OpenCover Usage Guide

Intro

The following guide describes how to use OpenCover (also available on NUGET and Chocolatey) to gather coverage statistics of your application.

a) Can handle 32 and 64 bit .NET processes running on the .NET 2 and .NET 4+ frameworks.
b) Will usually handle .net core applications, however only latest versions supported e.g. 2.2, 3.1 and 5.0.
c) Will gather sequence and branch coverage information of your assemblies that match the filters and for which the PDB files can be found.
d) Can gather coverage reports of Silverlight applications
e) Can gather coverage reports of Windows Service applications.
f) Can record which tests where executing at a particular time when a coverage point was visited – only MSTest and NUnit supported (requests accepted for others).

Currently OpenCover has no full presentation of results other than the XML output file; ReportGenerator (also available on NUGET) is currently the recommended tool for visualizing the results.

NOTE: When there is no PDB for an assembly then no coverage data will be gathered; this is different to PartCover which will default to IL coverage under this situation but it was considered as not required as this is supposed to be a code-coverage tool which can relate such coverage to ***your*** code.

Table of Contents

Command Arguments

OpenCover has a number of arguments that can be used to control the code coverage gathering. If an argument requires spaces then use "'s to wrap the argument, where they are applicable they will be indicated with an optional syntax [].

Mandatory

["]-target:<target application>["]

> The name of the target application or service that will be started; this can also be a path to the target application.

Alternatively use *-?* to show command line help, or *-version* to print the current version and exit.

Optional

["]-targetdir:<path to the target directory>["]

> The path to the target directory; if the target argument already contains a path then this argument can be used to provide an alternate path where PDB files may be found.

["]-targetargs:<arguments for the target process>["]

> Arguments to be passed to the target process.

[“]-searchdirs:<path to pdbs>[;<path to pdbs>[“]

Additional locations to check for PDB files.

-register[:user|path32|path64]

Use this switch to register and de-register the code coverage profiler. Alternatively use the optional *user* argument to do per-user registration where the user account does not have administrative permissions.

If access to registry is limited then try the *path32* or *path64*, depending on your application (32- or 64-bit), to use an alternate method to load the profiler; unfortunately you cannot profile 32- and 64-bit processes at the same time using these switches.

You should also consider using *path32* and *path64* options when profiling .net core targeted applications and .net 4.8 framework.

Alternatively use an administrative account to register the profilers using the regsvr32 utility; this is the recommended option when running on a build server especially if *–register:user* does not work.

-returntargetcode[:<opencoverreturncodeoffset>]

Return the target process return code instead of the OpenCover console return code. Use the offset to return the OpenCover console at a value outside the range returned by the target process.

-safemode:on|off|yes|no

Use this switch to disable safe mode (default is **on**/yes). When in safe mode the profiler will use a common buffer for all threads which may have performance impacts if you code or tests use threads heavily. When safe mode is disabled, there may on occasions be some data loss if the runtime closes the application under profile before the profiler has been able to retrieve the visit count data.

["]-output:<path to file>["]

The location and name of the output xml file. If no value is supplied then the current directory will be used and the output filename will be results.xml.

-threshold:<max visit count>

Limits the number of visit counts recorded/reported for an instrumentation point. May have some performance gains as it can reduce the number of messages sent from the profiler. Coverage results should not be affected but will have an obvious impact on the Visit Counts reported.

["]-filter:<space separated filters>["]

A list of filters to apply to selectively include or exclude assemblies and classes from coverage results. Filters have their own format **±[module-filter]class-filter**. If no filter(s) are supplied then a default include all filter is applied +[*]*. As can be seen you can use an * as a wildcard.

NOTE: Also an *exclusion* (-) filter takes precedence over an *inclusion* (+) filter.

-regex

Filters are supplied using regular expressions rather than wildcards.

-nodefaultfilters

A list of default exclusion filters are usually applied, this option can be used to turn them off. The default filters are:

-[mscorlib]*
-[mscorlib.*]*
-[System]*
-[System.*]*
-[Microsoft.VisualBasic]*

-mergebyhash

Under some scenarios e.g. using MSTest, an assembly may be loaded many times from different locations. This option is used to merge the coverage results for an assembly regardless of where it was loaded assuming the assembly has the same file-hash in each location.

-skipautoprops

Neither track nor record Auto-Implemented properties.

i.e. skip getters and setters like these

```
public bool Service { get; set; }
```

-showunvisited

Show a list of unvisited methods and classes after the coverage run is finished and the results are presented.

[“]-excludeddirs:<path to exclude>[;<path to exclude>[“]

Any assembly found in an excluded folder (or its children) will be ignored, regardless of any inclusive filter matches.

-excludebyattribute:<attrfilter>[;< attrfilter>][;< attrfilter>]

Exclude a class or method by filter(s) that match attributes that have been applied that have been applied. An * can be used as a wildcard.

e.g. -excludebyattribute:*.ExcludeFromCoverageAttribute

-excludebyfile:<filefilter>[;< filefilter>][;< filefilter>]

Exclude a class (or methods) by filter(s) that match the filenames. An * can be used as a wildcard.

-hideskipped:<skip-value>[|<skip-value>]

Remove information from output file (*-output:*) that relates to classes/modules that have been skipped (filtered) due to the use of the following switches *–excludebyfile:*, *-excludebyattribute:* and *–filter:* or where the PDB is missing.

<skip-value> can be one or more of the following values separated by the | symbol:

File | Filter | Attribute | MissingPdb | Inferred | AutoImplementedProperty | NativeCode | FolderExclusion | Delegate | All

e.g. -hideskipped:File|Filter|Delegate

-coverbytest:<dllfilter>[;< dllfilter>][;< dllfilter>]

Gather coverage by test by analysing the assemblies that match these filters for Test methods. Currently only MSTest and NUnit tests are supported; other frameworks can be added on request – please raise support request on GitHub.

e.g. -coverbytest:*.Test.dll

-log:[Off|Fatal|Error|Warn|Info|Debug|Verbose|All]

Change the logging level, default is set to Info. Logging is based on log4net logging levels and appenders.

-service

NOTE: "Administrator" privileges recommended.

The value provided in the target parameter is the name of a service rather than a name of a process.

-servicestarttimeout:[1m|23s|1m23s]

Overrides the default time to wait for the profiled service to start. The examples above correspond to a timeout of 1 minute, 23 seconds and 1 minutes and 23 seconds accordingly.

-oldstyle

Use old style instrumentation – the instrumentation is not Silverlight friendly and is provided to support environments where mscorlib instrumentation is not working.

-communicationtimeout:<timeout-ms>

Determines how long the profiler and the host wait for communication to complete before failing. Defaults to 10s. Max to 60s. Only considering increasing if the assembly being profiled is extremely large.

-enableperformancecounters

NOTE: “Administrator” privileges required.

Allows the monitoring in “Performance Monitor” of the following values:

1) “messages remaining on the queue”
2) “number of messages processed”

These values are usually cleared at the end of a performance run.

Handling Spaces

If your argument needs to escape quotes i.e. to pass arguments with spaces to the target process then you can use \".

e.g.

```
-targetargs:"\"c:\program files\" arg2 arg3"
```

Or

```
"-targetargs:\"c:\program files\" arg2 arg3"
```

Understanding Filters

Filters are core to understanding how OpenCover works and how it is determined which assemblies are to be instrumented to provide coverage results.

Filters can be inclusive and exclusive represented by + and – prefix respectively, where exclusive (-) filters take precedence over inclusive (+) filters.

The next part of a filter is the module-filter and usually this happens to be the same name as the assembly but without the extension and this rule will normally apply 99.999% of the time. If this filter isn’t working look in the coverage XML and compare the found <ModuleName/> entries against the filter.

The final part of the filter is the class-filter and this also includes the namespace part of the class as well.

Examples

`+[Open.*]* -[Open.Test]*`
Include all classes in modules starting with Open.* but exclude all those in modules Open.Test

`+[Open]* -[Open]Data.*`
Include all classes in module Open but exclude all classes in the Data namespace.

`+[Open]* -[Open]*Attribute`
Include all classes in module Open but exclude all classes ending with Attribute.

Note: These filters are case sensitive.

Regular Expressions in Filters

It is also possible to use regular expressions instead of wildcards but to do so require that you use the –regex switch when specifying the filters. NOTE: When using this feature it is required that all filters use regular expressions rather than wildcards.

Examples

```
+[(Open\..*)](.*) -[(Open\.Test)](.*)
```

Include all classes in modules starting with Open.* but exclude all those in modules Open.Test

```
+[(Open)](.*) -[(Open)](Data\..*)
```

Include all classes in module Open but exclude all classes in the Data namespace.

```
+[(Open)](.*) -[(Open)](.*Attribute)
```

Include all classes in module Open but exclude all classes ending with Attribute.

Running against IIS

Normally I'd suggest running against IISEXPPRESS as I think it is easier to automate. However for those who really want to run against a full blown IIS then the following instructions (supplied by a user) will hopefully suffice.

"The trick is to start OpenCover to run the w3wp.exe process in debug mode e.g.

```
OpenCover.Console.exe -target:C:\Windows\System32\inetsrv\w3wp.exe -targetargs:-debug
-targetdir:C:\Inetpub\wwwwoot\MyWebApp\bin\ -filter:+[*]* -register:user
```

There are some prerequisites though:

1. All applications running under the site must make use of the same app pool; you'll get errors in the EventLog otherwise.
2. inetserver needs to be stopped, before starting w3wp.exe in debug mode. You can use the following command:

   ```
   net stop w3svc /y
   ```

   After testing/code coverage completion you can close the w3wp.exe process and start the inetserver again:

   ```
   net start w3svc
   ```

This procedure was tested on a Win2008 machine with IIS7.5"

You can also run multiple OpenCover instances against separate IIS sites by using the *–s* option when running IIS to choose the siteid e.g.

```
OpenCover.Console.exe -target:C:\Windows\System32\inetsrv\w3wp.exe
        -targetargs:"-debug -s 1"
        -targetdir:%WebSite_Path%
        -filter:+[*]*
```

```
-register:user
-output:%CoverageResult_Path%
```

Then you can use ReportGenerator to merge the coverage results.

Running against an application

This most common usage of any code coverage utility such as OpenCover is in a testing environment

Sample

```
OpenCover.Console.exe -register:user -target:..\..\..\tools\NUnit-2.5.10.11092\bin\net-2.0\nunit-console-x86.exe -targetargs:"OpenCover.Test.dll /noshadow" -filter:"+[Open*]* -[OpenCover.T*]*" -output:opencovertests.xml
```

Running against a Silverlight application

To run against a Silverlight application it is necessary to ensure the site hosting the application is running beforehand. To profile a Silverlight application it is necessary to launch a browser against the site and as the PDB files are not packaged in the XAP files it is necessary to give the console a hint where to look for the PDB files (using the *–targetdir* option).

Sample

```
OpenCover.Console.exe -register:user "-target:C:\Program Files (x86)\Internet Explorer\iexplore.exe" "-targetargs:http://localhost:4128/SampleSilverlightTestPage.aspx "-targetdir:..\SampleSilverlight\SampleSilverlight\Bin\Debug"
```

Running against a Service application

It is preferable to run the service in a console mode if it has one rather than as a service however if you do decide to use it against a service then you will need to make sure you use an account that can access the windows synchronisation objects in the Global namespace (rather than Local namespace). “Local System” seems to work quite well and so do user accounts with the appropriate permissions; “Local Service” is usually problematic and is not recommended. The console host will also need to be run from an account that can access the Global namespace as such an Administrator account or an Administrative prompt is recommended.

Sample

```
OpenCover.Console.exe -target:"OpenCover Sample Service" -service –register
```

NOTE: Rather than use the *–register* switch, it is usually simpler to use the *regsvr32* utility to pre-register the two profiler assemblies (32 and 64-bit) beforehand.

Using the *-excludebyattribute* option

Normally you would include/exclude modules and classes by using the inclusion/exclusion filters, however there may be situations where you can’t get coverage via testing and you wish to ignore the uncovered method.

First create a “public” attribute that you can apply to class/method/property which you use to mark up something to ignore. You can have more than one and you can add other data to provide a reason why you are excluding it.

e.g.

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Method|AttributeTargets.Property)]
public class ExcludeFromCoverageAttribute : Attribute{}
```

Then you apply this attribute to the class/method/property that you wish to exclude.

Then you add this attribute to the *-excludebyattribute* option using namespaces and wildcards where necessary.

e.g.

```
-excludebyattribute:*.ExcludeFromCoverage*
```

NOTE: Use with care as you could exclude a method which you should be testing; also it can become too tempting to ignore a method and not test due to it being difficult and use this option to 'skip' it.

Using the *-excludebyfile* option

This is a useful option to use to ignore auto-generated files. This works on file and pathnames.

e.g. the following would ignore all code in files ending in *generated.cs*

```
-excludebyfile:*\*.generated.cs
```

NOTE: Use with care as you could exclude a method which you should be testing; also it can become too tempting to ignore a method and not test due to it being difficult and use this option to 'skip' it.

Shimming support

> In computer programming, a shim is a small library that transparently intercepts API calls and changes the arguments passed, handles the operation itself, or redirects the operation elsewhere. Shims typically come about when the behavior of an API changes, thereby causing compatibility issues for older applications which still rely on the older functionality. In such cases, the older API can still be supported by a thin compatibility layer on top of the newer code. Web polyfills are a related concept. Shims can also be used for running programs on different software platforms than they were developed for.
>
> - wikipedia

Depending on the provider of the Shimming utility will determine on how the OpenCover will be used alongside it:

Microsoft Moles support

To use Moles with OpenCover requires that you first inform Moles that you are using OpenCover.

- Before you run moles you need to set an environment variable
  `set CLRMONITOR_EXTERNAL_PROFILERS={1542C21D-80C3-45E6-A56C-A9C1E4BEB7B8}`
- Then use OpenCover to run the moles runner

Microsoft Fakes support

OpenCover has support for Microsoft Fakes just use OpenCover to execute vstest.console.exe and it will detect if the Microsoft Fakes profiler is going to be activated and it will do the rest.

TypeMock support

The developers at TypeMock added OpenCover support several years ago; please review their documentation to get both TypeMock and OpenCover to work correctly with the versions you have installed.

JustMock support

The developers at JustMock have also added support for OpenCover; please review their documentation to get both JustMock and OpenCover to work correctly with the versions you have installed.

Build system integration

It is not unexpected that OpenCover will be used in a build environment and that the build will be running under a system account under these scenarios it is recommended that you pre-register the profiler DLLs using the regsvr32 utility where applicable for your environment.

```
regsvr32 x86\OpenCover.Profiler.dll
regsvr32 x64\OpenCover.Profiler.dll
```

To assist your build environment when you install OpenCover using the MSI it will store in the registry a location of the installation folder. The location in the registry depends on whether it is a single-user or an all-user installation and also if you are on a 32/64 bit environment.

See the following examples based on default settings:

all-users (32-bit)

Registry Entry: `HKLM\Software\OpenCover\Location`
Install Location: `%PROGRAMFILES%\OpenCover`

all-users (64-bit)

Registry Entry: `HKLM\Software\Wow6432Node\OpenCover\Location`
Install Location: `%PROGRAMFILES(X86)%\OpenCover`

single-user

Registry Entry: `HKCU\Software\OpenCover\Location`
Install Location: `%LOCALAPPDATA%\Apps\OpenCover`

Reporting

It is recommended that ReportGenerator (also available on Nuget) is used to view the coverage results however if you want to make your own reporting then a sample XSLT has been made available by Pavan Tiwari (https://github.com/pawan52tiwari). It is simple to use with the supplied powershell script.

```
powershell -noexit -file ..\..\transform\transform.ps1 -xsl ..\..\transform\simple_report.xslt
-xml opencovertests.xml -output simple_output.html
```

Feel free to extend it to your own requirements.

FAQ

Why do I have no results?

There are two common reasons why this may happen.

1) Instrumentation skipped due to filters.

The usual reason for no results because OpenCover cannot locate the PDBs for assemblies that match the filters to be profiled i.e. gather coverage results from. When each assembly is loaded the location and reason the assembly wasn't profiled is provided in the coverage results file e.g.

```
<Module skippedDueTo="Filter" hash="0C-69-37-C2-8F-AB-FC-E7-72-51-E9-17-19-99-1A-4A-4C-07-72-08">
  <FullName>C:\Personal\opencover.git\working\main\bin\Debug\OpenCover.Test.dll</FullName>
  <ModuleName>OpenCover.Test</ModuleName>
  <Classes />
</Module>
```

The two most common reasons provided are `Filter` and `MissingPdb`.

`Filter` is obviously connected to the *–filter:* argument and that the ModuleName was not matched.

`MissingPdb` is usually because the PDB is not where the assembly is being loaded from. The most common reason is due to test tools such as NUnit or MSTest which copy the assembly under test to a new location; this can be corrected by either using the /noshadow or /noisolation option. An alternative is to use the *–targetdir:* argument to provide an alternative location for OpenCover to use.

The other reasons are only applicable to classes and are related to the use of *–excludebyattribute:* and *–excludebyfile:* options.

2) Failure to register the profiler assemblies.

The profiler assemblies are COM objects and need to be registered in the Registry before the target process is run.

This can usually be solved in one of two ways:

a) Use the *–register[:user]* switch
b) Pre-register the assemblies using the regsvr32 utility

All my tests are failing and I am getting MissingMethodException

This has been seen on a few systems where the following command has been executed to improve performance (or something similar)

```
ngen install /Profile "mscorlib"
```

There are two ways to fix or handle this issue

1) Undo the previous command - `ngen uninstall /Profile "mscorlib"`.
2) Use the *–oldstyle* switch, note however that this is not Silverlight friendly.