element. The nesting of elements within the <clover-report> task is as follows:

```
<clover-report>
  <current>
    <format/>
  </current>
</clover-report>
```

6.3.2. <current>

Generates a current coverage report. Specify the report format using a nested Format element. Valid formats are XML and HTML.

Attributes

Attribute	Description	Required
title	The title of the report	Yes
output	The output destination of the report. Reports can either generate a single file or a directory of related files.	Yes

6.3.3. <format>

Specifies the output format and various options controlling the rendering of a report.

Attributes

Attribute	Description	Required
type	The output format to render the report in. Valid values are `xml`, `html`.	Yes
bw	Specify that the report should be black and white. This will make HTML reports smaller (with no syntax hilighting)	No; defaults to "false"
orderBy	Specify how to order coverage	No; defaults to `PcCoveredAsc`

	tables. This attribute has no effect on XML format. Valid values are: **`Alpha`** Alphabetical. **`PcCoveredAsc`** Percentage coverage ascending. **`PcCoveredDesc`** Percentage coverage descending. **`ElementsCoveredAsc`** Number of elements covered, ascending. **`ElementsCoveredDesc`** Number of elements covered, descending. **`ElementsUncoveredAsc`** Number of elements uncovered, ascending. **`ElementsUncoveredDesc`** Number of elements uncovered, descending.	
noCache	(HTML only) if true, insert nocache directives in html output.	No; defaults to "false"
srclevel	if true, include source-level coverage information in the report.	No; defaults to "true"
showempty	If true, classes, files and packages that do not contain any executable code (i.e. methods, statements, or branches) are included in reports. These are normally not shown.	No; defaults to "false"
tabwidth	(Source level reports only) The spacing of tab stops. All tabs are expanded to to the next tabstop using spaces in reports.	No; defaults to 4

6.3.4. Examples

This example shows a minimal <clover-report> instance to produce an HTML report. Once this has run, the report will be available in the report directory.

```
          <clover-report>
            <current title="Test Report" output="report">
              <format type="html"/>
            </current>
          </clover-report>
```

This example produces a minimal report. It does not include source and the coverage bars are not rendered.

```
          <clover-report>
            <current title="test" output="report">
              <format type="html"
                      bw="true"
                      srclevel="false"
                      hidebars="true"/>
            </current>
          </clover-report>
```

6.4. Clover Check Task

6.4.1. Description

The <clover-check> task allows a build to ensure that particular coverage targets have been met.

6.4.2. Attributes

Attribute	Description	Required
target	The overall coverage target, expressed as a percentage	No
property	The name of a property which will contain the result of the coverage check.	No
messageProperty	The name of the property which contains a detailed report of which coverage targets have not been met.	No
haltOnFailure	If true the build will be stopped if coverage targets have not	No

	been met	

6.4.3. <namespace> Element

the namespace element allows you to specify coverage targets for specific namespaces.

Attributes

Attribute	Description	Required
name	The namespace name	Yes
target	the coverage target expressed as a percentage.	Yes

6.4.4. Examples

This example includes a check of overall coverage and the coverage of the NUnit.Core namespace

```
        <target name="check">
          <clover-check target="18%" haltOnFailure="false" property="covered" messagePr
            <namespace name="NUnit.Core" target="40%"/>
          </clover-check>

          <if test="${covered}">
            <echo message="Coverage Targets met" />
          </if>
          <ifnot test="${covered}">
            <echo message="Coverage target failed = ${coverage.message}"/>
          </ifnot>
        </target>
```

6.5. Adding Clover.NET Operations

6.5.1. Adding Clover.NET to your project

There are two approaches to adding Clover.NET to your existing NAnt-based build system. You can either add Clover.NET targets to your main build file or you can create a separate build file to manage just the Clover.NET related tasks. Both approaches are valid.

6.5.2. Updating your build file

When you add Clover.NET to your build file, we recommend the following steps:

1. Add the <loadtasks> call at the top of your build

   ```
               <loadtasks assembly="${Clover.home}\CloverNant-0.85.dll"/>
   ```

2. Add a target, "with-clover" for enabling Clover.NET instrumentation

   ```
               <target name="with-clover">
                 <clover-setup initstring="CloverBuild\clover.cdb"
                               builddir="CloverBuild"
                               flatten="true"/>
               </target>
   ```

3. Add a clover-report target to generate reports. Note the dependency between the report target and the with-clover target. This ensures the report target knows where the Clover.NET coverage database is located.

   ```
               <target name="clover-report" depends="with-clover">
                   <clover-report>
                     <current title="test" output="report">
                       <format type="html"/>
                     </current>
                   </clover-report>
               </target>
   ```

With the above changes in place you can continue to use your build file as before for normal builds. To build with Clover, you specify the with-clover target:

```
NAnt with-clover dist
```

For reports, you use the clover-report target. If you wish, you can add more dependencies to the clover-report target to build and run tests with the Clovered build.

6.5.3. Auxilliary Build File

The other approach to adding Clover.NET to your project is to create an auxilliary build file with just the Clover.NET related targets. Your main build file is left as is.

```
        <project default="build">
          <echo message="building with Clover"/>
          <property name="Clover.home" value="Install Directory"/>

          <loadtasks assembly="${Clover.home}\CloverNant-0.85.dll"/>
          <clover-setup initstring="CloverBuild\clover.cdb"
                        builddir="CloverBuild"
                        flatten="true"/>

          <target name="build">
            <nant buildfile="main.build"/>
```

```
  </target>

  <target name="report">
    <clover-report>
      <current title="test" output="report">
        <format type="html"/>
      </current>
    </clover-report>
  </target>
</project>
```

As shown in the example, the clover.build build file enables Clover for all operations. For build operations it delegates to your main.build build file, which will Instrument code prior to compilation.

7. Clover.NET Usage

7.1. Source Directives

Clover supports a number of directives that you can use in your source to control instrumentation. Directives can be on a line by themselves or part of any valid single or multi-line comment.

7.1.1. Turning Instrumentation On and Off

```
        CLOVER:ON
        CLOVER:OFF
```

These directives enabled and disable Clover instrumentation. This is useful if you don't want Clover to instrument a section of code for whatever reason. Note that the scope of this directive is the current file only.

7.1.2. Force Clover to flush

```
        CLOVER:FLUSH
```

Clover will insert code to flush coverage data to disk. The flush code will be inserted as soon as possible after the directive.

7.2. Assembly Signing

7.2.1. Assembly Signing

In many cases, .NET assemblies are signed using a key. Often this is specified by a relative path to a key file in an AssemblyInfo.cs file such as:

```
        [assembly: AssemblyKeyFile("..\\..\\..\\project.key")]
```

The Clover Runtime assembly is strongly named, so Clovered assemblies may be signed. If the original assembly is signed, the Clovered assembly will also be signed. When Clovering, Clover.NET will replace any relative key file references with an absolute file path. This ensures that the assembly can be signed even if it is created at a different relative location from the key file.

7.3. Environment Variables

7.3.1. Environment Variables

Clover.NET supports two environment variables:

- CLOVER_DB

  Setting this environment variable allows you to override the location of the Clover.NET coverage database. This can be used, for example, if you move the executable for testing purposes, to another machine. It would also allow the Clovered executable to be used on a different platform, such as Mono.

- CLOVER_DEBUG

  Setting this environment variable to any value will cause the Clover Runtime to append debug information to a file clover.log along side the coverage database. At present, this just records the location of the assembly which creates a coverage recorder.

8. Miscellaneous

8.1. Frequently Asked Questions

8.1.1. Questions

8.1.2. Answers

1. General

1.1. Can't find an answer here?

Try our Online Forums, or contact us directly.

2. NAnt Tasks

2.1. Why can't I install the NAnt tasks into the NAnt bin directory?

NAnt tasks are named by a .NET attribute, the TaskName attribute. The Clover.NET csc tasks works by replacing the csc task and then delegating operations to the original csc task.

To do this the Clover version of the task must have a TaskName attribute of csc. If this were to be deployed in the NAnt bin directory, it would collide with the standard csc task. NAnt does not currently allow a task to be registered under a name other than the name supplied by the TaskName attribute.

3. Visual Studio.NET Plugin

3.1. I have installed the Clover.NET plugin for Visual Studio and it does not appear or it appears and all of the menus are greyed out.

The Clover.NET plugin requires the Microsoft VSIP interop assemblies to be installed. the installer for these is included in the Clover.NET download and should be installed prior to installing Clover.NET. If you have doen that and you still have issues getting the Clover.NET plugin to work, please go to thr install directory and type in:

```
CloverReg /root:Software\Microsoft\VisualStudio\7.1 CloverPackage.dll
```

If you receive any errors, please contact support for further assistance.

3.2. Why are projects which I have selected as excluded still copied to the Clover build area?

Even when a project is excluded from Clovering, Clover.NET needs to update any inter-project dependencies. For example, if a project you've excluded depends on one you've decided to Clover, Clover.NET needs to update the project dependency to point to the Clovered version. It does this in a copy of the excluded project. So, when you select a project for exclusion, Clover.NET will still copy it to the CloverBuild area but it will not instrument the code. Clover.NET will still make some changes to the code to update an relative key file locations used in assembly signing.

3.3. Is Visual Studio.NET 2005 supported?

At this time, only Visual Studio.NET 2003 is supported.

4. Problems with Clover

4.1. I have Clovered my code and run my tests but when I generates reports my coverage shows as 0% and there are no coverage recordings

The Clover.NET runtime writes out a coverage recording when the Application Domain in which it is running is unloaded. If you run your tests in NUnit, you will need to exit NUnit (or load another test suite). When you do so, the coverage recording is written and you can then load the coverage results or generate a coverage report.

An alternative is to enable flushing in your Clovered code so that the coverage is written out at regular intervals.

4.2. How do I Clover an ASP.NET application

As with the previous question, you need to have ASP.NET unload the AppDomain it is using for your webapp. When that unload occurs, the coverage is written out. The tricky part is getting ASP.NET to unload your webapp. This can be done by either deploying the unClovered version of your webapp or deleting the Clovered assemblies. ASP.NET will notice the change in the webapp assembly and unload the Clovered version.

4.3. When my Clovered ASP.Net web application starts up, it cannot find the Clovered database even though I can see it on the disk.

If your Visual Studio project is under your "My Documents" folder, and you accept the default Clover.NET database location, the generated coverage database will not be available to the ASP.NET process since its "My Documents" is actually a different folder in the file system. If you set the Clover database location to a location available to the ASP.NET user, you should have no problems.

5. Technical Background

5.1. How are the Clover.NET coverage percentages calculated?

The "total" coverage percentage of a class (or file, namespace, project) is provided as a quick guide to how well the class is covered - and to allow ranking of classes. The Total Percentage Coverage (`TPC`) is calculated using the formula:

```
TPC = (CT + CF + SC + MC)/(2*C + S + M)

where

CT - conditionals that evaluated to "true" at least once
CF - conditionals that evaluated to "false" at least once
SC - statements covered
MC - methods entered

C - total number of conditionals
S - total number of statements
M - total number of methods
```