

Exception Hunter 2.0

August 2009

Contents

Getting started	3
Analyzing your code	4
Working with assemblies	5
Viewing the results	10
Generating reports	14
Using the command line	15
Examples using the command line	17
Acknowledgements	20

Getting started

Exception Hunter analyzes your .NET assemblies to detect the unhandled exceptions that can be thrown by a particular method.

This is useful, for example, to find all the possible exceptions that might be thrown by a selected method and view the relevant stack trace to identify where those exceptions originate. Exception Hunter allows you to navigate easily through the assembly, and also provides a powerful search to find methods, classes, and other components of your code. You can then view the source code to find the line of code that throws an exception.

Exception Hunter can analyze any .NET assemblies from version 1.1 to version 4.

Analyzing your code

To start analyzing your code:

1. Add the assemblies.

When you first start Exception Hunter, you need to add the assemblies (page 5) that you want to analyze. If another assembly is referenced by an added assembly, it is added automatically. Any referenced assembly that cannot be found on your file system is identified as "Not Found"; you can then browse to locate it, or ignore it in the analysis.

2. Locate the method you want to analyze.

To select the method you want to analyze, you can search for a method (page 5) using the **Find** box, or drill down (page 5) to view all namespaces, classes, structs, and their methods.

3. View the results.

Exceptions are listed by type. You can explore the list of exceptions by selecting an exception type to see all the places in your code that the exception is thrown.

4. Drill down through the stack trace for the selected exception class.

To find situations in which the exception may be thrown, view the source code of the method selected in the stack trace.

5. Adjust the options, if required.

From the **Tools** menu, select **Options** to display the Options dialog box, in which you can set a number of options for how Exception Hunter analyzes your code. For example, you can set the version of the .NET Framework for detecting exceptions, or use a more detailed analysis, which detects more exceptions but can take longer to run. Hints are available for each option in the Options dialog.

Notes

- Exception Hunter cannot detect exceptions that may be thrown when following delegate calls, for example Event Handler calls.
You should therefore analyze the target methods for such delegates. We recommend that you wrap any exceptions and throw them as a domain-specific exception type.
- Static classes appear in the list as abstract sealed classes, as this is how they are represented by the .NET CLR.
- Runtime Exceptions (other than some NullReferenceExceptions and InvalidCastExceptions) generated by the .NET CLR are not detected by Exception Hunter.


Working with assemblies

This topic describes how to load .NET assemblies into Exception Hunter for analysis, how to remove assemblies and how to locate assemblies that cannot be found. It also explains how to navigate through your code and search for a method to analyze.

Adding the assemblies

Before you add the assemblies to analyze, we recommend that you specify the version of the .NET Framework you want to analyze them against. Changing the .NET Framework option *after* adding the assemblies will remove the list of loaded assemblies. To specify the version of .NET, from the **Tools** menu select **Options**. The default setting is version 2.0; you do not need to change the option if you use version 2.0.

When you start Exception Hunter for the first time, the main window is empty. Click  **Add Assembly** to display the the **Add Assembly** dialog box, and select the assemblies to analyze.

Alternatively, if your project is an ASP.NET web application, click  **Add ASP.NET Page** to select any ASP.NET component from one of the following sources:

- web pages (*.aspx)
- controls (*.ascx)
- web services (*.asmx)





Exception Hunter compiles and then analyzes the selected component, as well as the rest of its web application.

If you have used Exception Hunter previously, the assemblies that were added when you closed the application are listed.



Working with loaded assemblies

All the assemblies that are referenced by assemblies you have added are loaded automatically. For some large assemblies, the loading may take a few seconds; a message and a green progress indicator is displayed in the status bar while the assemblies are loading.






The icon next to the assembly name indicates its status:

-  **OK** The assembly has been successfully loaded. Assemblies shown in bold type are those that you specifically selected.
-  **OK** The assembly has been successfully loaded. Assemblies shown in normal type are referenced assemblies.
-  **Invalid** An assembly is identified as invalid if it is not a .NET assembly, or if the file is corrupt or inaccessible. Right-click on the assembly and select  **Remove assembly** to delete it from the

list.

 Not found The assembly is referenced by one of the other assemblies, but it cannot be located in the file system. Right-click on the assembly name and select  **Locate assembly** to specify a location from where it can be loaded.

You can view a tooltip for any assembly name that is too long to display in full:

 OK	Accessibility	2.0.0.0	neutral
 OK	System.Runtime.Serialization...	2.0.0.0	neutral
 OK	System.Deployment	2.0.0.0	neutral
 OK	DevExpress.Data	System.Runtime.Serialization.Formatters.Soap	
 OK	System.Data	This .NET assembly has been successfully loaded.	

Showing only documented exceptions

As an alternative to performing analysis on the referenced assemblies, Exception Hunter can show only the exceptions each method is documented to throw, by reading documentation from an XML file provided with the assembly. This can make analysis faster, and may reduce irrelevant exceptions. However, the results may not include all exceptions if the XML file is incomplete.

This option is only available for assemblies where an XML file is present.

To choose to show only documented exceptions from an assembly:

- select **Use XML File** next to the assembly

Assembly Name	Use XML File
...eal\Test4vb.exe	<input type="checkbox"/>
...cuments and Settin...	<input checked="" type="checkbox"/>
...INDOWS\Microsoft...	<input type="checkbox"/>
...INDOWS\Microsoft...	<input checked="" type="checkbox"/>
...INDOWS\Microsoft...	<input type="checkbox"/>

Resolving missing assemblies

When you add an assembly to Exception Hunter, it may reference a number of other assemblies. If the referenced assembly cannot be located on your computer, you can specify its location. Note that if you do not resolve missing assemblies at this stage, you will be prompted to locate any missing assembly when you select a method to analyze.

To specify the location of a missing assembly:

1. Right-click on the assembly name and select **Locate Assembly**.
2. In the **Locate Missing References** dialog box, all the unresolved assemblies are listed. Select the assembly you want to find then click **Locate Assembly**.

- In the **Locate Assembly** dialog box, browse to the folder containing the assembly, select the file and click **Open**. Note that any other assemblies in this folder will also be resolved automatically.

If you do not want to include a missing assembly in the analysis, or you cannot locate it, you can choose not to be prompted for that missing assembly again. In the **Locate Missing References** dialog box, select **Don't ask again** next to the assembly:



Removing assemblies

To remove an assembly, right-click on it and select **Remove Assembly**.

To remove all assemblies, on the **File** menu click **Remove All Assemblies**; alternatively you can right-click on any assembly in the list, then click **Remove All Assemblies**.

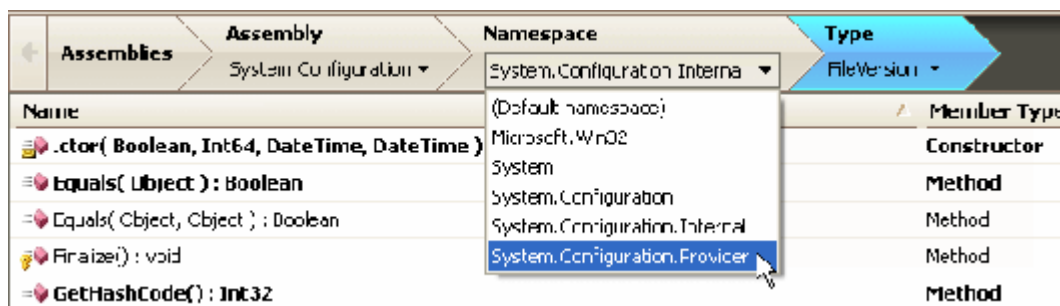
Navigating through an assembly

As you navigate through the assembly by selecting a namespace, then a type (class or struct) and then a method, the navigation bar shows how you have drilled down:



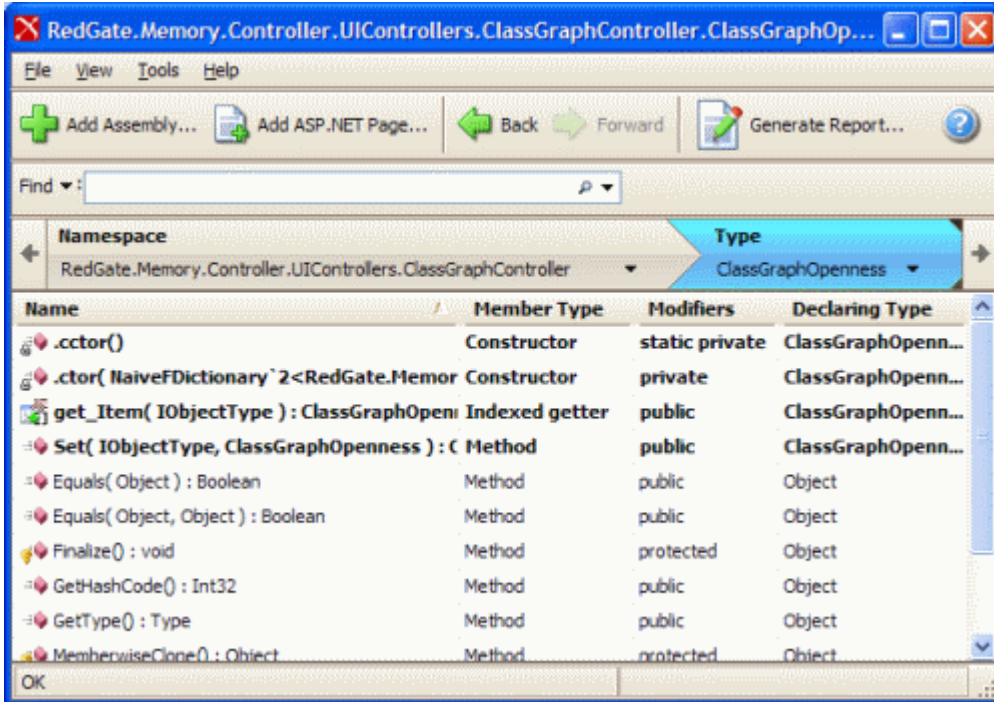
You can drag the navigation bar to see any names that are not visible; when the pointer changes to a hand, click and drag the bar to the left or right. You can also use the arrows at the left and right of the bar to scroll it.

- The current selection category (for example **Method**) is added to the right of the navigation bar and highlighted in blue.
- You can go back to a previous category by clicking on it. For example, you can click **Assembly** to view all the namespaces for that assembly.
- You can use the drop-down list within any of the displayed categories to change the selection. For example, if you are currently looking at the list of methods for a selected class, you can go back to select a different namespace and view the types for that namespace.



- You can use the Back and Forward buttons to return to the previous/next view.

The members available for the selected type are listed alphabetically by default. Methods in bold are those that are called by the object class itself rather than by a super-type of the class, and are listed first by default.



You can sort the methods by clicking on a column heading, for example to arrange the methods by declaring type or modifiers.

Click on a method to start analyzing it for unhandled exceptions. Any missing assemblies that you have not told Exception Hunter to ignore will be displayed in the **Locate Missing References** dialog box before the results are displayed.

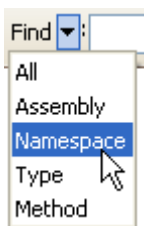
The exceptions that have been found are listed. For details, see Viewing the results.

Using the Find box

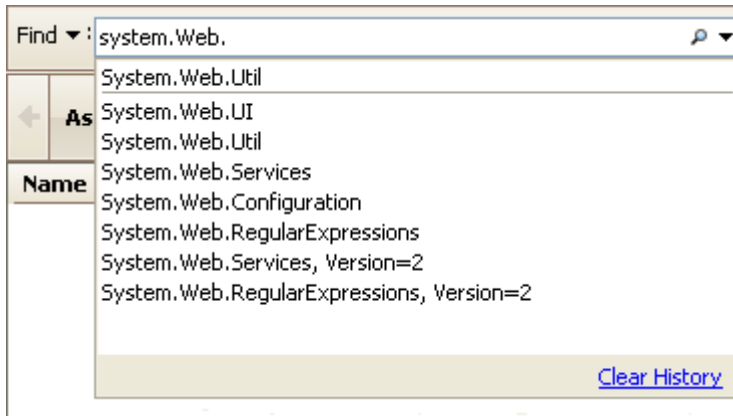
You can use the **Find** box to search for a method or other element of your code across all the loaded assemblies.




From the drop-down list on the left, you can limit the search to the category you want:



Type the first few letters of the term you want to search for. As you type, Exception Hunter suggests a word to auto-complete the term based on the contents of the loaded assemblies; press TAB to accept the suggestion. Type a full stop to specify that the first part of the search term is a namespace or a class.



Exception Hunter also lists search terms that match what you type. To use a suggested search term, click on a term in the list, or use the down arrow key to highlight it and then press Enter.

If none of the suggested terms are suitable, type the characters you want to search for and click .

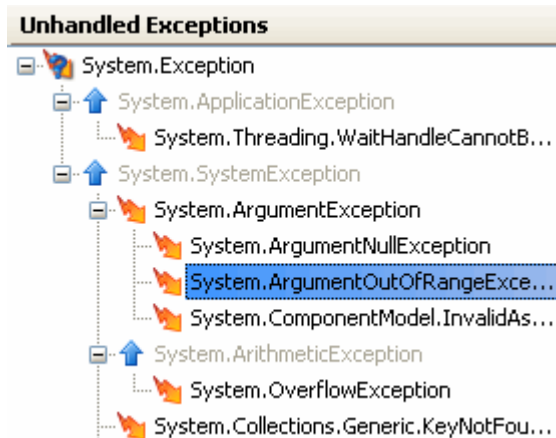
The results are displayed in the **Results** panel at the bottom of the window. You can then:

- click on a heading
This is a good way to sort by method or class, for example. Use Shift+Click to sort by more than one heading.
- from the **Find** box, change the filter by selecting from the drop-down list
For example, if you initially searched using the **Namespace** filter, and then select **Method** from the drop-down list, the results are updated to show all methods matching the search term.

Click on a method in the **Results** pane to start analyzing it for unhandled exceptions. Any missing assemblies that you have not told Exception Hunter to ignore will be displayed in the **Locate Missing References** dialog box before the analysis is performed. See Viewing the results.

Viewing the results

When you select a method (page 5), Exception Hunter lists the different classes of unhandled exceptions that have been found, and displays a stack trace for each. The exceptions are sorted by class.



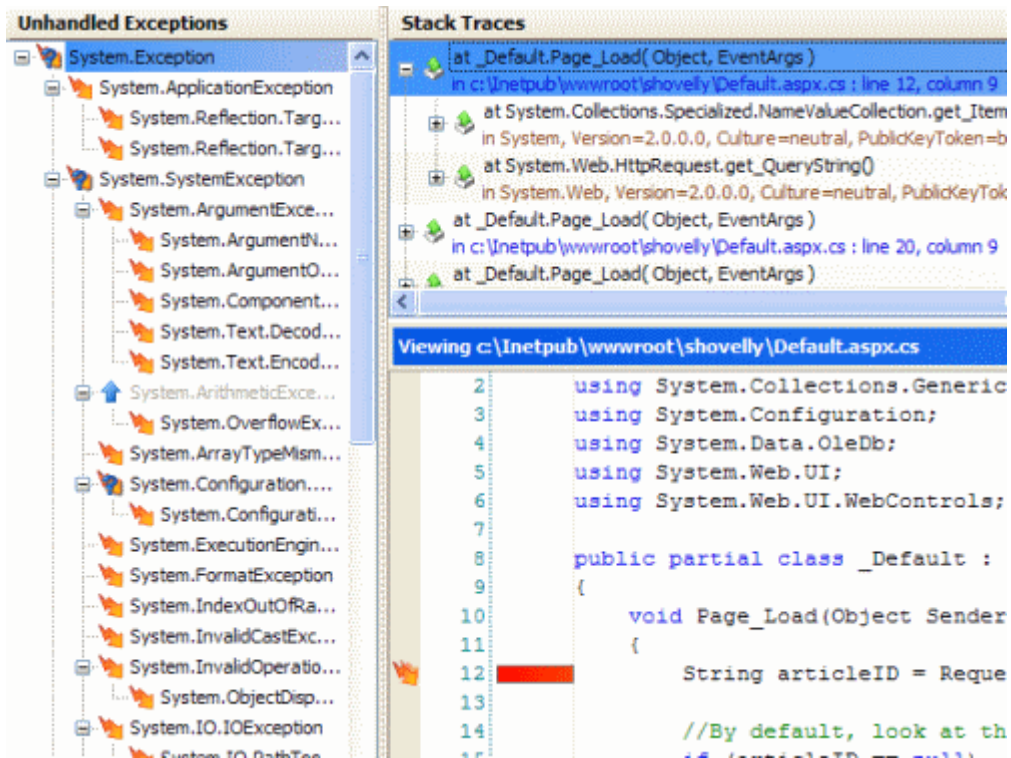
Note that some types of exceptions may not be listed, depending on the options you have selected. For example, if you have cleared the **Detect NullReferenceExceptions** and **Detect InvalidCastExceptions** check boxes in the **Options** dialog box, then these type of exceptions are not be listed (unless explicitly thrown by a method).

Some exceptions are shown in gray. These exceptions are not thrown directly, but they are superclasses of exceptions that are thrown.

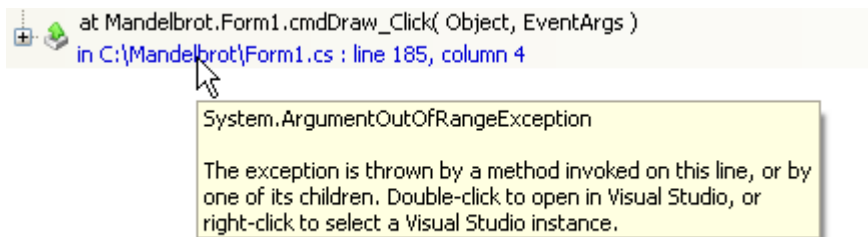
Move the mouse pointer over an exception class to see a tooltip that tells you how many stack traces are available for that exception.

Viewing the stack trace

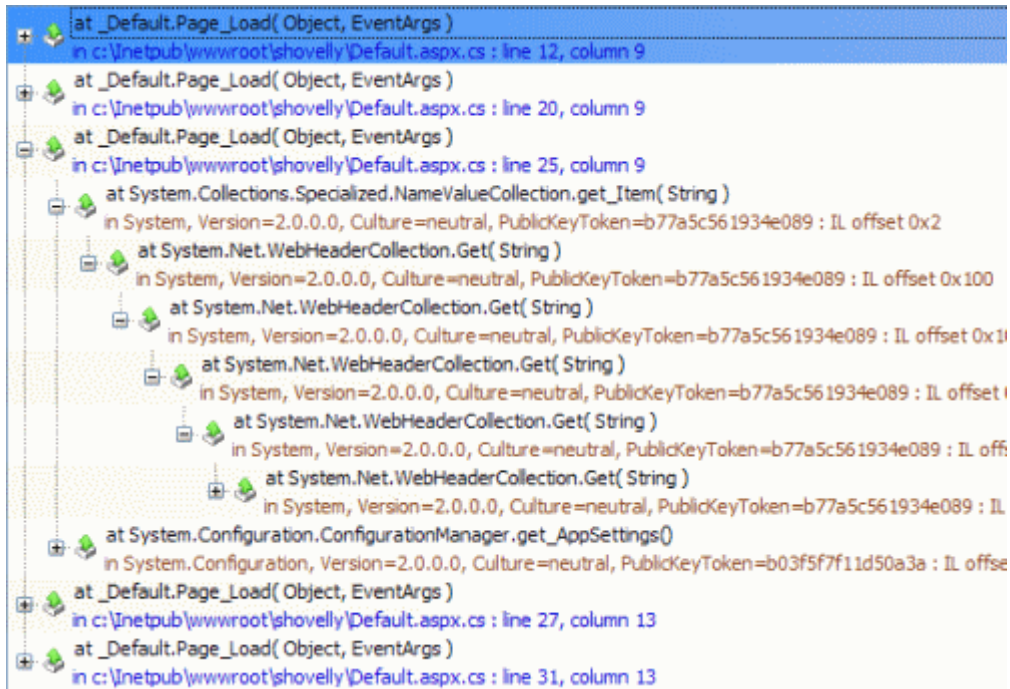
To view the stack traces for an exception, select the exception in the **Unhandled Exceptions** pane.





Move the mouse pointer over a line in the stack trace to view tooltip information for that line:



You can drill down through a stack trace using the plus and minus buttons.




The icons in the **Stack Traces** pane indicate the method that throws the exception:

-  the line of code that first throws the exception
-  a method that throws the exception further up the stack


The color of the line below the method name indicates whether Exception Hunter can find the PDB file associated with the assembly, and locate the source file referenced in the PDB.

 at Mandelbrot.Image.Clear()
in C:\Mandelbrot\Image.cs : line 51, column 4

Blue text means that Exception Hunter can locate the source file containing the line of code that throws the exception. Click to view the file.

 at Mandelbrot.Complex.ToString()
in C:\Mandelbrot\Complex.cs : line 32, column 4

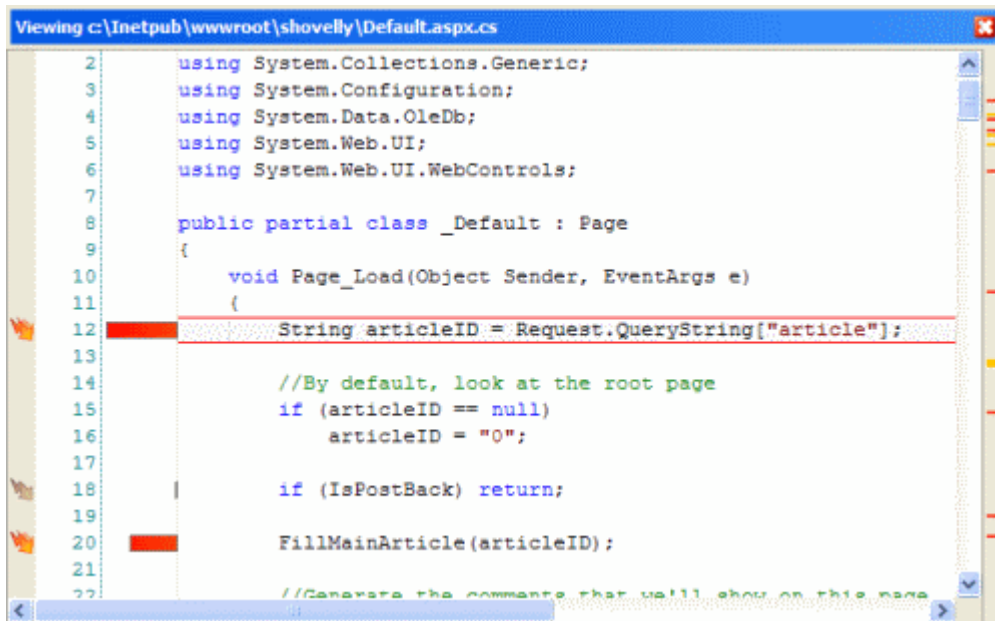
Red text means that Exception Hunter cannot locate the source file referenced in the PDB file.

 at System.Drawing.Bitmap..ctor(Int32, Int32)
in System.Drawing, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a


Orange text means that Exception Hunter cannot locate a PDB file associated with the assembly, and shows instead the assembly name.

Viewing the source code

Exception Hunter shows the source code of the selected method, as you drill down through a stack trace.



```
Viewing c:\inetpub\wwwroot\shovelly\Default.aspx.cs
2      using System.Collections.Generic;
3      using System.Configuration;
4      using System.Data.OleDb;
5      using System.Web.UI;
6      using System.Web.UI.WebControls;
7
8      public partial class _Default : Page
9      {
10         void Page_Load(Object Sender, EventArgs e)
11         {
12             String articleID = Request.QueryString["article"];
13
14             //By default, look at the root page
15             if (articleID == null)
16                 articleID = "0";
17
18             if (IsPostBack) return;
19
20             FillMainArticle(articleID);
21
22             //Generate the comments that we'll show on this page
23         }
```

The source code view displays which exceptions each line throws. An orange  icon indicates that the currently selected exception is thrown by the line.

If no source code is available for the selected method, Exception Hunter uses technology from Red Gate's .NET Reflector to display decompiled code for the method instead.

Notes

- Exception Hunter cannot detect exceptions that may be thrown when following delegate calls, for example Event Handler calls.
- Static classes appear in the list as abstract sealed classes, as there is no concept of a 'static class' in the .NET CLR.
- Runtime Exceptions (other than some NullReferenceExceptions and InvalidCastExceptions) generated by the .NET CLR are not detected by Exception Hunter.

Generating reports

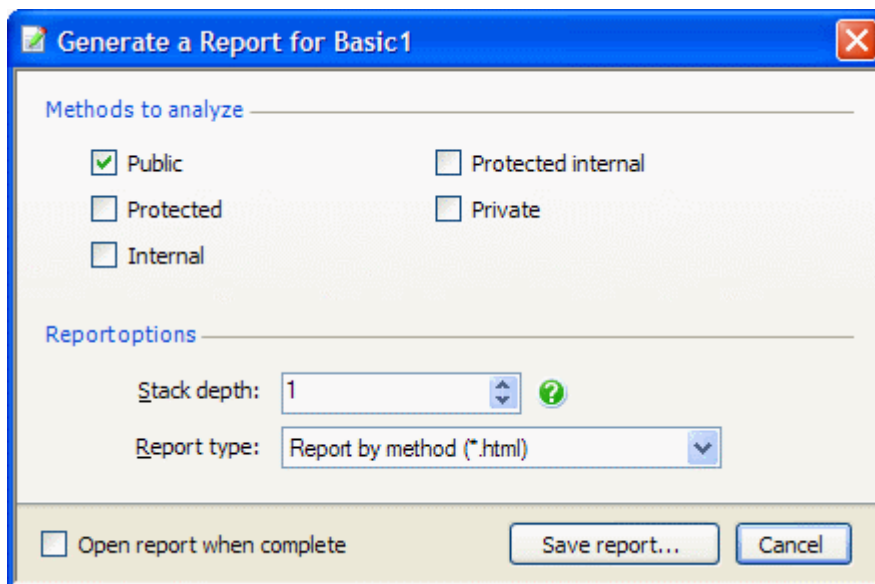
Exception Hunter can generate a report of the exceptions thrown by each of the methods in an assembly.

Three types of report are available:

- *Report by method (*.html)* shows which exceptions are thrown by each method
- *Report by exception (*.html)* shows which methods throw each exception
- *XML report (*.xml)* gives full details of exceptions in a format that can be read programmatically

To generate a report:

- navigate to the assembly (page 5), then click **Generate Report**



Alternatively, you can generate a report from the command line (page 15).

Using the command line

This topic describes how to use the basic features of the command line.

Getting help from the command line

To display help on any of the tools from the command line, enter:

```
Hunt /help
```

This displays a brief description of the tool, and basic help on all the command line switches.

For more detailed help enter:

```
Hunt /help /verbose
```

This displays a detailed description of each switch and the values it can accept (where applicable), and all exit codes. To output the help in HTML format, enter:

```
Hunt /help /verbose /html
```

Entering a command

When you enter a command line, the order of switches is unimportant. You are recommended to follow the Microsoft convention of separating a switch from its values using a colon as shown below.

```
/out:output.txt
```

(You can separate a switch that accepts a single value from its value using a space, but this is not recommended.) Values that include spaces must be delimited by double quotation marks ("). For example:

```
/out:"c:\output file.txt"
```

Note that if you delimit a path with double quotation marks, you must not terminate the path with the backslash character (\), because the backslash will be interpreted as an escape character. For example:

Incorrect: /location:"C:\Packages\"

Correct: /location:"C:\Packages"

For switches that accept multiple values, use commas to separate the values. For example:

```
/searchPaths:"C:\MyProjects", "D:\WorkFolder\Code"
```

For switches that accept a compound value, separate each part of the value using a colon.

Aliases

Many of the switches have an alias. The alias provides a convenient short-hand way to specify the switch. For example, */?* is the alias for the */help* switch, and */v* is the alias for the */verbose* switch. Note that switches and aliases are not case-sensitive.

/verbose and */quiet* switches

The standard output mode prints basic information about what the tool is doing while it is executing. You can specify verbose and quiet modes using the */verbose* and */quiet* switches, respectively: in verbose mode, detailed output is printed; in quiet mode, output is printed only if an error occurs.

Redirecting command output

Output from all commands can be redirected to a file by one of several methods:

- Use the */out* switch to specify the file to which you want output directed:

```
Hunt ... /out:outputlog.txt
```

where *outputlog.txt* is the name of the file. If the file exists already, you must also use the */force* switch to force the tool to overwrite the file, otherwise an error will occur.

- Use the output redirection features that are provided by the shell in which you are executing the command.

From the standard command prompt provided by Windows, you can redirect output to a file as follows:

```
Hunt ... > outputlog.txt
```

Note that the redirection operator (*>*) and file name must be the last items on the command line. If the specified file exists already, it will be overwritten. To append output from the tool to an existing file, for example to append to a log without losing the data already present in the log, enter the following:

```
Hunt ... >> existinglog.txt
```

If you are scripting using a language such as VBScript, JScript, PHP, Perl, or Python, or if you want to access the tool from Web pages using ASP.NET, refer to the documentation for the relevant language.

Examples using the command line

This topic provides some simple examples of how to use the command line interface.

Analyzing an assembly for exceptions

To detect the exceptions for all methods in the assembly *WidgetAssembly.dll* :

```
hunt /assembly:WidgetAssembly.dll  
  
    /all
```

If you do not specify the path to the assembly file, Exception Hunter assumes it is in the same folder as the *Hunt.exe* file. To specify an assembly in a different folder, enclose the full path in quotes:

```
hunt /assembly:"C:\MyProjects\WidgetProject\WidgetAssembly.dll"  
  
    /all
```

Note: If you do not specify either `/all` or at least one method using the `/method` argument, an error is returned.

Analyzing more than one assembly

To analyze more than one assembly, use separate `/assembly` arguments to specify each assembly:

```
hunt /assembly:"C:\MyProjects\WidgetProject\WidgetAssembly1.dll"  
  
    /assembly:"C:\MyProjects\WidgetProject\WidgetAssembly2.dll"  
  
    /assembly:"C:\MyProjects\WidgetProject\WidgetAssembly3.dll"  
  
    /all
```

Selecting specific methods to analyze

To specify particular methods to analyze, use the `/method` argument. Use a separate argument for each method:

```
hunt /assembly:"C:\MyProjects\WidgetProject\WidgetAssembly1.dll"  
  
    /method:System.Int32.ToString(String)  
  
    /method:System.Code.WidgetFunction()
```

You need to specify only enough of the name to make it unique.

Producing a report

By default the results of the analysis are output to the console.

Use the `/xml` switch to produce a results file containing the results as a set of xml data that you can translate into whatever format you require. You can also choose one of two HTML format reports:

- `/methodReport`
The results are organized by method.
- `/exceptionReport`
The results are organized by exception type.

Setting analysis options

Use the following switches to control how Exception Hunter analyzes your methods:

<code>/frameworkVersion x.x</code>	Analyzes assemblies against a specified version of the .NET Framework. For example, <code>/frameworkVersion1.1, frameworkVersion3.0.</code> If a framework switch is not specified, .NET framework 2.0 is assumed.
<code>/noGAC</code>	Does not search the GAC when analyzing assemblies.
<code>/stackDepth: n</code>	Sets the maximum depth of the stack that is used when displaying exceptions. Note that a value of 5 or greater may result in Exception Hunter taking a long time to complete its operation.
<code>/maxOverrides: n</code>	Specifies the maximum number of virtual or interface overrides to follow. If the number of possible implementations of an object is greater than the number specified in this argument, Exception Hunter will ignore all exceptions thrown by this method. Some methods like <code>GetHashCode()</code> and interfaces like <code>IComparer</code> can be implemented by very many objects, so increasing the value of this option can result in much longer processing times.
<code>/ignoreCodeBranches</code>	When this argument is specified, Exception Hunter does not attempt to eliminate code branches based on the possible values that have been evaluated for a variable.

<code>/fastMode</code>	Uses a faster algorithm to analyze the code. Produces results more quickly but detects fewer exceptions. Typically, Exception Hunter detects fewer potential <code>NullReferenceException</code> and <code>InvalidCastException</code> exceptions but more exceptions that may never be thrown due to code logic, for example <code>ArgumentNullException</code> .
<code>/ignoreChattyClasses</code>	Ignores calls to the <code>ResourceManager</code> and other apparently simple .NET library methods that call into many other parts of the library; using this switch can reduce the 'noise' of unexpected results and reduces the time required to complete the analysis.

For a complete list of arguments and syntax for the command line, type:

```
hunt /? /v
```

Acknowledgements

Trademarks and registered trademarks

Red Gate is a registered trademark of Red Gate Software Ltd registered in the U.S. Patent and Trademark Office.

.NET Reflector and SQL Compare are registered trademarks of Red Gate Software Ltd registered in the U.S. Patent and Trademark Office.

ANTS Performance Profiler, ANTS Memory Profiler, .NET Reflector Pro, Exception Hunter, Schema Compare for Oracle, SQL Backup, SQL Data Compare, SQL Comparison SDK, SQL Dependency Tracker, SQL Doc, SQL HyperBac, SQL Log Rescue, SQL Monitor, SQL Multi Script, SQL Packager, SQL Prompt, SQL Refactor, SQL Scripts Manager, SQL Storage Compress, SQL Toolbelt, SQL Virtual Restore, and Exchange Server Archiver are trademarks of Red Gate Software Ltd.

Microsoft, Windows, Windows 98, Windows NT, Windows 2000, Windows 2003, Windows XP, Windows Vista, Windows 7, Visual Studio, and other Microsoft products referenced herein are either registered trademarks or trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

InstallShield is a registered trademark and service mark of InstallShield Software Corporation.

PagerDuty is a registered trademark of PagerDuty, Inc.

Copyright information

All Red Gate applications are © Red Gate Software Ltd 1999 - 2011

SQL Backup, SQL Compare, SQL Data Compare, SQL Packager, and SQL Prompt contain software that is Copyright © 1995 - 2005 Jean-loup Gailly and Mark Adler.

SQL Doc includes software developed by Aspose (<http://www.Aspose.com>).

SQL Backup contains software that is Copyright © 2003 - 2008 Terence Parr. Refer to the ACKNOWLEDGEMENTS.txt file in your SQL Backup installation directory for the full license text.