

# **ANTS Performance Profiler 6.3**

July 2011

Note: these pages apply to a version of this product that is not the current released version.

For the latest support documentation, please see <http://documentation.red-gate.com>

# Contents

---

Getting started .....	5
Worked example: Profiling performance of an algorithm .....	6
Worked example: Profiling network overheads.....	10
Worked example: Profiling an ASP.NET application .....	12
Worked example: Profiling from the command line.....	17
Worked example: Profiling SQL queries.....	20
Setting up and running a profiling session .....	25
Working with application settings .....	27
Setting up Charting Options .....	30
Profiling .NET executables .....	32
Profiling managed code add-ins .....	33
Profiling ASP.NET applications running on IIS .....	35
Profiling SharePoint.....	38
Profiling ASP.NET applications running on the web development server .....	41
Profiling Silverlight 4 browser applications.....	43
Profiling Windows services .....	47
Profiling COM+ server applications .....	50
Profiling XBAP applications .....	53
Attaching to a running .NET 4 process .....	55
Profiling SQL queries .....	58
Profiling File I/O.....	65
Setting up MSTest.....	70
Profiling from the command line .....	73
Integrating ANTS Performance Profiler in a test procedure .....	80
Using the Visual Studio add-in .....	82
The ANTS Performance Profiler user interface .....	84
Working with profiling results .....	85
Working with the timeline .....	86
Working with the call tree .....	90
Working with the methods grid .....	94
Working with the call graph.....	96
Working with source code .....	100
ANTS Performance Profiler options .....	102
Troubleshooting application crashes .....	104
Troubleshooting missing results .....	107

Troubleshooting PDB problems .....	109
Troubleshooting SharePoint Profiling.....	112
Troubleshooting IIS profiling .....	120
Acknowledgements .....	124

## Getting started

---

ANTS Performance Profiler enables you to profile the code of applications written in any of the languages available for the .NET Framework, including Visual Basic .NET, C#, and Managed C++. This is useful, for example, to identify inefficient areas of your application by recording the time spent in each line of your code or method as you run your application.

You can use ANTS Performance Profiler to profile .NET desktop applications, ASP.NET web applications hosted in Internet Information Services (IIS) or the ASP.NET Development Server, .NET Windows services, COM+ server applications, Silverlight 4 or later applications, and XBAPs. In addition, you can profile applications that host the .NET Runtime, for example Visual Studio .NET plug-ins.

You can use ANTS Performance Profiler with the following versions of the .NET Framework:

- 1.1 (32-bit applications only)
- 2.0 (32-bit or 64-bit applications)
- 3.0 (32-bit or 64-bit applications)
- 3.5 (32-bit or 64-bit applications)
- 4.0 (32-bit or 64-bit applications)

ANTS Performance Profiler: step-by-step

1. Set up a new profiling session, and start profiling.
2. Optionally, select a region on the timeline to restrict the profiling results to a specific period.
3. Review the profiling results.

### Worked example

Learn more about performance and memory profiling with ANTS Performance Profiler by following this detailed example:

- Worked example: profiling the performance of an application.

## Worked example: Profiling performance of an algorithm

---

This worked example shows how you can use ANTS Performance Profiler to identify the most time-critical parts of a demonstration application, particularly where two different approaches to a problem are optimal in different circumstances.

Note that the demonstration application used in this worked example is supplied with ANTS Performance Profiler 6.3 and later. If you have an earlier version of ANTS Performance Profiler, either upgrade to version 6.3, or see the Mandelbrot worked example ([http://www.red-gate.com/supportcenter/Content?p=ANTS%20Performance%20Profiler&c=ANTS\\_Performance\\_Profiler/help/6.2/app\\_Worked\\_example\\_performance.htm&toc=ANTS\\_Performance\\_Profiler/help/6.2/toc140804.htm](http://www.red-gate.com/supportcenter/Content?p=ANTS%20Performance%20Profiler&c=ANTS_Performance_Profiler/help/6.2/app_Worked_example_performance.htm&toc=ANTS_Performance_Profiler/help/6.2/toc140804.htm)) for version 6.2.

### Introducing TimeLineDemo

This worked example uses *TimeLineDemo*.

*TimeLineDemo* is a simple Windows application that checks which of a set of numbers are prime.

There are two different versions of *TimeLineDemo*:

- In *Brute Force* configuration, a brute-force method is used to check whether the number is prime.

The *Brute Force* build of *TimeLineDemo* is found in `%Program Files%\Red Gate\ANTS Performance Profiler 6\Tutorials\CS\Precompiled\TimeLine\BruteForce\`

- In *Miller-Rabin* configuration, the Miller-Rabin algorithm is used to check whether the number is prime.

The *Miller-Rabin* build of *TimeLineDemo* is found in `%Program Files%\Red Gate\ANTS Performance Profiler 6\Tutorials\CS\Precompiled\TimeLine\MillerRabin\`

The application has two options:

- Max Random
- Sample Size

When *TimeLineDemo* first starts, it creates a list of as many prime numbers as possible in 15 seconds.

When you click **Go**, *TimeLineDemo* creates a list of positive integers, which is *Sample Size* items long. All of the integers are random numbers between 1 and *Max Random*. For each integer in the list, *TimeLineDemo* checks whether the number is prime, using the following algorithm:

1. If the number is in the list created in the first 15 seconds, it is prime.
2. If a number in the list created in the first 15 seconds is a factor of the current number, the current number is not prime.

3. In all other cases, use either a brute-force mechanism or the Miller-Rabin algorithm to check whether the number is prime (depending on the build in use).

Note: Strictly speaking, the Miller-Rabin algorithm is probabilistic. For the purposes of this demonstration, however, the algorithm can be assumed to be deterministic. This is because the algorithm's results are exact for integers which are smaller than the largest integer that the field accepts.

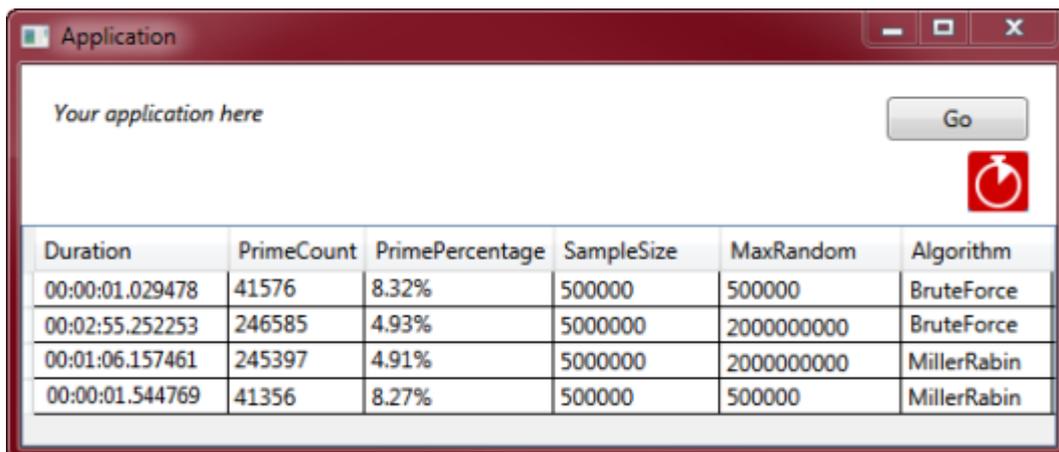
## Walkthrough

1. Start **ANTS Performance Profiler**
2. Select **.NET executable**
3. Enter the path to the brute-force build of *TimeLineDemo.exe*
4. Click **Start Profiling**
5. If required by your configuration, grant elevation permission
6. The demo application starts and goes through initial (start-up) phase
7. The application is shown and processor use decreases
8. Click the ANTS Performance Profiler logo
9. Set **MaxRandom** to *500,000* and **SampleSize** to *500,000*
10. Click **OK**
11. Click **Go**
12. The application performs the calculations
13. To see the expensive calculation methods, drag over that phase in the timeline
14. In the Call tree, select **RedGate.Demo.MainWindow.<GoButtonClick>**
15. The calculations took 1029 milliseconds; that's quick enough
16. Open the **File** menu and click **New Profiling Session...**
17. Click **Start Profiling**
18. When *TimeLineDemo* has started, click the ANTS Performance Profiler logo
19. Set MaxRandom to *2000000000* and SampleSize to *5000000*
20. Click **OK**
21. Click **Go** and wait for the calculations to finish
22. Select the calculation phase in the timeline, then **RedGate.Demo.MainWindow.<GoButtonClick>**
23. The calculations took 175,252 milliseconds or nearly 3 minutes. That's too slow.
24. To see the brute-force method, open *TimeLineDemo.csproj* in Visual Studio
25. In APP, right-click `ResultSet.Generate()` and select **Open with (Solution)**  
If prompted to elevate Visual Studio, click **Ignore**.
26. VS shows the expensive line. Right-click the method. Select **Go to Declaration**.
27. The Generate() method is displayed

28. The brute-force method is in use. Try the Miller-Rabin algorithm instead.
29. To switch to Miller-Rabin, click **Debug** and select **Configuration Manager...**
30. Next to **TimeLineDemo**, click **Debug** and select **MillerRabin**. Rebuild the demo.
31. In APP, open the **File** menu and click **New Profiling session...**
32. Change the **.NET executable** to the **Miller-Rabin build**
33. Click **Start Profiling** and wait for the demo to initialize
34. Click the ANTS Performance Profiler logo
35. Set **MaxRandom** to *2000000000* and **SampleSize** to *5000000*
36. Click **Go**
37. Select the calculation phase on the timeline as before
38. The calculations took 66,149 milliseconds, about 66% faster than brute-force
39. Is Miller-Rabin always faster than brute-force? Profile *TimeLineDemo* again.
40. Set **MaxRandom** to *500000* and **SampleSize** to *500000* again
41. The calculations took 1,545 milliseconds, which is slower than brute-force

## Conclusion

This walkthrough has demonstrated a realistic programming scenario. It has compared two methods for testing if a number is prime and shown that Miller-Rabin is more efficient than the brute force approach when the number being tested is large.



Duration	PrimeCount	PrimePercentage	SampleSize	MaxRandom	Algorithm
00:00:01.029478	41576	8.32%	500000	500000	BruteForce
00:02:55.252253	246585	4.93%	5000000	2000000000	BruteForce
00:01:06.157461	245397	4.91%	5000000	2000000000	MillerRabin
00:00:01.544769	41356	8.27%	500000	500000	MillerRabin

This walkthrough has also demonstrated that you can select just a portion of the timeline when analyzing results, allowing you to ignore a slow initialization phase, for example.

## Learning more

To perform the task demonstrated in this walkthrough more efficiently, you can use the following additional functionality:

- To help you identify the section of the timeline to select, you can refer to the Event markers in the Events bar.
- You can name and bookmark sections of the timeline.

For more information, see [Working with the timeline](#).

## Worked example: Profiling network overheads

---

This worked example demonstrates how to use ANTS Performance Profiler to profile a .NET executable that exhibits latency problems caused by fetching HTTP data from a network.

Note that the demonstration application used in this worked example is supplied with ANTS Performance Profiler 6.3 and later.

### Introducing LatencyDemo

This worked example uses *LatencyDemo*.

*LatencyDemo* is a simple Windows application that fetches the RSS feeds from the ANTS Memory Profiler and ANTS Performance Profiler forums on the Red Gate website.

A copy of *LatencyDemo* is supplied with ANTS Performance Profiler in `%Program Files%\Red Gate\ANTS Performance Profiler 6\Tutorials\CS\Precompiled\LatencyDemo\`

### Walkthrough

1. Start **ANTS Performance Profiler**
2. Select **.NET executable**
3. Enter the path to the demo application *LatencyDemo.exe*
4. Click **Start Profiling**
5. If required by your configuration, grant elevation permission
6. The demo application starts
7. Load the ANTS Memory Profiler RSS feed by clicking **ANTS Memory Profiler**
8. Load the ANTS Performance Profiler RSS feed by clicking **ANTS Performance Profiler**
9. Click **Intro**
10. Click **Auto test**. Focus switches between the tabs 20 times.
11. In ANTS Performance Profiler, click **Stop Profiling**. *LatencyDemo* closes.
12. On the **Events bar**, find the Method event for *Click on Auto test*
13. On the **timeline**, drag over the time when the tabs were switching, starting from the Method event
14. Look at the hot stack trace. Time is lost by the tab control, but you can't see why.
15. Switch to **wall clock time**.
16. You should only look at one thread at a time in wall clock time, so find the UI thread.
17. The time is being lost in *System.Net.HttpWebRequest.GetResponse()*

18. Look at the hit count. 40 hits. We're not cacheing the result.

## Worked example: Profiling an ASP.NET application

---

This worked example describes how to profile a sample ASP.NET website called TheBeerHouse. The original ASP.NET MVC source code for TheBeerHouse can be obtained from CodePlex (<http://thebeerhouse.codeplex.com/releases/view/27519>). To create this worked example, TheBeerHouse has been recompiled for .NET 4.

TheBeerHouse has been installed on the same computer as the one being used to profile it, and it can be accessed in a web-browser from the address *[http://localhost/TBH\\_Web/](http://localhost/TBH_Web/)*

Imagine that the problem with TheBeerHouse is just that it is slow when loading pages. I want to know whether I can do anything to improve the site's performance, before I spend lots of money on improving the hardware it runs on.

There are three main steps:

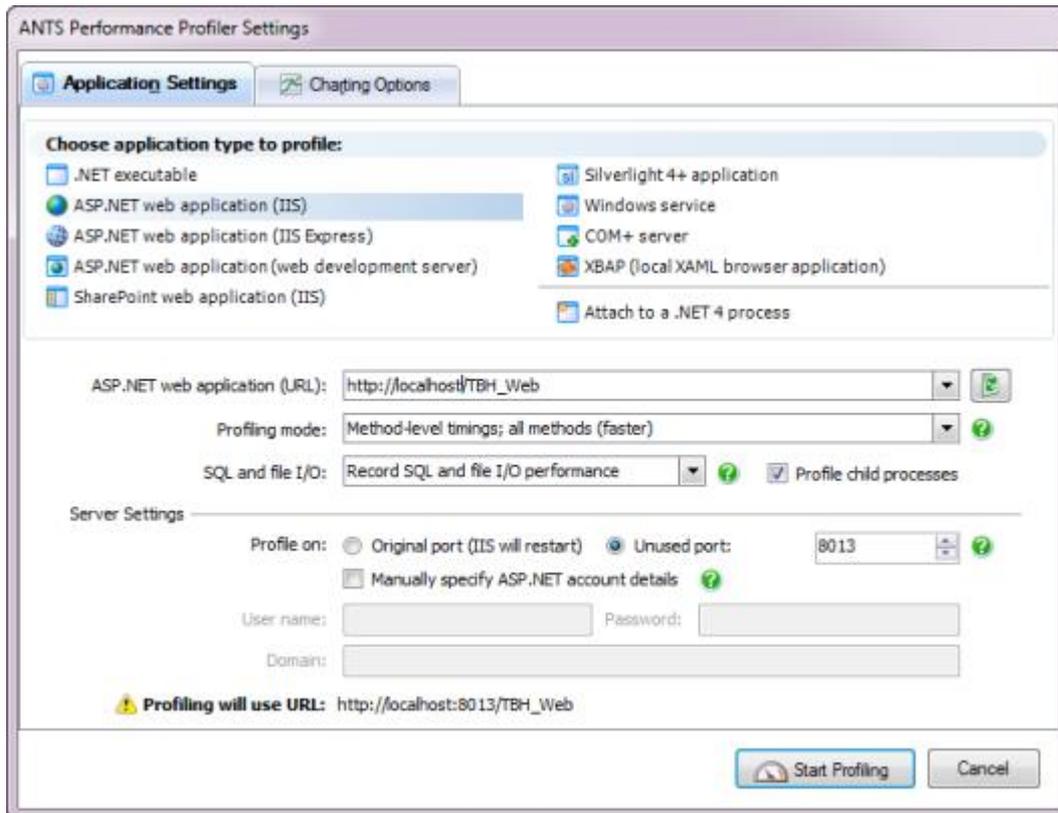
1. Set up ANTS Performance Profiler
2. Use TheBeerHouse
3. Analyze the profiler's results

### Setting up ANTS Performance Profiler

To set up ANTS Performance Profiler:

1. In the ANTS Performance Profiler settings, on the Application Settings tab, select **ASP.NET web application (IIS)**.
2. In the **ASP.NET web application (URL)** dropdown menu, select the site's URL: *[http://localhost/TBH\\_Web/](http://localhost/TBH_Web/)*.
3. Choose the required profiling mode.

4. Choose **Record SQL and file I/O performance** if you are interested in seeing these values. (Not available on Windows XP or Windows Server 2003.)



5. Select the port on which to profile your application:
  - If you are using IIS 6 or IIS 7, select **Unused port** and choose a port that is not used by IIS.

IIS will start a new instance of your application on the specified port.

Note that this will not work if your application's code binds to a specific port.

- If you are using IIS 5, or if you are using IIS 6 or 7 and your application binds to a specific port, select **Original port**.

IIS will restart so that the profiler can attach to the port.

Note that restarting IIS stops IIS and only restarts the application that you are profiling. If your website depends on another site running on the same IIS instance, that other site will not be present when IIS restarts.

If your application takes too long to start, IIS might not restart correctly. Use IIS Manager to stop the website manually until you have finished profiling.

The port where the application will be profiled is displayed at the bottom of the ANTS Performance Profiler Settings dialog box.

1. Click  .

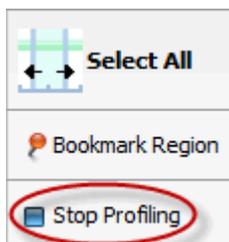
- Internet Explorer launches and shows TheBeerHouse. If you prefer not to use Internet Explorer, you can open a different browser at the same address. You must leave the instance of Internet Explorer created by the profiler open, however.



## Using TheBeerHouse

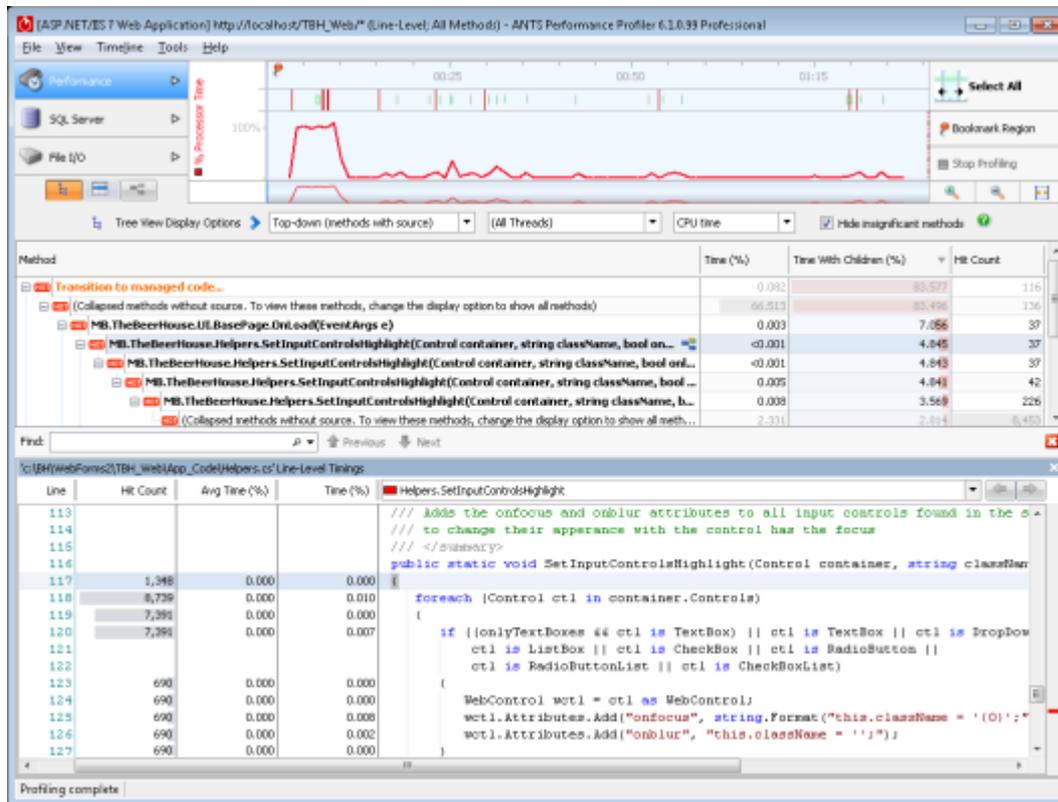
In this scenario, it is not known exactly where the performance problem is, and so initially a number of different pages are accessed, including some which are known to rely heavily on database queries, and some which mainly contain static HTML. After any particular performance problems have been identified, those pages can be profiled again in a more systematic manner.

After a number of different pages have been opened, in the ANTS Performance Profiler window, click **Stop Profiling**.



## Analyzing the profiler's results

After a few moments, the results are shown. ANTS Performance Profiler shows the 'hottest' stack trace; that is, the code which is using the greatest amount of CPU time. This is usually a good place to start looking for opportunities to optimize the code.



TheBeerHouse is already quite well optimized, with 66% of processor time being spent on very inexpensive methods. The site could be improved, however, because nearly 5% of the processor's time is spent on a method called `SetInputControlsHighlight()`, which runs when each page loads.

Select that row.

Because the source code for TheBeerHouse is available, ANTS Performance Profiler shows the source code for this method in the lower pane. Every time a page loads, `SetInputControlsHighlight()` iterates over the input fields it contains, and adds `onfocus` and `onblur` attributes to the HTML output, in turn causing the DOM to change their class when the input has focus. This is clearly a good candidate for optimization, because the same result can be achieved by just changing the CSS file to add the `:focus` pseudo-class.

In this instance, the File I/O and SQL results do not show anything abnormal.



## Worked example: Profiling from the command line

---

This worked example demonstrates how you can use ANTS Performance Profiler from the command line.

Profiling from the command line is useful if you want to integrate performance profiling into your usual testing or build processes. The profiler results can be output to CSV, XML or HTML, which means that you can easily check the results for abnormal values as part of your automated routines.

This example uses the following simple C# Console Application (called *SimpleApp.exe*) which prints '.' to the console 100 times:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SimpleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("The application has started");
            // Count from 0-99
            int i = 0;
            while (i < 100)
            {
                Console.Write('.');
                i++;
            }
            Console.WriteLine("The application is exiting");
        }
    }
}
```

Another simple C# Console Application can read the results CSV file created by ANTS Performance Profiler, to check that Write() is called exactly 100 times, thereby verifying that *SimpleApp* is performing correctly:

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ReadOutput
{
    class TextFileReader
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader(new
run FileStream("C:\testing\results.csv", FileMode.Open, FileAccess.Read));
            string line;
            int ok = 1;
            // Read file line-by-line
            while ((line = sr.ReadLine()) != null)
            {
                char separator = ',';
                string[] linedata = new string[10];
                linedata = line.Split(separator);
                // Try to cast linedata[3] as int, not string
                int i;
                try
                {
                    i = (int)int.Parse(linedata[3]);
                    // Check the number of times that Write(char value) is
                    if ((linedata[2] == "Write(char value)") && (i != 100))
                    {
                        Console.WriteLine("Test failed");
                        ok = 0;
                    }
                }
                catch
                {
                    // Do nothing
                }
            }
        }
    }
}

```



## Worked example: Profiling SQL queries

---

This walkthrough demonstrates how you can profile the SQL queries generated by your application.

Note: SQL Profiling is only available with Windows Vista or later. You must profile a locally-hosted SQL server. It is not possible to use SQL profiling with Microsoft SQL Server Express editions.

### Before you start

To follow this walkthrough, you will need to:

- install the *Customers* database table on your SQL server

A copy of the *Customers* database table is provided in  
*%ProgramFiles%\Red Gate\ANTS Performance Profiler  
6\Tutorials\CS\QueryBee\Customers.zip*

Ensure that no indexes or primary keys are set on the table.

- start *QueryBee.exe*

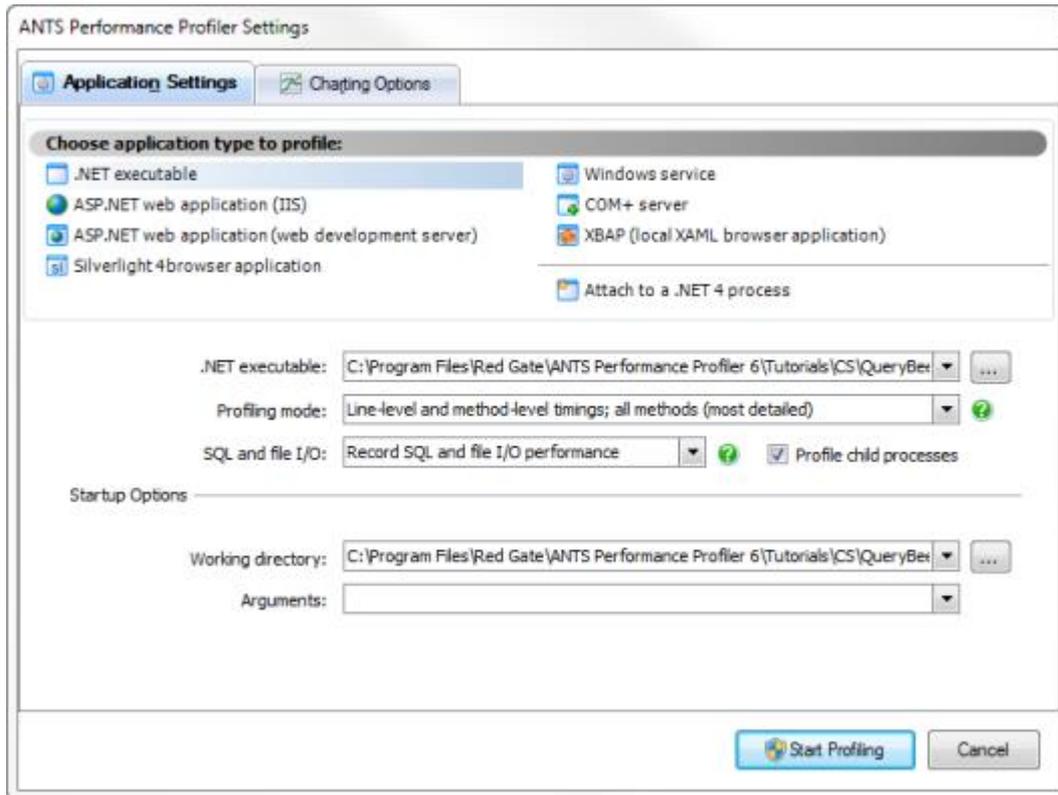
A copy of *QueryBee.exe* is provided in *%ProgramFiles%\Red Gate\ANTS Performance Profiler 6\Tutorials\CS\QueryBee\QueryBee.exe*

### Procedure

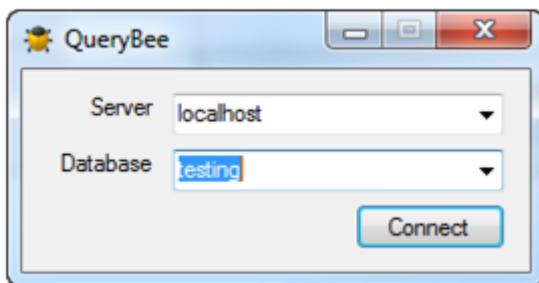
To profile *QueryBee* while it sends SQL queries to the server:

1. Start ANTS Performance Profiler.
2. Select **.NET executable**
3. Enter the path to *QueryBee.exe*

4. Ensure that **Record SQL and file I/O performance** is enabled.

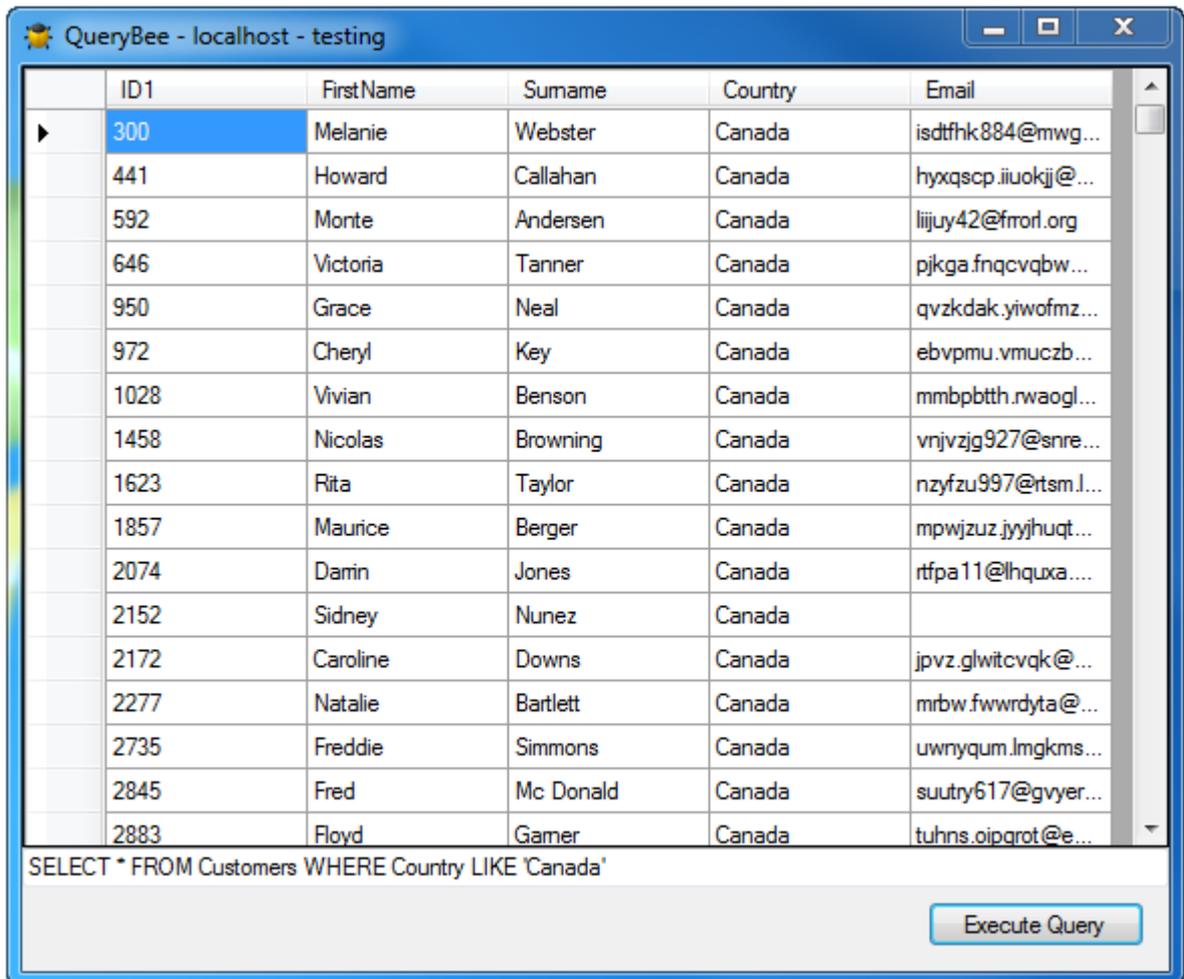


5. Click **Start Profiling** to start QueryBee. Select the test database and click **Connect**.



6. Run the query `SELECT * FROM Customers` to ensure that all records are read into memory, and that further queries will not include file I/O overheads.

7. Run the query `SELECT * FROM Customers WHERE Country LIKE 'Canada'`



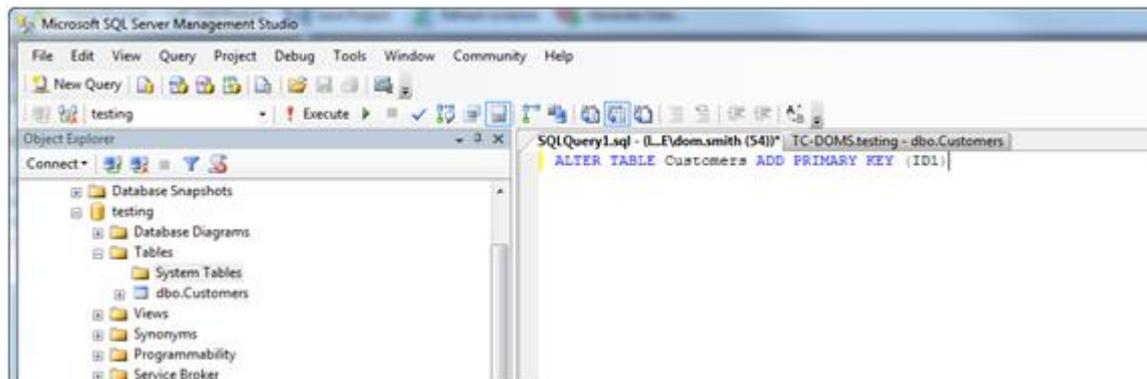
The screenshot shows a window titled "QueryBee - localhost - testing". It displays a table with the following columns: ID1, FirstName, Surname, Country, and Email. The table contains 20 rows of data, all with "Canada" in the Country column. The first row is highlighted in blue.

ID1	FirstName	Surname	Country	Email
300	Melanie	Webster	Canada	isdtfhk884@mwg...
441	Howard	Callahan	Canada	hyxqscp.iuokjj@...
592	Monte	Andersen	Canada	lijuy42@frrorl.org
646	Victoria	Tanner	Canada	pjkgafnqcvqbw...
950	Grace	Neal	Canada	qvzkdak.yiwofmz...
972	Cheryl	Key	Canada	ebvpmu.vmuczb...
1028	Vivian	Benson	Canada	mmbpbth.rwaogl...
1458	Nicolas	Browning	Canada	vnjvzjg927@snr...
1623	Rita	Taylor	Canada	nzyfzu997@rtsm.l...
1857	Maurice	Berger	Canada	mpwjzuz.jyyjuqt...
2074	Darin	Jones	Canada	rtpa11@lhquxa....
2152	Sidney	Nunez	Canada	
2172	Caroline	Downs	Canada	jpvoz.glwitcvqk@...
2277	Natalie	Bartlett	Canada	mrbw.fwwrdyta@...
2735	Freddie	Simmons	Canada	uwnyqum.lmgkms...
2845	Fred	Mc Donald	Canada	suutry617@gvyer...
2883	Floyd	Gamer	Canada	tuhns.oipqrot@e...

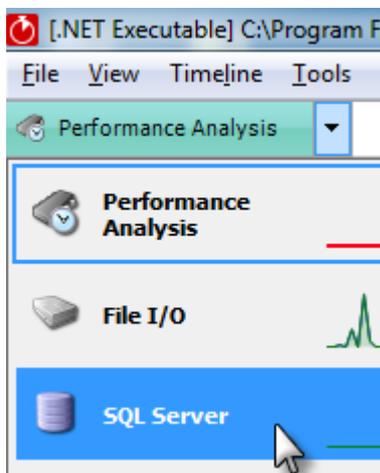
Below the table, the SQL query is displayed: `SELECT * FROM Customers WHERE Country LIKE 'Canada'`. An "Execute Query" button is located at the bottom right of the window.

8. The following query is executed, using the equals operator instead:  
`SELECT * FROM Customers WHERE Country='Canada'`
9. To test updated information, run the following query:  
`UPDATE Customers SET Email='a@example.com' WHERE ID1=100`

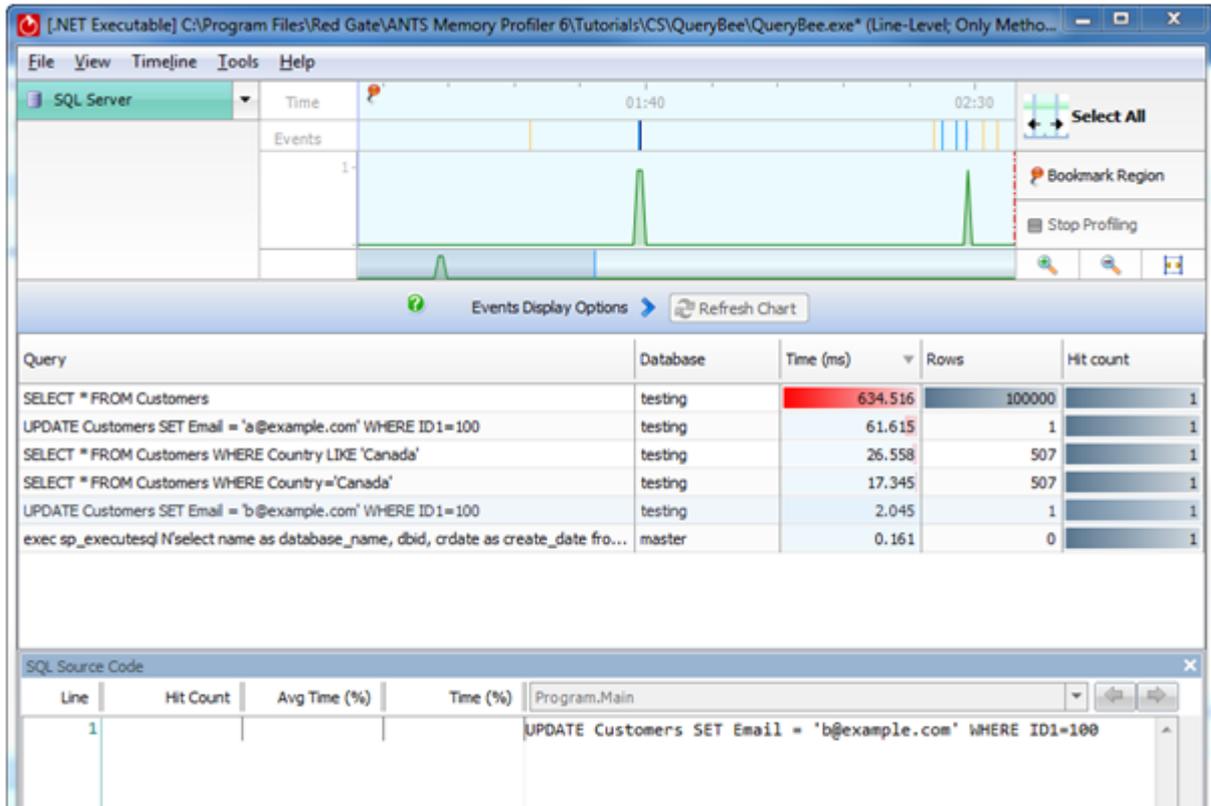
10. To show that a primary key will make this query quicker, a primary key is applied to the ID1 field in SQL Server Management Studio:



11. Run the following query in QueryBee:  
`UPDATE Customers SET Email='b@example.com' WHERE ID1=100`
12. In the ANTS Performance Profiler window, click **Stop Profiling**
13. On the **Performance Analysis** menu, set ANTS Performance Profiler to **SQL Server** profiling mode:



ANTS Performance Profiler displays the time taken for each query and the number of rows affected or returned. The LIKE operator is slower than the = operator for text comparison, and the update query was much quicker with a primary key than without a key.



The examples in this worked example have been kept deliberately simple.

A more complex use case for SQL profiling that you may consider is when complex code generates an SQL query at runtime. In such cases, it might be difficult to check exactly what query has been run, but profiling an application with ANTS Performance Profiler allows you to see immediately how your application has interacted with the SQL server.

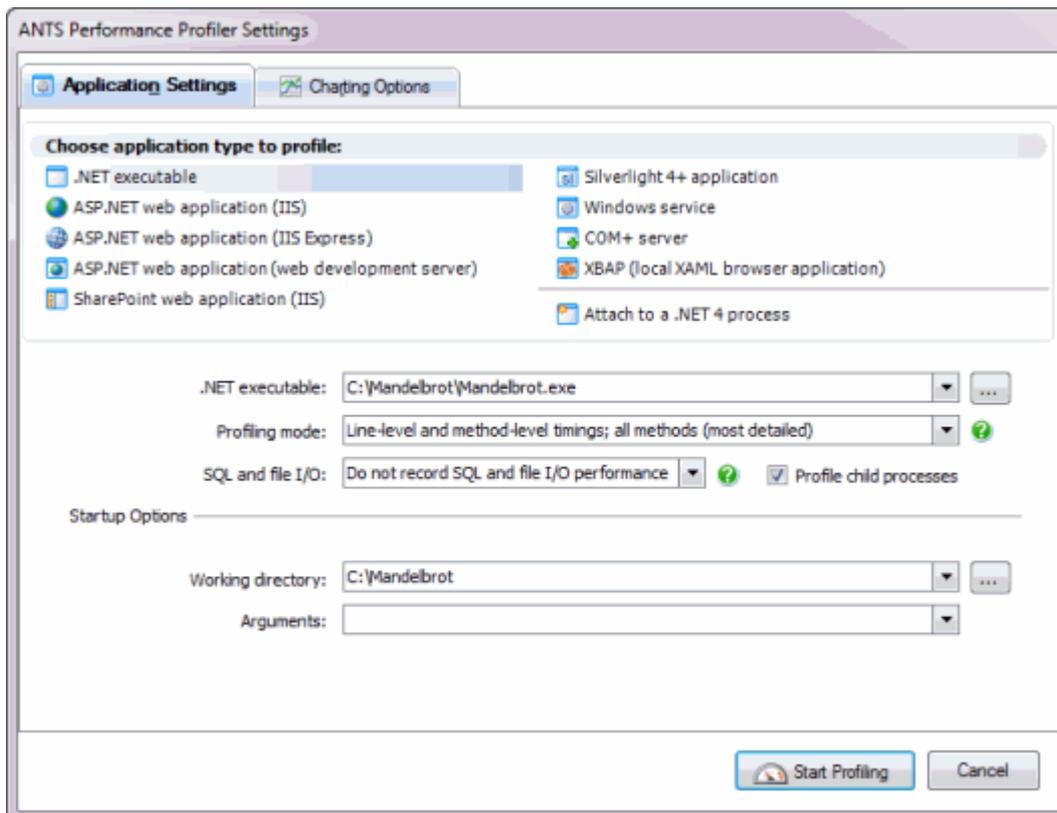
## Setting up and running a profiling session

---

To profile an application, you must first set up a profiling session. A session specifies:

- The application type, location, and options for the application you want to profile.
- The profiling mode, which determines the level of detail gathered by the profiler while your application is running.
- The method used to calculate timing values (CPU time or wall-clock time).
- The performance counters to display on the timeline.

When you start ANTS Performance Profiler, the **ANTS Performance Profiler Settings** dialog box is automatically displayed; if ANTS Performance Profiler is already running, click **New Profiling Session** on the **File** menu.

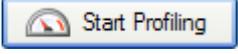


The **Application Settings** tab displays the settings for the last profiling session you ran. The settings available depend on the selected application type, and may differ from those illustrated above.

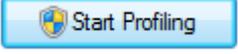
The **Charting Options** tab enables you to choose which performance counter values to display on the timeline for the new profiling session.

## To set up and run a profiling session:

1. On the **ANTS Performance Profiler Settings** dialog box, complete the details on the Application Settings tab.
2. Choose which performance counters to monitor during profiling using the Charting Options tab.

3. Click .

On Windows Vista, Windows Server 2008, and Windows 7, if you are not running ANTS Performance Profiler as an Elevated administrator, the **Start Profiling** button

has a User Account Control (UAC) shield: . The UAC shield indicates that ANTS Performance Profiler will request elevation when you start profiling.

The timeline is displayed at the top of the main ANTS Performance Profiler window, and the application you want to profile is automatically started. Status text at the bottom-left of the main window indicates what ANTS Performance Profiler is doing during the profiling session.

The timeline starts displaying performance-counter data and events in near-real time. There may be a slight delay between starting a profiling session and seeing the first performance-counter data appear on the timeline.

4. To display profiling results, do one of the following:

- ◆ Drag a region on the timeline.

Profiling data is summarized and displayed for the selected time period only. Your application will continue running and profiling will continue.

- ◆ Click  **Stop Profiling**.

Your application will be closed. Profiling data is summarized and displayed for the entire profiling period.

- ◆ Close your application.

Profiling data is summarized and displayed for the entire profiling period.

You can continue working with the timeline to locate periods of interest during the execution of your application, and to display the associated profiling results.

Once you have displayed some profiling data, you can view and analyze it. For more information about the different ways you can do this, see [Working with profiling results](#).

## Working with application settings

---

Each application type is associated with a number of settings. These include settings that are common to all application types (**Profiling mode** and **Default timing display**), and some settings that are specific to individual application types.

### Application types

Select the application type from the **Choose application type to profile** list. The settings available change depending on your selection:

- *.NET executable*  
**Startup Options.** You can specify the command line arguments that are to be used when running the application.
- *ASP.NET web application (hosted in IIS)*  
**Server Settings and ASP.NET Account Details.** If you are using IIS version 6 or IIS version 7 you can select a different port to profile on; this is not available if you are using IIS version 5. If you choose a port which is already in use, you must stop IIS to free the port before you start profiling. You can also manually specify ASP.NET account details, so that you can run a profile as a different user. This is useful if your web application needs access to a remote server.  
Web applications which implement the Windows Communication Foundation (WCF) can also be profiled.  
Web applications are profiled in Microsoft Internet Explorer, even if it is not your preferred browser. This is because ANTS Performance Profiler uses the low-level data exposed by Internet Explorer.
- *ASP.NET web application (hosted in web development server)*  
**Server Settings.** You can specify the URL that your web application starts on. For example, if you specify "staging" for the virtual directory and "8013" for the port number, your web application starts on URL *http://localhost:8013/staging/*.  
Web applications are profiled in Microsoft Internet Explorer, even if it is not your preferred browser. This is because ANTS Performance Profiler uses the low-level data exposed by Internet Explorer.
- *Silverlight 4 browser*  
**Silverlight application URL.** Enter the URL of the Silverlight 4 browser application you want to profile. This feature requires the Silverlight 4 plugin in Internet Explorer.
- *Windows service*  
**Startup Options.** You can specify command line arguments that are to be used when running the application.
- *COM+ server*  
**COM+ server Application.** You can specify the location of the COM+ server application.  
**Client Application.** You can specify command line arguments that are to be used when running the client application.  
See Profiling a COM+ server for more details.

- *XBAP (XAML Browser Application)*  
No additional setup options are provided. To profile a remotely-hosted XBAP application, select the .NET executable application type, and then profile Internet Explorer (iexplore.exe) and navigate to the XBAP application.
- *Attach to .NET 4 process*  
Choose the .NET process you want to attach to. This feature requires Windows Vista or later and .NET 4.

## Profiling mode

The **Profiling mode** determines the level of detail gathered by the profiler while your application is running. The level of detail that you choose may affect the profiling speed and the overall accuracy of the results.

Profiling Mode	Speed	Accuracy	Detail	Profiling Data
<i>Line-level and method-level timings; all methods*</i>	★	★★	★★★★	All methods. This includes methods without source code, such as those in the .NET Framework class libraries.
<i>Method-level timings; all methods</i>	★★★	★★★★	★★	Only methods for which source code is available, for example, timings will not be measured for .NET Framework methods.
<i>Line-level and method-level timings; only methods with source*</i>	★★	★	★★★	
<i>Method-level timings; only methods with source</i>	★★★★	★★★★★	★	
<i>Sample method-level timings</i>	★★★★★	★	★	

\*Profiling data is also collected for individual lines of code.

## SQL and file I/O

- If you have Windows Vista or later, you can choose to record file I/O operations.
- If you have Windows Vista or later and a SQL server (except Express editions) installed on the local machine, you can choose to record file I/O operations and SQL queries.

## Default timing display

The **Default timing display** determines the method used to calculate timing values:

- *CPU*  
Timings *exclude* any periods of time for which a process is blocked. This can include sleeping, waiting for I/O, or waiting for some other resource.
- *Wall clock*  
Timings *include* any periods of time for which a process is blocked (including sleeping, waiting for I/O, or waiting for some other resource).

You can also change the timing display *after* you have collected profiling data, by selecting *CPU time* or *Wall-clock time* from the **Timing** list on the display toolbar. See *Working with the call tree*, *Working with the methods grid*, or *Working with the call graph* for more information.

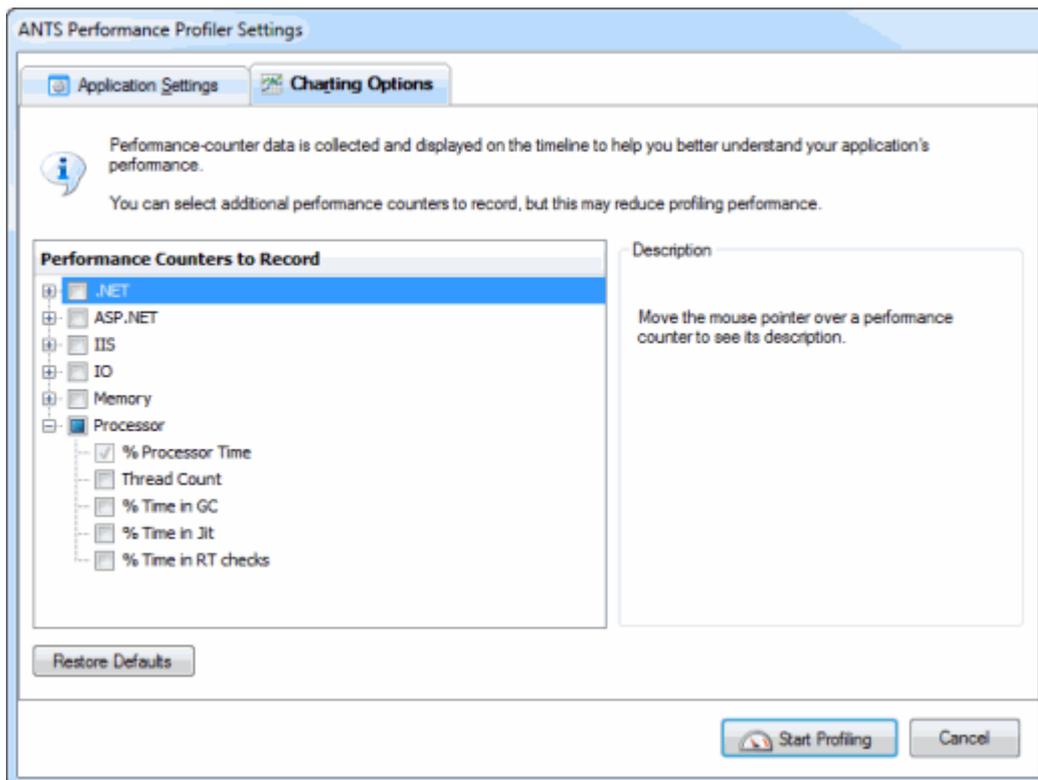
## Setting up Charting Options

---

ANTS Performance Profiler can monitor the values of a number of built-in Windows performance counters while the application you are profiling is executing. The values of these counters are constantly updated on the timeline as profiling proceeds.

You choose the performance counters you want to monitor using the **Charting Options** tab on the **ANTS Performance Profiler Settings** dialog box.

Note that this feature requires ANTS Performance Profiler Professional.



Not all performance counters are appropriate to all application types that you may be profiling. You can find more information about individual performance counters under the **Description** group box, including details about a counter's relevance to particular application types.

Your choice will depend on your own requirements but, as an example, you might choose the following:

From the **.NET** group:

- The *Gen 0 Promoted Bytes/sec* counter  
This will give the rate at which the garbage collector promotes objects from Generation 0 to Generation 1.

From the **Memory** group:

- The *Working Set* counter  
This shows the total amount of physical memory used by your service (including memory used by shared DLLs and the .NET runtime itself)

From the **Processor** group

- The *% Processor Time* counter (selected by default)  
This shows the percentage of time which all running threads use on the CPU.
- The *% Time in GC* counter  
This shows the percentage of time which the process was suspended to allow the last garbage collection to take place.

We recommend you avoid adding more performance counters than you need, as each additional counter that is recorded adds to the overhead introduced by the profiler. Adding too many counters may slow your application substantially.

### Adding custom performance counters

You can add custom performance counters to the list of available counters in the **Charting Options** tab. To do this:

1. Close ANTS Performance Profiler.
2. Expose your performance counter to the Windows Performance Counter API using the *PerformanceCounter* and *PerformanceCounterCategory* classes of the *System.Diagnostics* namespace. (An example describing how to do this is given at <http://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter.aspx> (<http://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter.aspx>))

3. Create a new XML file as follows:

```
<Counters>
  <Category Name="CategoryName">
    <Counter Category="CategoryName" Name="CounterName"
Units="Measurement Units">
      <Instanced />
    </Counter>
  </Category>
</Counters>
```

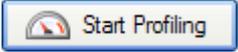
Ensure that *CategoryName* and *CounterName* are the same as the names used for the *PerformanceCounterCategory* and *PerformanceCounter*. Remove the `<Instanced />` node if your counter collects data about the computer, not only the individual process. You can add multiple categories and counters in the same XML file.

4. Save the XML file as *UserCounters.xml* in *%LOCALAPPDATA%\Red Gate\ANTS Performance Profiler 6\*.
5. Restart ANTS Performance Profiler.
6. The counters that you defined are shown in the list on the **Charting Options** tab.

## Profiling .NET executables

---

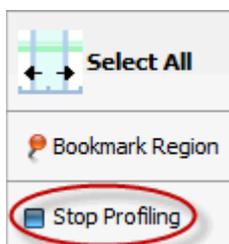
To profile .NET executables, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **.NET executable**.
2. Browse to the **.NET executable** that you want to profile.  
Use the dropdown list to select a recently-profiled application.
3. Select the required **Profiling mode**, **SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. If required, change the **Working directory**.  
The working directory is the path where the application will start. By default, this is the directory where the executable is located.  
Use the dropdown list to select a recently-used working directory.
5. If required, specify any command line **Arguments** that are to be used when running the application.
6. If required, change the performance counters to record; see Setting up Charting Options.
7. Click .

The .NET executable starts; interact with the application normally.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your application, click the  **Stop Profiling** button in ANTS Performance Profiler.



See also Worked example: Profiling the performance of an algorithm.

## Profiling managed code add-ins

---

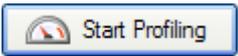
Microsoft Visual Studio, Microsoft Word, Microsoft Excel and Microsoft SQL Server Management Studio are all examples of desktop applications which host the .NET Common Language Runtime. You can create .NET add-ins for these programs.

The procedure for using ANTS Performance Profiler to profile a native desktop application that hosts the .NET runtime is similar to that for profiling other .NET applications.

A SQL Server Management Studio add-in is used as an example in this topic. SQL Server Management Studio is a native application which hosts the CLR which, in turn, allows it to execute managed code. If you are profiling an add-in for a different program (Word or Excel, for example), you must use that program instead of SQL Server Management Studio in all of the following instructions.

### Setting up the performance profiler

To profile managed code add-ins, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **.NET executable**.
2. Browse to the **.NET executable** that you want to profile.  
Note: this is the native application which contains your add-in; for example, SQL Server Management Studio.  
Use the dropdown list to select a recently-profiled application.
3. Select the required **Profiling mode**, **SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. If required, change the **Working directory**.  
The working directory is the path where the application will start. By default, this is the directory where the native application is located.  
Use the dropdown list to select a recently-used working directory.
5. If required, specify any command line **Arguments** that are to be used when running the application.
6. If required, change the performance counters to record; see Setting up Charting Options.
7. Click .

SQL Server Management Studio starts. Connect to a server and interact with your add-in.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.



### Other applications

The same approach will work with all other hosting applications. For example, to profile a Visual Studio .NET add-in, enter the full path to *devenv.exe* as the .NET application that you want to profile.

You can also profile a managed code Microsoft Management Console (MMC) snap-in in the same way; select *mmc.exe* as the .NET executable, add the snap-in, then interact with it.

## Profiling ASP.NET applications running on IIS

---

To profile ASP.NET applications running on IIS, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **ASP.NET web application (IIS)**.
2. Select the **ASP.NET web application (URL)** for the root directory of the web application that you want to profile.

To load a list of currently-running sites from IIS into the dropdown list, click .

Note that the port specified in this URL is the port where the application usually runs under IIS, which is not necessarily the same as the port where the application is to be profiled (see step 4).

3. Select the required **Profiling mode, SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. Select the port on which to profile your application:

- ◆ If you are using IIS 6 or IIS 7, select **Unused port** and choose a port that is not used by IIS.

IIS will start a new instance of your application on the specified port.

Note that this will not work if your application's code binds to a specific port.

- ◆ If you are using IIS 5, or if you are using IIS 6 or 7 and your application binds to a specific port, select **Original port**.

IIS will restart so that the profiler can attach to the port.

Note that restarting IIS stops IIS and only restarts the application that you are profiling. If your website depends on another site running on the same IIS instance, that other site will not be present when IIS restarts.

If your application takes too long to start, IIS might not restart correctly. Use IIS Manager to stop the website manually until you have finished profiling.

The port where the application will be profiled is displayed at the bottom of the ANTS Performance Profiler Settings dialog box.

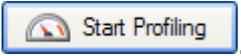
5. If required, select **Manually specify ASP.NET account details** and enter the **User name, Password** and **Domain**.

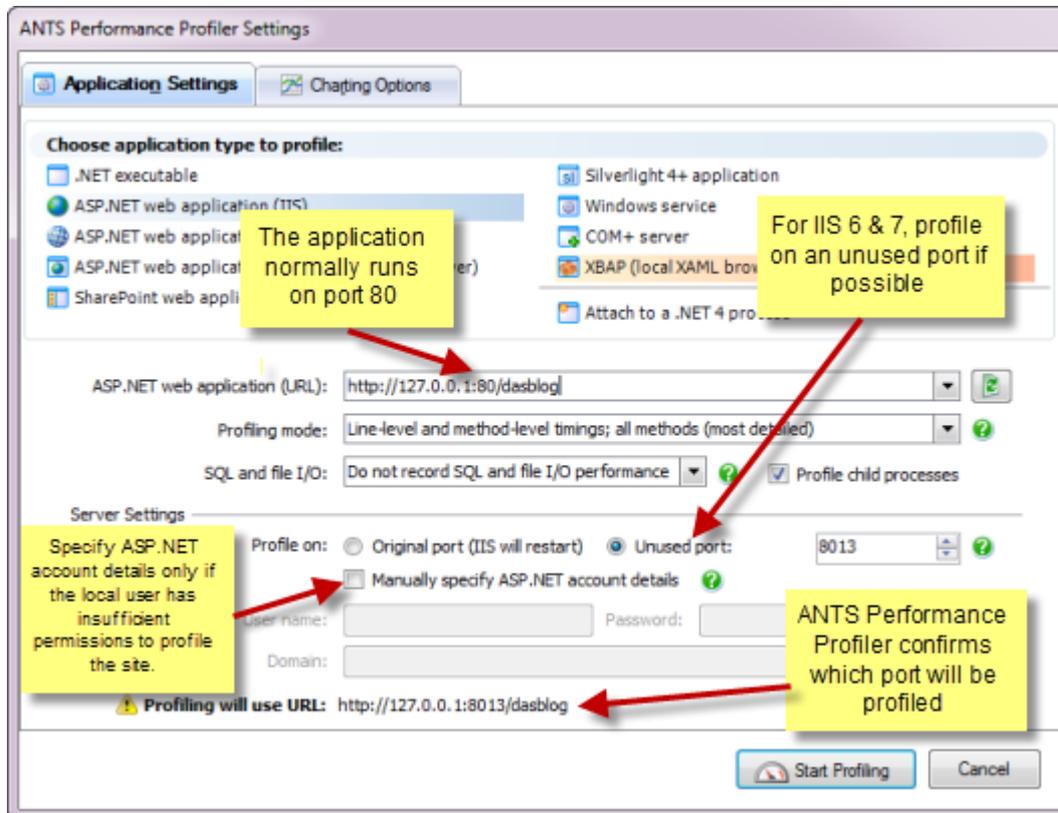
This option is available for IIS 6 and IIS 7 only.

With IIS 6 or IIS 7, ANTS Performance Profiler profiles your web application as the Windows Local System user by default. This is appropriate for most websites. If your web application connects to a remote server (such as a database server), for example, the Windows Local System user might not have appropriate permissions to make the remote connection. In this case, enter the credentials of a user who does have the required permissions. Note that the user you specify must be an administrator, and must have permission to read from `%ProgramFiles%\Red Gate\ANTS Performance Profiler 6\ProfilerCore.dll`

With IIS 5, your application is always profiled as the ASPNET user. Ensure that the ASPNET user has permission to read from %ProgramFiles%\Red Gate\ANTS Performance Profiler 6\ProfilerCore.dll

6. If required, change the performance counters to record; see Setting up Charting Options.

7. Click .



ANTS Performance Profiler launches the IIS user mode worker process (*w3wp.exe*), using a cut-down configuration file based on the site's *applicationHost.config* configuration file. Internet Explorer then starts and displays your web application.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.



Note that you can only profile ASP.NET applications running on the computer where the profiler is installed. You cannot profile web applications that use SSL, because *w3wp.exe* does not support running SSL websites outside of IIS.

See also Profiling an ASP.NET application (worked example).

If you encounter problems while trying to profile an application in IIS, see Troubleshooting IIS profiling ([http://www.red-gate.com/supportcenter/Content/ANTS\\_Performance\\_Profiler/help/6.3/APP\\_cannot\\_start\\_iis](http://www.red-gate.com/supportcenter/Content/ANTS_Performance_Profiler/help/6.3/APP_cannot_start_iis)).

## Profiling WCF services running on IIS

The procedure for profiling Windows Communication Foundation (WCF) services running in IIS is similar to the procedure used to profile other types of web application in IIS. It may help to think of the service as a server in a server-client relationship.

Please note the following:

- Before you start, change the WCF client contract to communicate on the unused port that you select in ANTS Performance Profiler (by default, 8013).

Changing the port is necessary because otherwise the client will communicate with the copy of the server hosted in IIS, not the copy in the worker process started by ANTS Performance Profiler.

- Set the **ASP.NET web application (URL)** to the path to the web application on the server.
- When you start profiling, Internet Explorer will launch.  
Minimize this window and interact with your client application instead. Do not close the Internet Explorer window during profiling; this will stop ANTS Performance Profiler from collecting the performance data.

## Profiling SharePoint

---

ANTS Performance Profiler can profile managed code that runs on a Microsoft SharePoint server because Microsoft SharePoint 2007 is implemented as an ASP.NET web application.

Please note that ANTS Performance Profiler was not designed to support SharePoint 2010, but a workaround may allow you to profile managed code running on a Sharepoint 2010 server with IIS 7. For details, see Profiling Sharepoint 2010.

### Setting up the Performance Profiler

To profile SharePoint, perform the following steps:

1. Open **Internet Information Server (IIS) Manager** and stop the website.
2. In ANTS Performance Profiler, on the **ANTS Performance Profiler Settings** dialog box, under **Choose application type to profile**, click **ASP.NET web application (IIS)**.
3. Enter the path to the ASP .NET web application that hosts your site collection. The path should be in the following format:  
*http://server:port/*

The server is the name of the local server and the port is the TCP port on which the web application normally runs. If the site collection is on the root virtual directory for the site, you must include the trailing slash.

Note that you must enter the path manually, because the drop-down list of available sites does not support SharePoint.

4. Select the required **Profiling mode, SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
5. Select the port on which to profile your application:
  - ◆ If you are using IIS 6 or IIS 7, select **Unused port** and choose a port that is not used by IIS.  
Note that this will not work if your application's code specifically binds to a specific port.
  - ◆ If you are using IIS 5, or if you are using IIS 6 or 7 and your application binds to a specific port, select **Original port**.  
IIS will restart so that the profiler can attach to the port.  
If IIS does not restart correctly, use IIS Manager to stop the website until you have finished profiling.

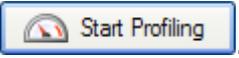
The port where the application will be profiled is displayed at the bottom of the ANTS Performance Profiler Settings dialog box.

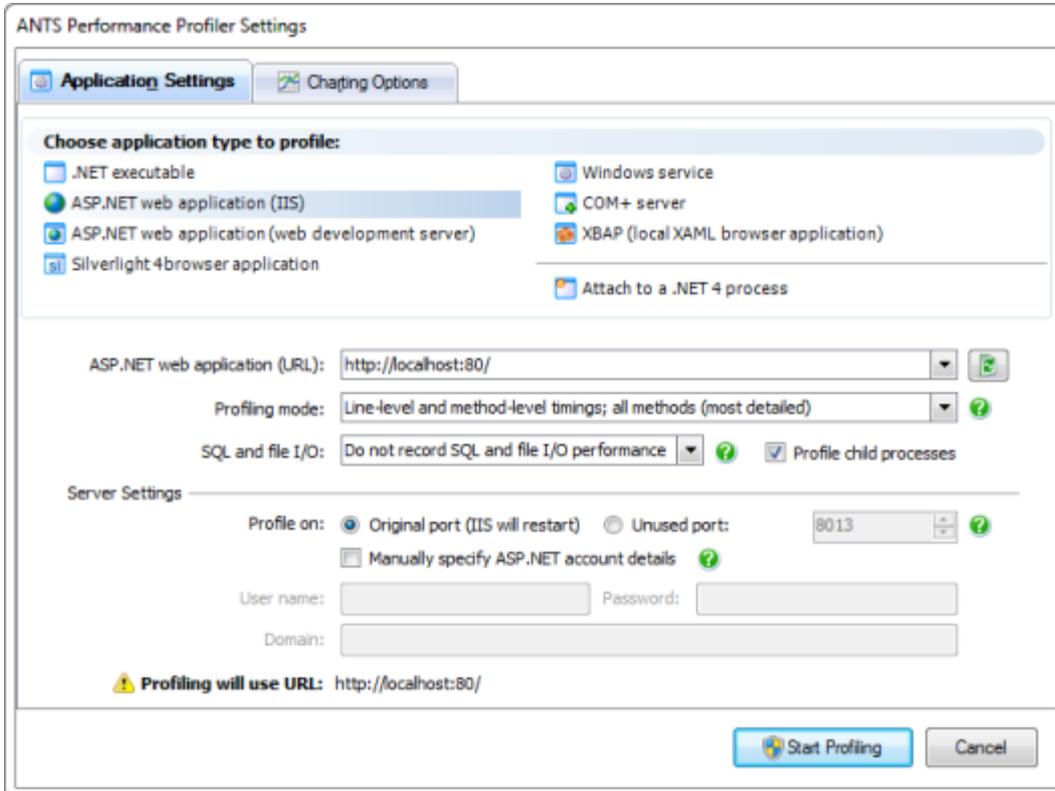
6. If required, select **Manually specify ASP.NET account details** and enter the **User name, Password** and **Domain**.

ANTS Performance Profiler profiles your web application as the Windows Local System user. If the Windows Local System user might not have appropriate permissions to

use SharePoint site collection, enter the credentials of a user who does have the required permissions. Note that the user you specify must be an administrator.

7. If required, change the performance counters to record; see Setting up Charting Options.

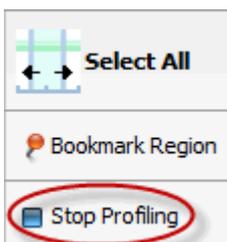
8. Click .



Use your SharePoint 2007 site collection as normal. Any additions that you have coded, such as web parts and lists, will be reflected in the ANTS Performance Profiler results if these objects have been accessed.

During a profiling session you can interact with the profiler whilst your site collection is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.



Note that you can only profile SharePoint site collections running on the computer where the profiler is installed.

**See also**

Troubleshooting SharePoint Profiling ..... 112

## Profiling ASP.NET applications running on the web development server

---

The ASP.NET web development server (also known as 'WebDev' or 'Cassini') is the built-in web server for your development environment. It is a good place to start debugging web applications because, unlike in IIS, you do not have to worry about configuration and security settings. You also do not have to worry about stopping and restarting the web server. The web development server does have limitations, however, so eventually you will want to test your web application under IIS. This is especially true if your web application has pages accessible only by users with the appropriate security settings.

You can profile a debug build of your ASP.NET web application while it runs in the web development server by following the instructions below.

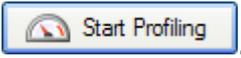
### Setting up the Performance Profiler

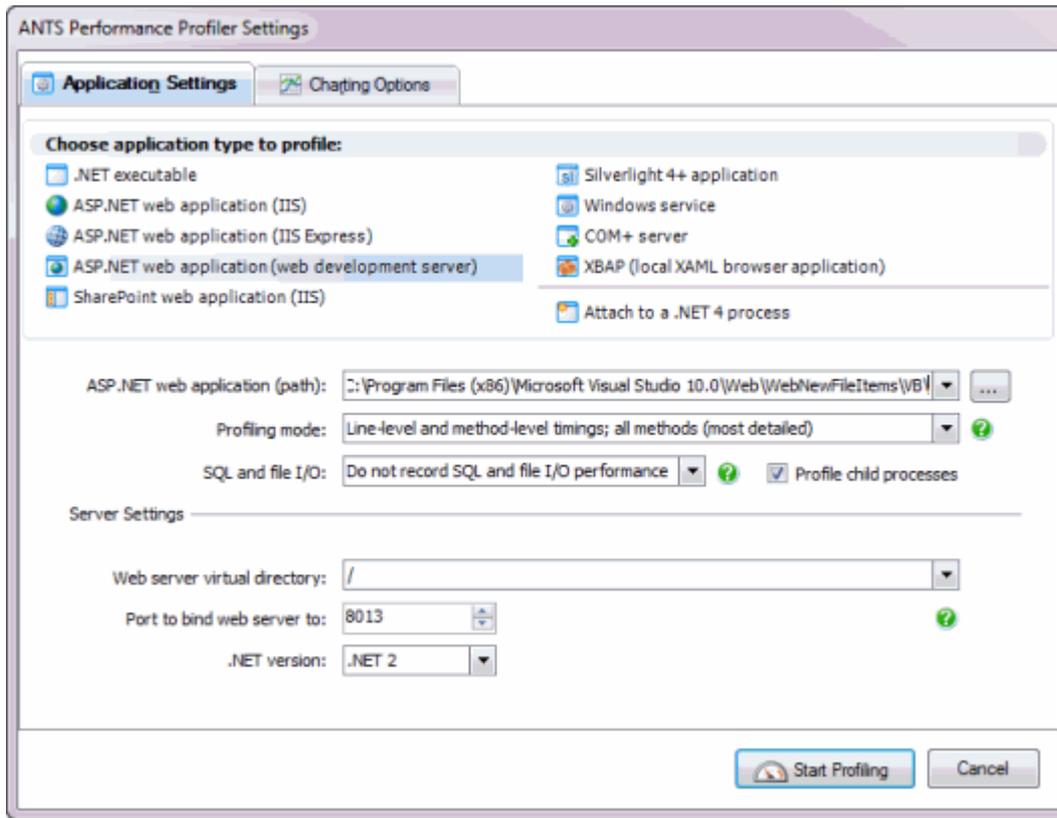
To profile ASP.NET applications running on web development server, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **ASP.NET web application (web development server)**.
2. Set the **ASP.NET web application (path)** for the web application that you want to profile.  

The path is where the file with the file extension `.aspx` is located.  
Note: You may find that the `.aspx` file is saved in the `/WebSites/` directory, not in the `/Projects/` directory where you would normally expect to find it.
3. Select the required **Profiling mode, SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. Under **Server Settings**, set **Web server virtual directory** to the application's virtual path on the web server.
5. In the **Port to bind web server to** box, set the port on which ANTS Performance Profiler should listen.  

For example, if you specify *staging* for the virtual directory and *8013* for the port number, your web application starts on URL `http://localhost:8013/staging/`.
6. Set the **.NET version** used by your web application.
7. If required, change the performance counters to record; see Setting up Charting Options.

8. Click .



The web development server starts and the web application is shown in Internet Explorer. Note that, although you can interact with the application using any web browser, closing the Internet Explorer instance opened by ANTS Performance Profiler will end your profiling session.

During a profiling session, while your application runs, you can obtain results in the profiler by selecting areas of the timeline.

When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.



This closes both the web development server and the ASP.NET application. ANTS Performance Profiler shows all of the profiling data collected for the application.

## Profiling Silverlight 4 browser applications

---

You can use ANTS Performance Profiler to profile Silverlight 4 applications.

Note that line-level timings are not available when profiling Silverlight applications, and that some performance counters are not shown on the timeline.

### Introducing Silverlight 4 applications

Microsoft Silverlight 4 applications can run either in a web browser, or in an Out-Of-Browser mode, depending on the setting chosen at compile-time.

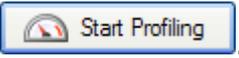
Silverlight applications that run in a web browser are contained in an HTML or ASPX file which may be stored:

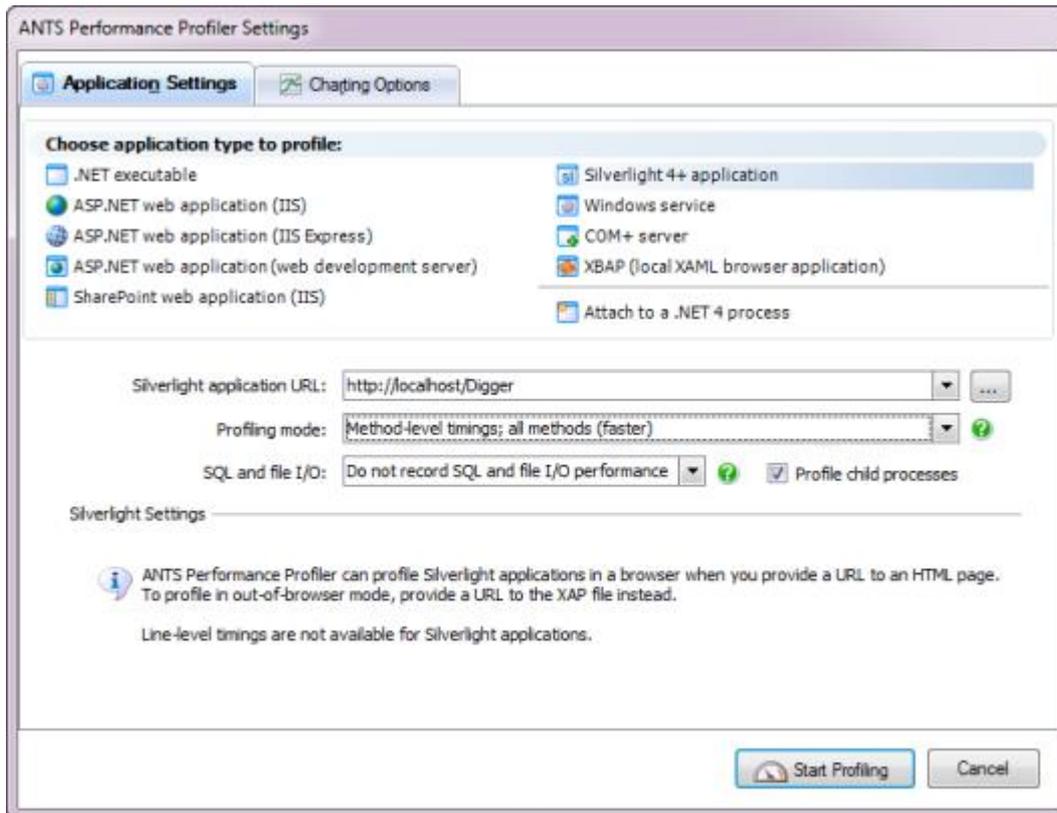
- in the local file system.
- on an HTTP server on the local computer.
- on an HTTP server on a remote computer.

### Setting up the Performance Profiler (Silverlight applications running in a web browser)

To profile Silverlight 4 browser applications running in a web browser, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Close all instances of Internet Explorer that are currently running on your computer.  
This ensures that ANTS Performance Profiler connects to the correct instance of *iexplore.exe* when you start profiling.
2. Under **Choose application type to profile**, click **Silverlight 4 browser application**.
3. For the **Silverlight application URL**, enter either the local file path (prefixed by *file:///*) to the HTML file that embeds the Silverlight application, or the URL of the HTML or ASPX file that embeds the application.  
Note that the URL must be a server running on the same computer as the computer where ANTS Performance Profiler is installed.
4. Select the required **Profiling mode**, **SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.  
Note that line-level timings are not available for Silverlight applications.
5. If required, change the performance counters to record; see Setting up Charting Options.

6. Click .



Microsoft Internet Explorer launches, and the Silverlight application is shown.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

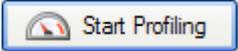
When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.

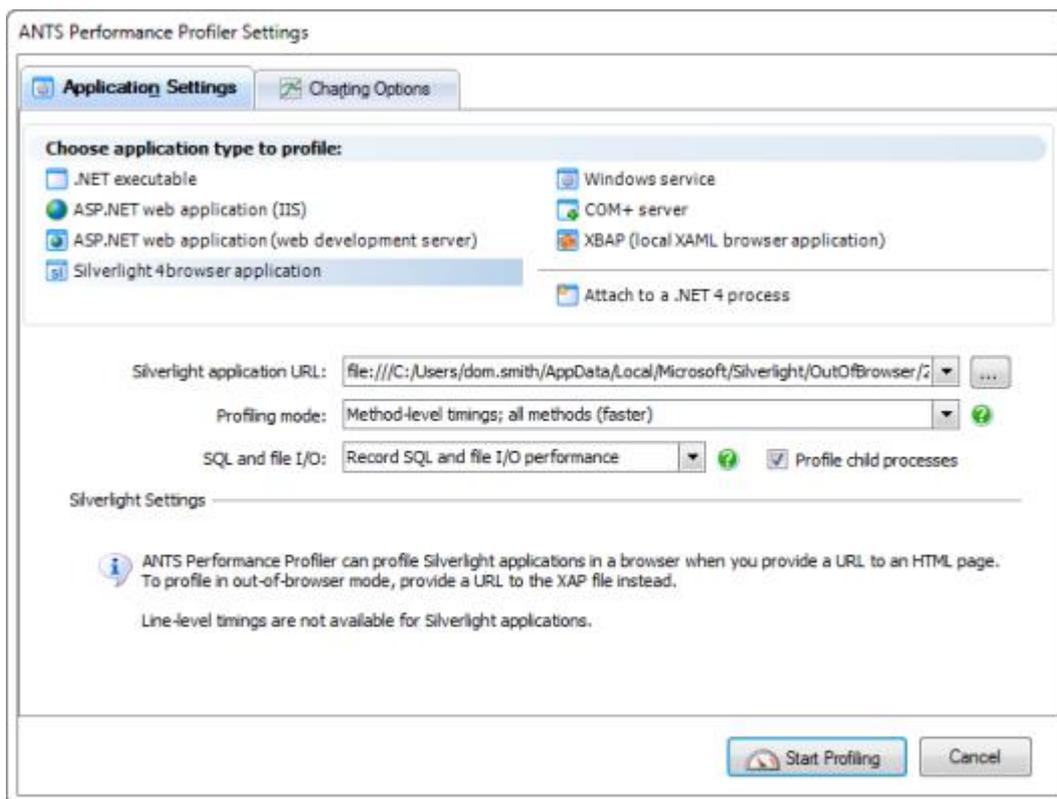


### Setting up the Performance Profiler (Silverlight applications running in Out-Of-Browser mode)

To profile Silverlight 4 browser applications running in Out-Of-Browser mode, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **Silverlight 4 browser application**.
2. For the **Silverlight application URL**, enter either the local file path (prefixed by *file:///*) to the XAP file.  
This is normally found in *%LOCALAPPDATA%\Microsoft\Silverlight\OutOfBrowser\*.
3. Select the required **Profiling mode**, **SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.  
Note that line-level timings are not available for Silverlight applications.
4. If required, change the performance counters to record; see Setting up Charting Options.

5. Click .



The Microsoft Silverlight Out-Of-Browser launcher starts, and the Silverlight application is shown.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your web application, click the  **Stop Profiling** button in ANTS Performance Profiler.



## Troubleshooting

If you experience problems:

- Ensure that the Silverlight application is compiled against Silverlight 4. Older versions of Silverlight may appear to work but will give inaccurate results.
- Check that the Silverlight application runs correctly on the computer which you are profiling it on, without the profiler attached.
- Note that you may need to run ANTS Performance Profiler as an Administrator for some types of Silverlight application and computer configuration.

## Profiling Windows services

---

A Windows service is a long-running executable that performs specific functions and which is designed not to require user intervention.

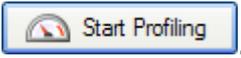
To profile a Windows service, it must be installed on the computer on which the profiling will take place. You can install your service using the *installutil.exe* utility which is supplied with Microsoft Visual Studio.

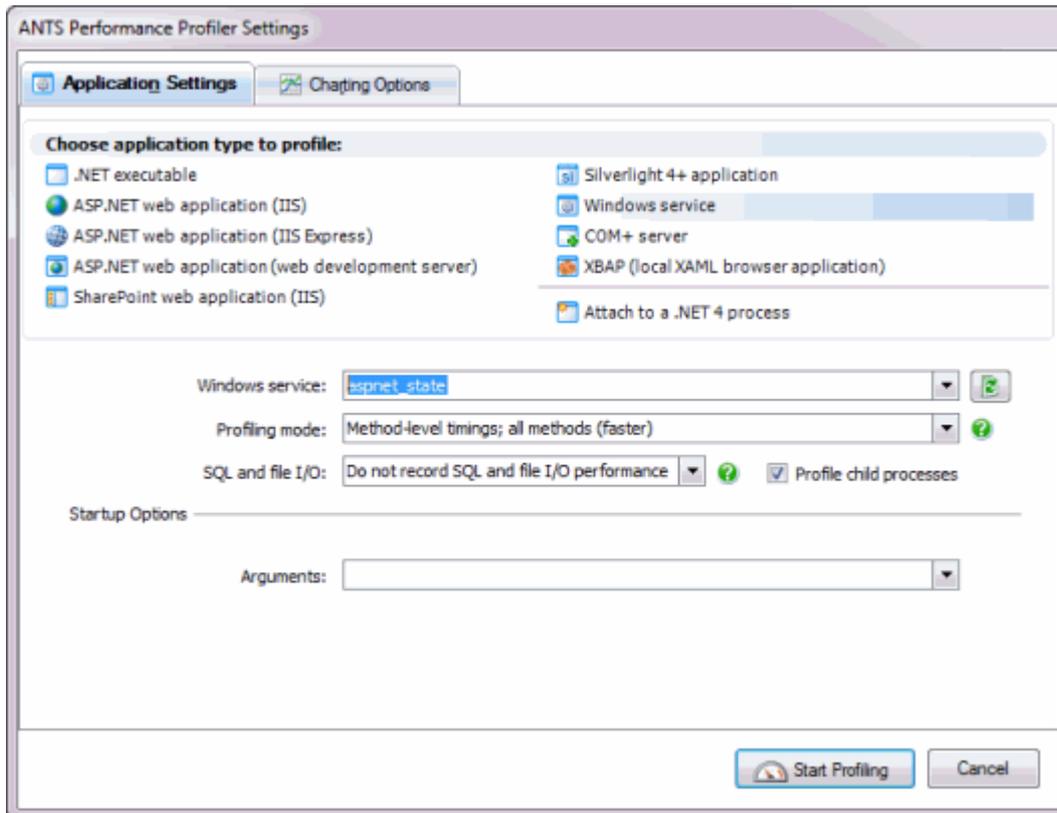
To obtain results with source code, you must use a debug build of your .NET Windows service.

### Setting up the performance profiler

To profile Windows services, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **Windows service**.
2. Select your .NET **Windows Service** from the drop down list. The service here is named *Sample Service*.
3. Select the required **Profiling mode, SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. If required, specify any **Arguments** that are to be used when running the service.
5. If required, change the performance counters to record; see Setting up Charting Options.

6. Click .

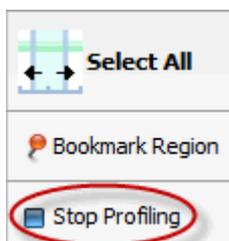


If the service is not already started, ANTS Performance Profiler will start the service.

If the service is already started, ANTS Performance Profiler will restart the service.

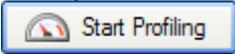
During a profiling session you can obtain results by selecting areas of the timeline.

When you have finished profiling the service, click the  **Stop Profiling** button in ANTS Performance Profiler.



## Profiling WCF services

If the service that you want to profile is implemented using the Windows Communication Foundation (WCF), think of the service as a server in a server-client relationship.

Set up ANTS Performance Profiler as described above but after you have clicked , start interacting with the client program to call the service.

The service's communications with the client are included in the results.

## Profiling COM+ server applications

---

To profile a COM+ server application correctly, there are two main steps:

1. Change the COM+ server so that it can be profiled.
2. Set up ANTS Performance Profiler.

### Change the COM+ server so that it can be profiled

ANTS Performance Profiler can only profile COM+ server applications which are activated in a process provided by the system, not by the client. To do this, set the `ApplicationActivation` attribute as follows:

```
[assembly: ApplicationActivation(ActivationOption.Server)]
```

If you cannot set this attribute (for example, if you do not have access to the source code), you may still be able to profile the COM+ application by profiling the client application. You should note, however, that:

- the resulting server application will not be a true COM+ application and will run in the client process
- ANTS Performance Profiler will profile the client application, and will treat the COM+ server as a DLL invoked by the client.

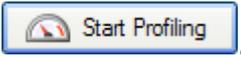
### Set up ANTS Performance Profiler

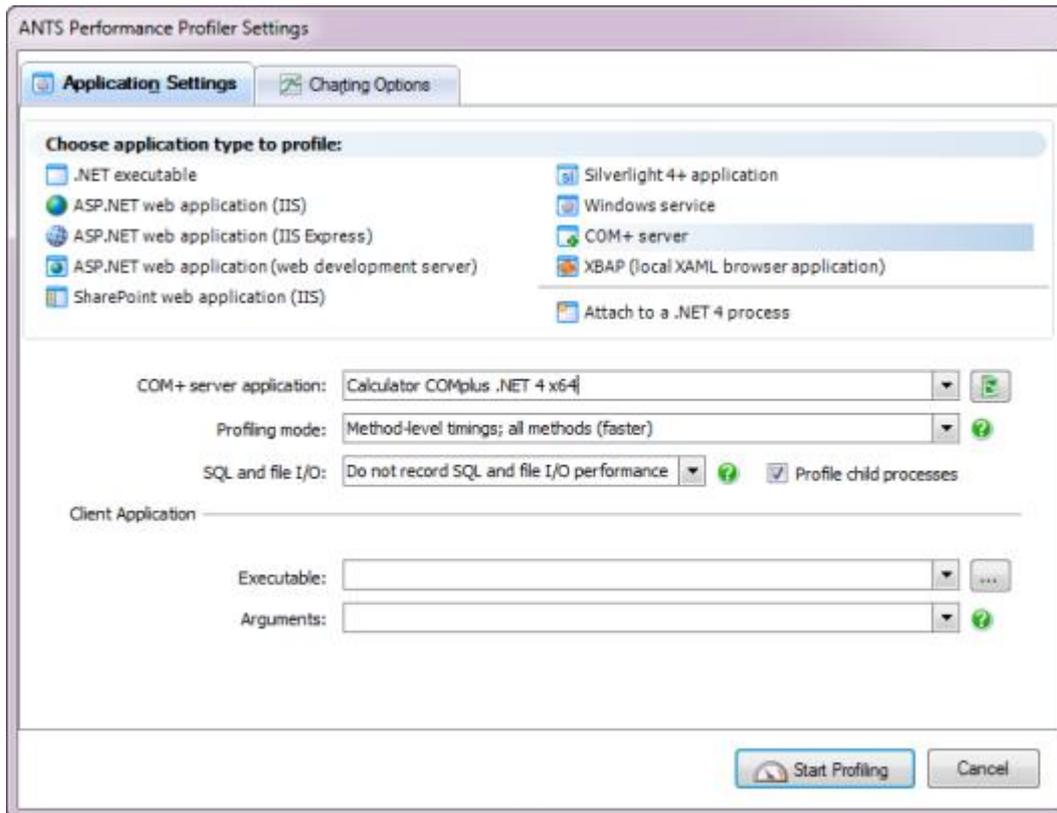
To profile COM+ server applications, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **COM+ server**.
2. Use the dropdown list to select the **COM+ server application** that you want to profile.

Click  to update the list of COM+ server applications.

3. Select the required **Profiling mode, SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.
4. If you want to profile a client application's communications with the COM+ server:
  - a. Browse to the client application **Executable**.
  - b. If required, specify any command line **Arguments** that are to be used when running the client application.
5. If required, change the performance counters to record; see Setting up Charting Options.

6. Click .

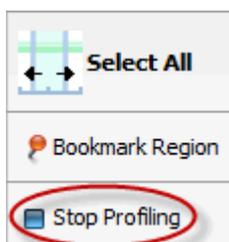


If the client application was specified, the client application starts.

If the client application was not specified, start it manually.

During a profiling session you can interact with the client application whilst your COM+ server is still being profiled. You obtain results by selecting areas of the timeline.

When you have finished interacting with your client application, and you are ready to finish profiling the COM+ server, click the  **Stop Profiling** button in ANTS Performance Profiler.



## Profiling remote COM+ applications

The security architecture of COM+ does not allow COM+ applications to be started from a Remote Desktop session with a GUI, or a terminal services session.

To profile COM+ applications remotely, type `mstsc /console` at a command prompt to start a Remote Desktop session in console mode.

Note that all users who are logged on to the computer that you are connecting to will be logged off when you connect.

## Troubleshooting

Some COM+ server applications need to be fully trusted before being profiled. If profiling the COM+ server does not work, you may need to make the application's code fully trusted by setting the `ApplicationAccessControl` attribute as follows:

```
[assembly: ApplicationAccessControl(false)]
```

Note that you should not normally release the COM+ application in this trusted state.

## Profiling XBAP applications

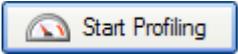
---

The procedure for profiling XBAP applications depends on whether the XBAP is locally-hosted or remotely-hosted.

### To profile locally-hosted XBAP applications

To profile locally-hosted XBAP applications, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

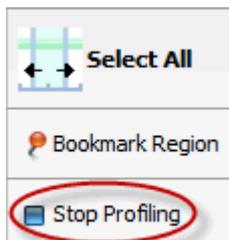
1. Under **Choose application type to profile**, click **XBAP (local XAML browser application)**.
2. Navigate to the **XBAP application** that you want to profile.
3. If required, change the performance counters to record; see Setting up Charting Options.

4. Click .

Internet Explorer starts, and displays the XBAP application. Use the XBAP application normally.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your XBAP application, click the  **Stop Profiling** button in ANTS Performance Profiler.



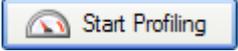
### To profile remotely-hosted XBAP applications

To profile remotely-hosted XBAP applications, on the **ANTS Performance Profiler Settings** dialog box, perform the following steps:

1. Under **Choose application type to profile**, click **.NET executable**.
2. Choose Internet Explorer as the **.NET executable** that you want to profile.  
Internet Explorer is normally located in *%ProgramFiles%/Internet Explorer/*
3. Select the required **Profiling mode**, **SQL and file I/O**, and **Profile child processes** options; see Working with Application Settings.

Note that line-level timings are not available when profiling XBAP applications.

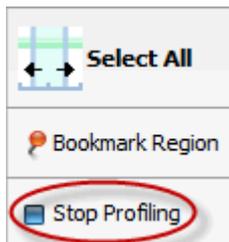
4. If required, change the performance counters to record; see Setting up Charting Options.

5. Click .

Internet Explorer starts. Navigate to the page which embeds the XBAP application, and use the XBAP application normally.

During a profiling session you can interact with the profiler whilst your application is still being profiled, and obtain results by selecting areas of the timeline.

When you have finished interacting with your XBAP application, click the  **Stop Profiling** button in ANTS Performance Profiler.



## Attaching to a running .NET 4 process

---

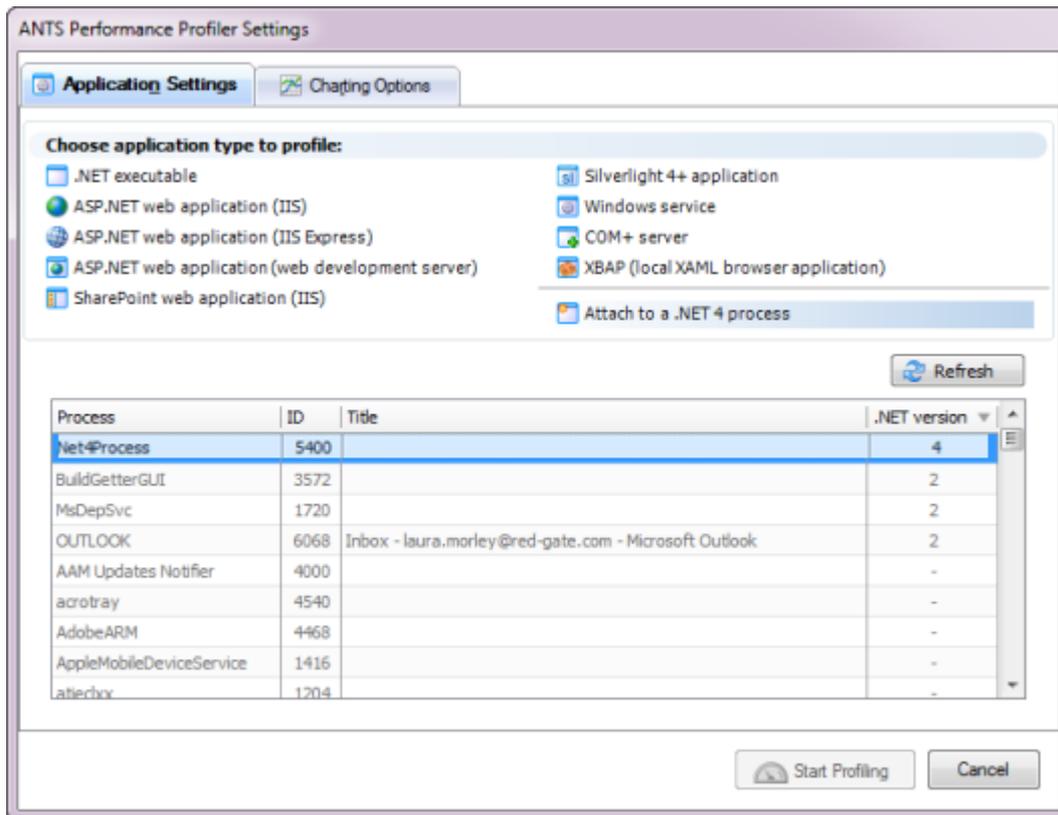
ANTS Performance Profiler can attach to a .NET 4 process which is already running. This feature allows you to start profiling your applications when performance problems are noticed, without having to restart the application from scratch. This topic explains how to do this, using a simple C# program (called *Net4Process.exe*), which counts down for 300 seconds.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Net4Process
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Countdown test process");
            int i = 300;
            int[] array = new int[301];
            while (i >= 0)
            {
                Console.WriteLine(i + " seconds");
                array[i] = i;
                i--;
                System.Threading.Thread.Sleep(1000);
            }
        }
    }
}
```

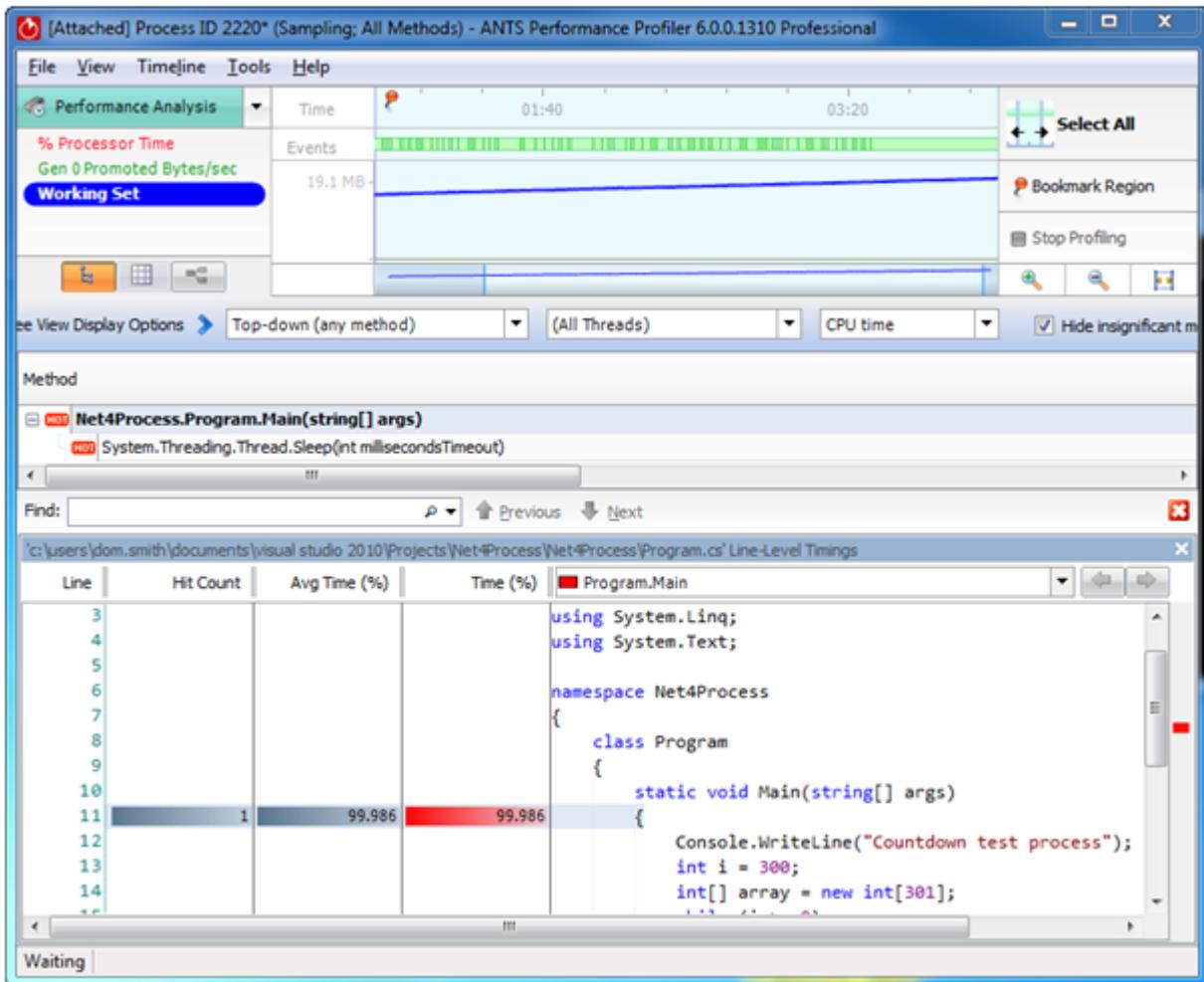
1. Start *Net4Process.exe* from the command prompt.
2. When the countdown starts, open ANTS Performance Profiler.
3. In the ANTS Performance Profiler Settings dialog, under **Application Settings**, select Attach to a .NET 4 process.

A list of the processes currently running is displayed with the .NET framework version that they use. Processes that are not .NET 4 processes are unavailable.



4. Select *Net4Process* and click **Start Profiling**.
5. When the application has finished (or if you press ANTS Performance Profiler's **Stop Profiling** button), the results are shown in the call tree in ANTS Performance Profiler.

As expected, almost 100% of the time is spent in `Main()` and the increasing amount of memory used by the program can be seen in the timeline.



If you need to save results for analysis later, attaching ANTS Performance Profiler to a running process means that you can save just the results you need, helping to reduce the size of the results file. This is particularly useful for line-level timings for all methods.

## Profiling SQL queries

---

You can use ANTS Performance Profiler 6.3 Professional to profile database queries sent by an application to a Microsoft SQL server instance.

You might want to profile SQL queries if performance timings have revealed that a line of code involving a database query is particularly slow.

Note that:

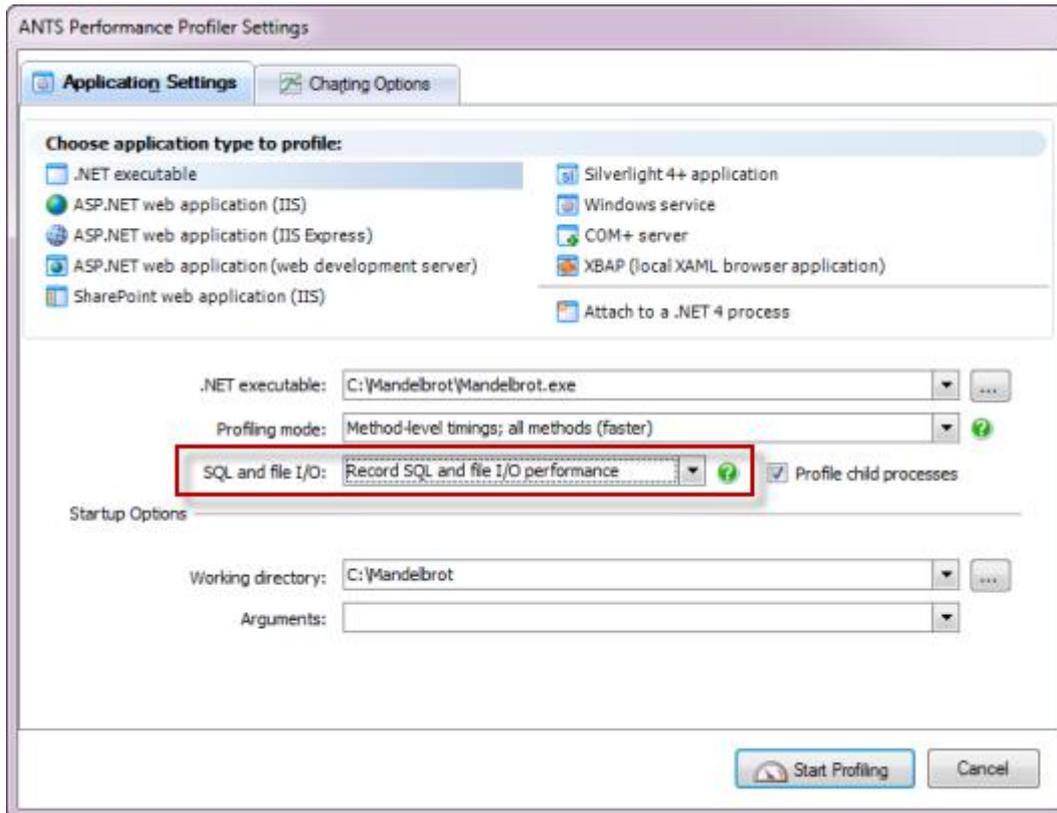
- profiling SQL queries is not possible with SQL Server Express: Because SQL Server Express does not expose performance counters, ANTS Performance Profiler cannot obtain results.
- the SQL Server instance must be on the same computer that the profiler is running on.
- you can only profile SQL queries on Windows Vista or later, or Microsoft Server 2008.

### Setting up SQL profiling

Before profiling SQL, we recommend that you check for performance issues in your code (and any third-party code). Profile SQL when you have identified a slow line of code that involves a SQL query.

Set up a new profiling session using the Application Settings dialog.

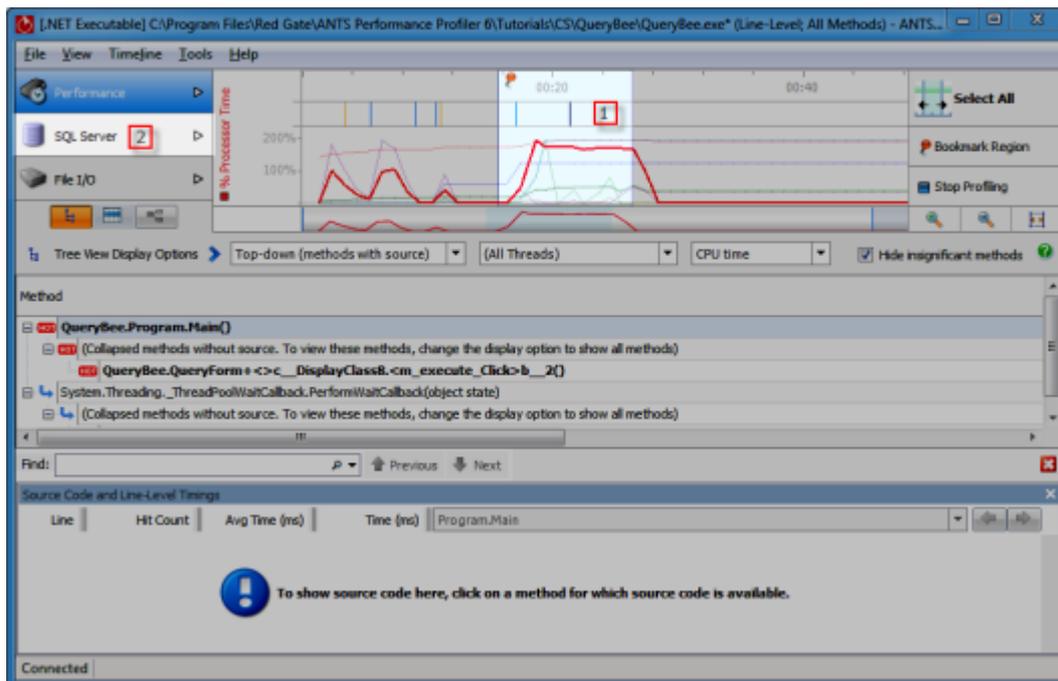
Ensure that **SQL and file I/O** is set to **Record SQL and file I/O performance**.



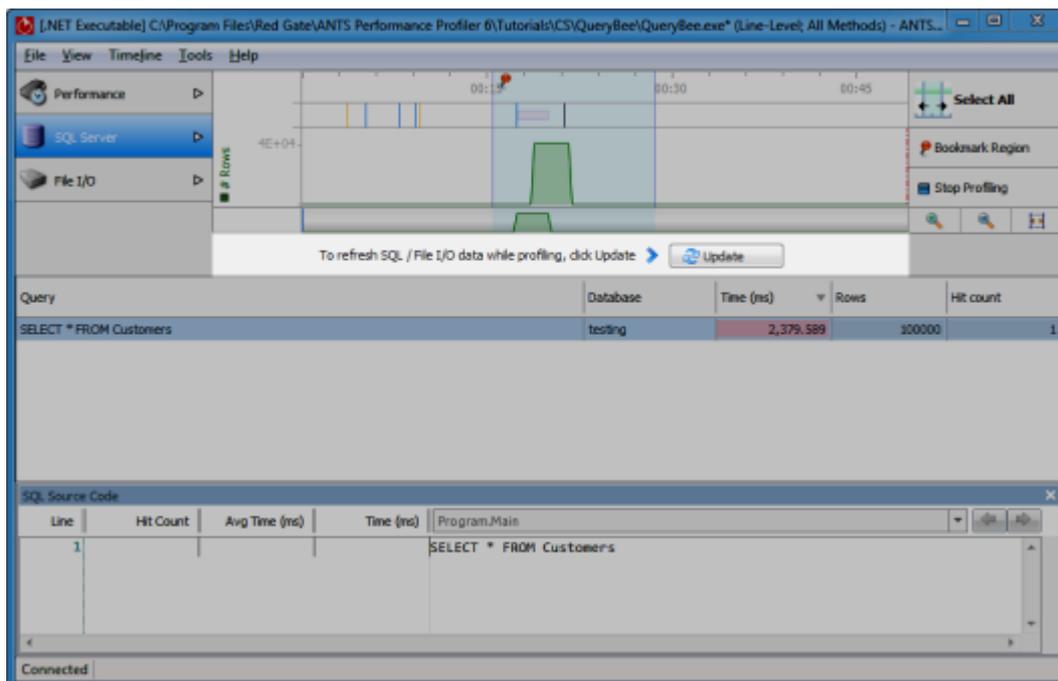
## Viewing SQL results while profiling

1. Drag to select the portion of the timeline that you are interested in.

2. Click **SQL Server**.



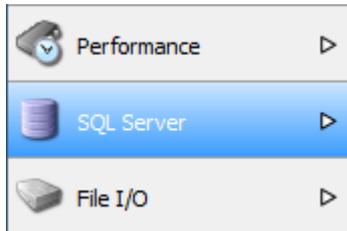
3. Note that the timeline is not automatically updated while profiling SQL queries. To display queries performed since you switched to SQL Server view, click **Update**.



The timeline and the Query panel update to show the latest data.

## Viewing SQL results after profiling

If profiling is not currently in progress, click **SQL Server**.



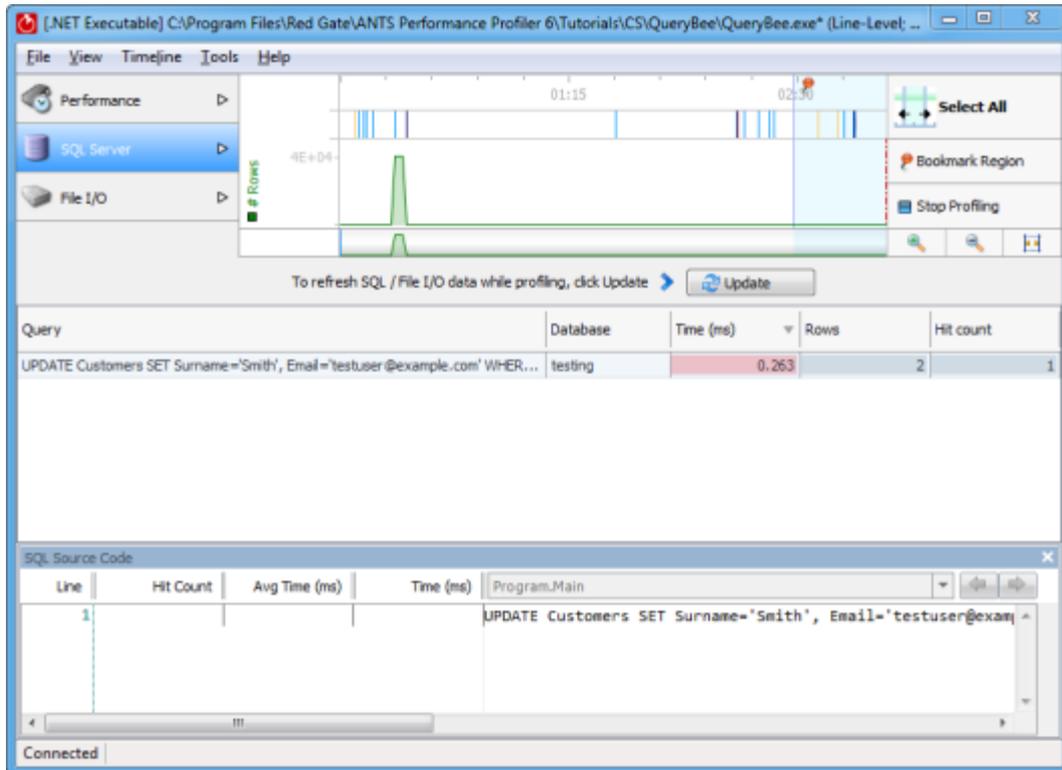
SQL results for the entire complete profiling session are displayed on the timeline.

## Tips

### Long-length queries

If the SQL query is a multi-line query, or is just very long, it may be truncated in the Query list. To display the full query, select it. The query is shown in the scrollable SQL Source Code panel.

Note that the Hit Count, Avg Time and Time columns are empty in the SQL Source Code panel, because line-level timings are unavailable for SQL Server. See the Hit Count and Time columns in the Query list to view the time taken by the entire query.

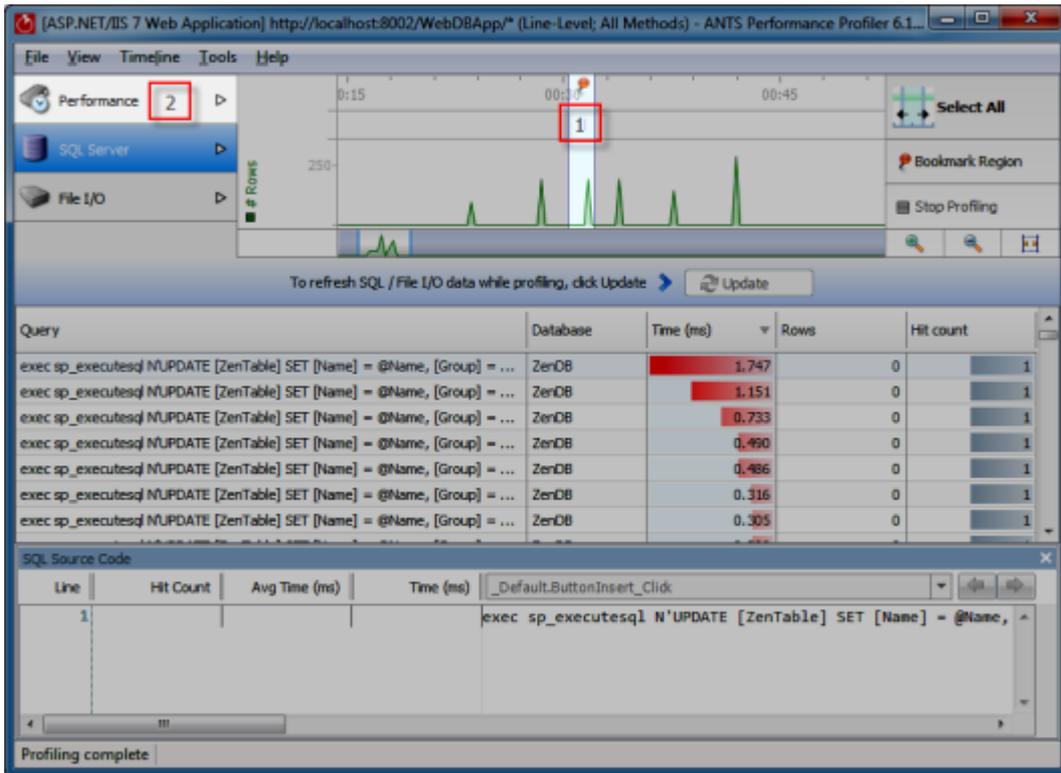


### Linking back to your code

To find out which of your code's methods ran a particular SQL query:

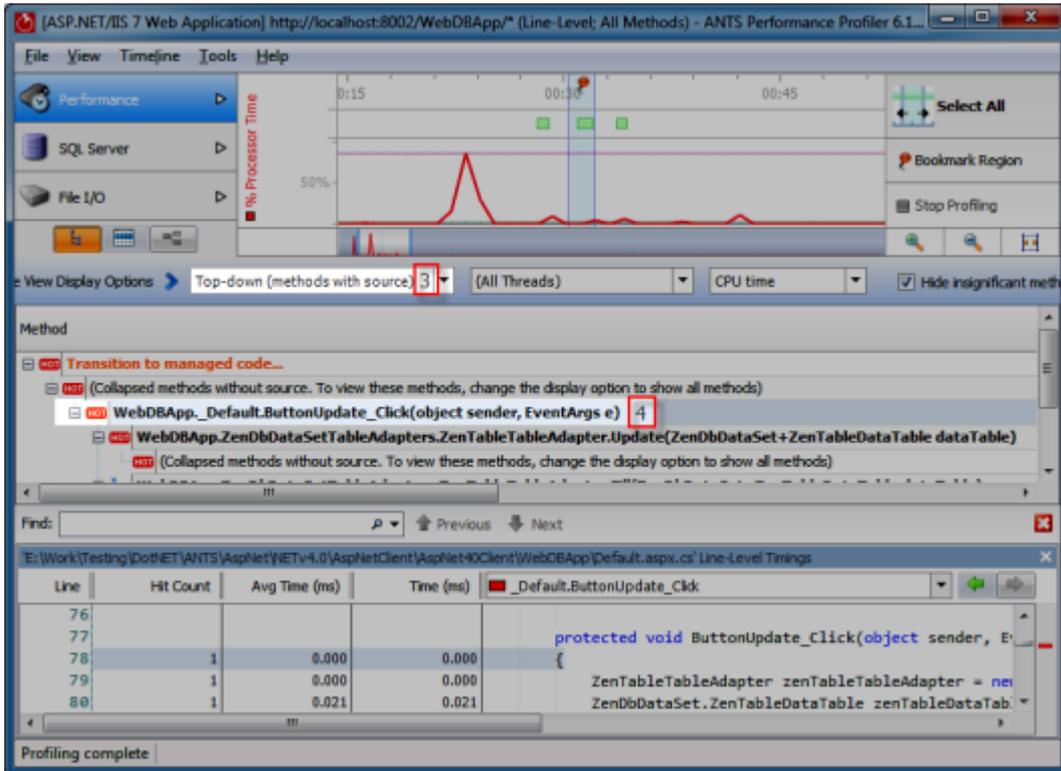
1. Select the time when the query ran on the timeline. Include some time just before the SQL query ran, and the server load increased: this will ensure your selected range includes the time when the method that runs the query was called.

2. Switch back to **Performance** view.



3. Under Tree View Display Options, select **Top-down (methods with source)**.

4. In the tree view, the method that ran the SQL query should be found near the highlighted method.



Browse the line-level timings to find code that could be optimized.

## Profiling File I/O

---

You can use ANTS Performance Profiler 6.3 Professional to profile when the application that you are profiling reads from discs or writes to discs (including network drives).

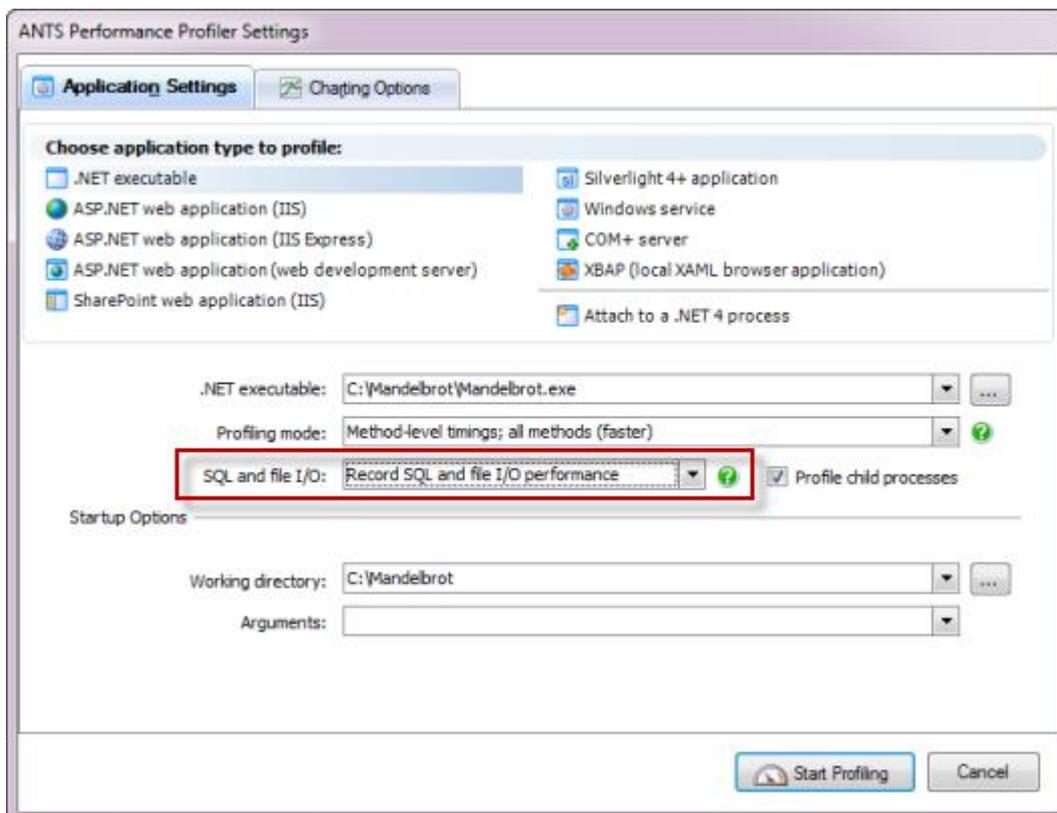
Note that you can only profile File I/O on Windows Vista or later, or Microsoft Server 2008.

### Setting up File I/O profiling

Before profiling File I/O, we recommend that you check for performance issues in your code (and any third-party code).

To enable File I/O profiling, set up the profiler in the same way that you would configure the settings for performance profiling. Ensure that **SQL and file I/O:** is set to **Record SQL and file I/O performance**.

While profiling in Performance view, set the Tree View Display Options to show **Wall-clock time**, because CPU time does not include time spent blocked waiting for File I/O. When you have identified a slow line of code, which involves a reading from or writing to a disc, profile File I/O.



## Viewing File I/O results while profiling

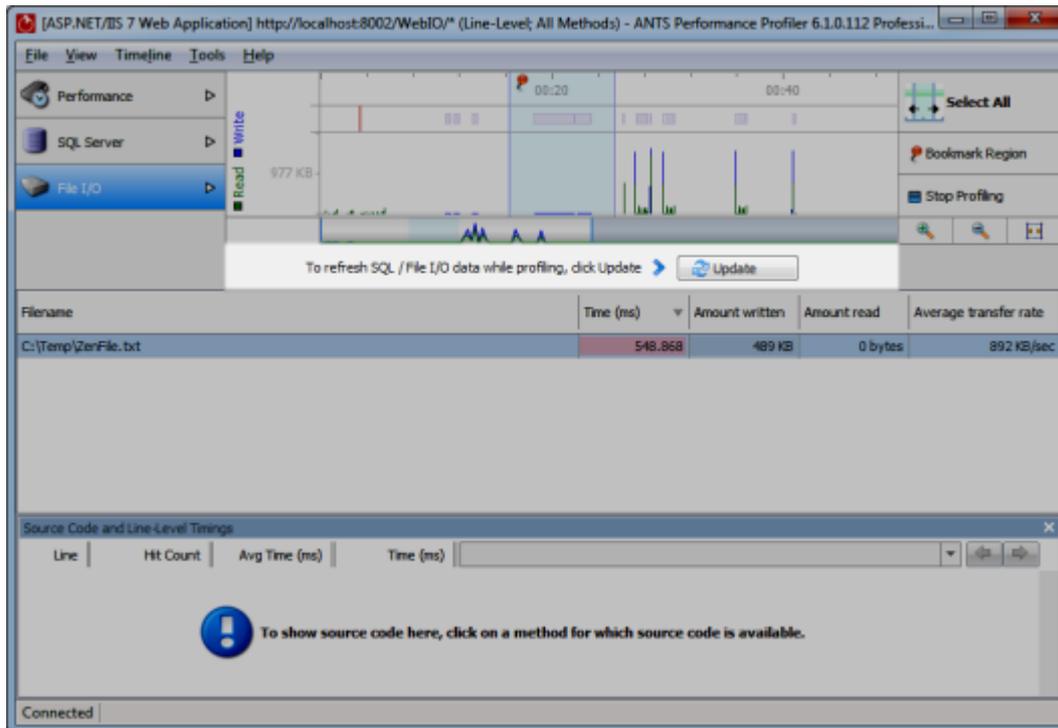
1. Drag to select the portion of the timeline that you are interested in.
2. Click **File I/O**.

The screenshot shows the ANTS Performance Profiler interface. The top menu includes File, View, Timeline, Tools, and Help. The left sidebar has a tree view with 'Performance', 'SQL Server', and 'File I/O' (highlighted with a red box '2'). The main area displays a timeline with a red line representing '% Processor Time' and a blue selection box (labeled '1') over a period from 00:20 to 00:40. Below the timeline, a table lists methods with their execution times and hit counts.

Method	Time (ms)	Time With Children (ms)	Hit Count
Transition to managed code...	<0.001	4,318.677	1
(Collapsed methods without source. To view these methods, change the display option ...)	9.676	4,318.677	1
WebIO_Default.ButtonWrite_Click(object sender, EventArgs e)	5.166	4,309.001	1

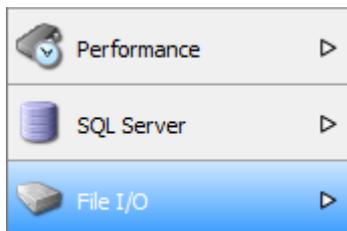
Below the table, there is a 'Find:' search bar and a 'Source Code and Line-Level Timings' section with columns for Line, Hit Count, Avg Time (ms), and Time (ms). A message box states: 'To show source code here, click on a method for which source code is available.'

- Note that the timeline is not automatically updated while profiling File I/O. To display queries performed since you switched to File I/O view, click **Update**.



## Viewing File I/O results after profiling

If profiling is not currently in progress, click **File I/O**.

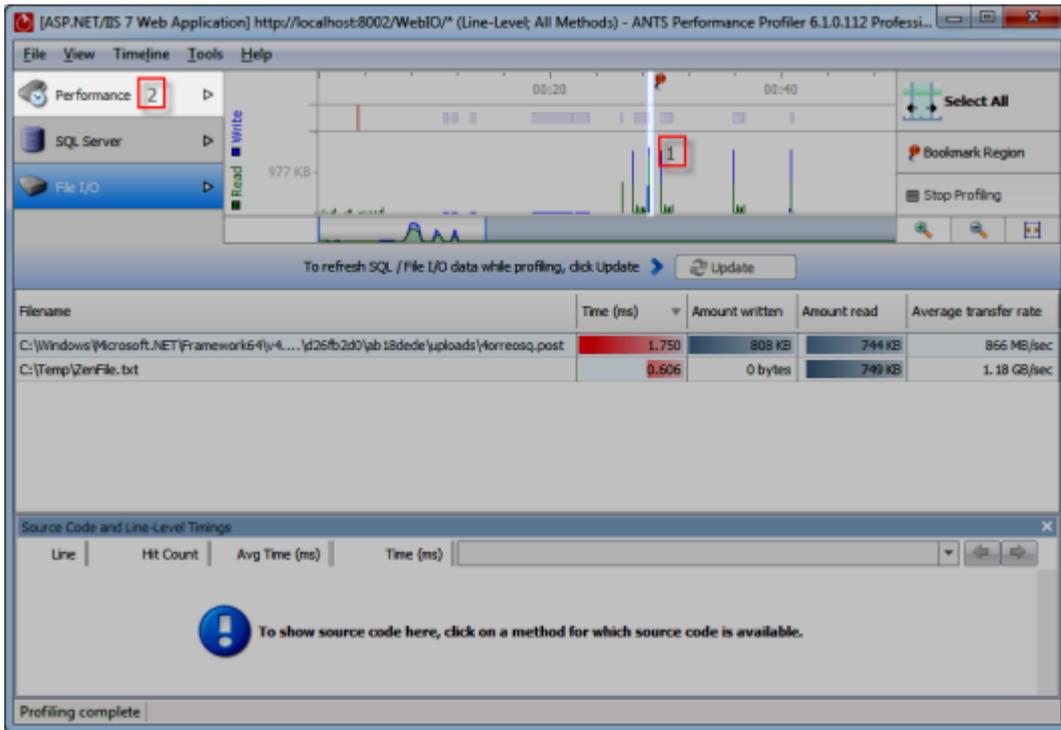


## Linking back to your code

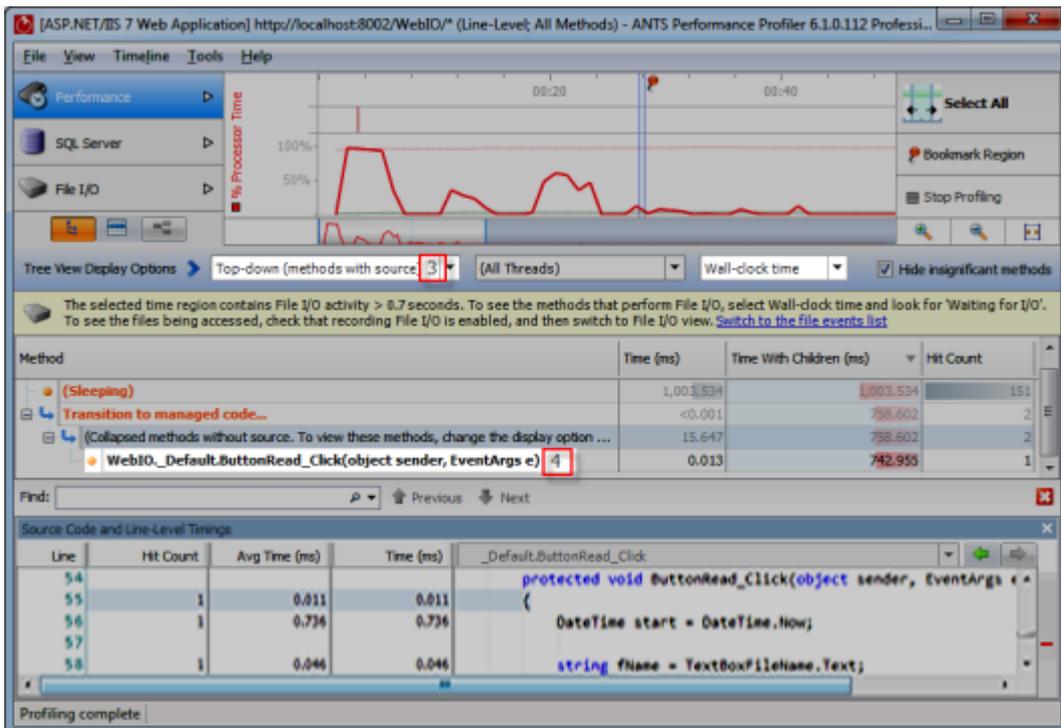
To find which of your code's methods caused File I/O to occur:

- Select the time when the I/O occurred on the timeline. Include some time just before the I/O because the method which caused the I/O will be called before the I/O takes place.

2. Switch back to **Performance** view.



3. Under Tree View Display Options, select **Top-down (methods with source)**.
4. In the tree view, the method which caused the File I/O should be found near the highlighted method.



Use the line-level timings to look for code that could be optimized.

## Setting up MSTest

---

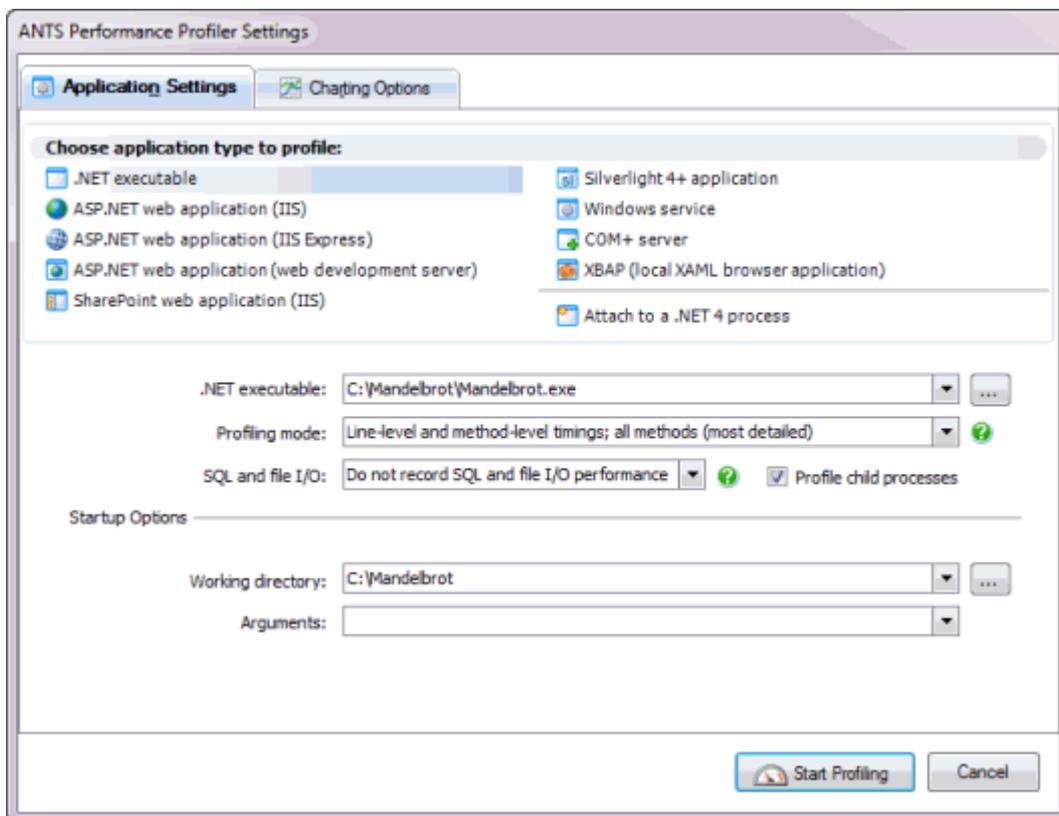
MSTest is a software unit testing framework developed by Microsoft, which lets you create, manage, and run unit tests from within the Visual Studio IDE, as well as from the command line.

This topic assumes that you are familiar with MSTest, and have already built an assembly full of tests that you want to profile.

### Setting up the performance profiler

Before you start profiling, you will need a debug build of your test assembly.

Start ANTS Performance Profiler (or use the New Profiling Session option on the File menu if ANTS Performance Profiler is already running).



On the **Performance Profiler Settings** dialog box:

1. Set the **application type** to profile to **.NET executable**.

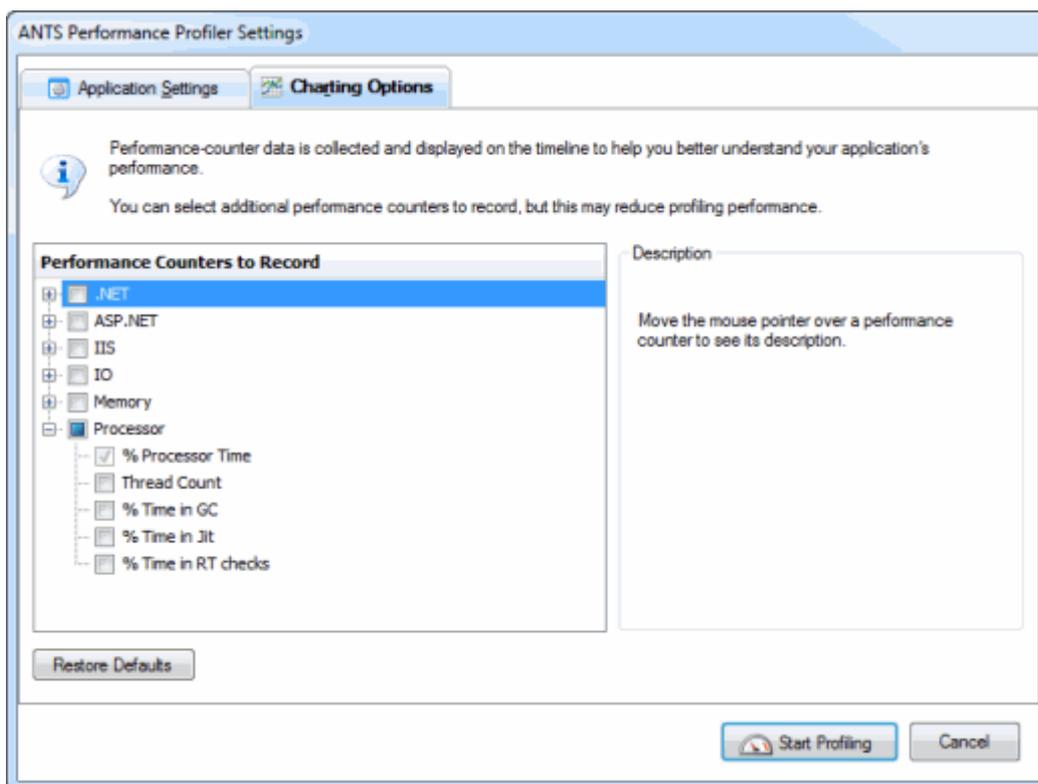
2. Set the **.NET executable** box to the path where *MSTest.exe* is installed. For example: *C:\Programs\Microsoft Visual Studio 9.0\Common7\IDE\MSTest.exe*

### Setting the arguments

In the **Arguments** box, set a `/testcontainer` argument to tell MSTest the path to the assembly that contains all of the tests. For example:

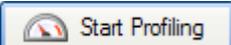
```
/testcontainer:"C:\Profiles\<<USER NAME>\My Documents\Visual Studio 2008\Projects\LoginForm\MyTests\bin\Debug\MyTests.dll"
```

On the **Performance Profiler Settings** dialog select the **Charting Options** tab.



Choose the performance counters which you want to record.

### Starting the profiler

Click .

MSTest starts and executes all of the tests contained within the assembly.

## Getting results

During a profiling session, you can interact with the profiler while your tests are still being profiled.

To finish the tests, you can either wait for MSTest to complete and exit, or press the **Stop Profiling** button in ANTS Performance Profiler.



ANTS Performance Profiler shows the results of the tests. You can investigate the results in exactly the same way as with any .NET application. This allows you to spot any big bottlenecks in your tests quickly.

## Profiling from the command line

---

You can profile applications from the command prompt in ANTS Performance Profiler Professional.

To run a profiling session from the command prompt, run *Profile.exe* with the appropriate options.

For example, to profile an executable called SimpleApp.exe using line-level timings and saving results as a CSV, use:

```
Profile.exe /e:"C:\testing\SimpleApp.exe" /ll /csv:"C:\testing\results.csv"
```

Profiling applications from the command prompt is useful if you want to integrate performance profiling in an automated test procedure; see Integrating ANTS Performance Profiler in a test procedure.

### Profiling IIS from the command line

You cannot currently profile applications which run on IIS from the command line. To work around this limitation:

1. Profile IIS from the ANTS Performance Profiler graphical user interface.
2. When you have finished profiling, on the **File** menu, click **Save Project**.
3. Use the `/project` argument on the command line to load the saved project.

### List of command line arguments

`/help` (Alias: `/?`)

Displays this help message. Use in conjunction with `/verbose` for more detailed information.

If this switch is used with any switches other than `/verbose`, `/html`, `/out`, `/force` or `/outputwidth` then those switches will be ignored, the help message will be printed, and 0 will be returned as the process exit code.

`/html`

Causes help to be output as HTML.

Must be used with the `/help` switch.

`/quiet` (Alias: `/q`)

Quiet mode - no output.

`/verbose` (Alias: `/v`)

Verbose mode.

`/force` (Alias: `/f`)

Forces overwriting of output files that already exist. If this flag is not set and a file already exists then program will exit with an exit code indicating an I/O error.

`/argfile:<argfile>`

File containing the XML argument specification.

`/out:<fileName>`

Redirects console output to the specified file.

`/project:<project>` (Alias: `/p`)

A performance profiler project that should be used to begin profiling.

`/executable:<executable>` (Alias: `/e`)

An executable to run in the profiler.

`/arguments:<arguments>` (Alias: `/args`)

The arguments to pass to the executable.

`/workingDirectory:<workingDirectory>` (Alias: `/wd`)

The working directory to use when profiling the application.

`/port:<port>`

The port that web applications should be profiled on. (Default: 8013)

`/cassini` (Alias: `/webdev`)

Profiling should be performed on the ASP.NET web development server (Cassini).

`/cassiniPath:<cassiniPath> (Alias: /cpath)`

The location of the ASP.NET project to use with the ASP.NET web development server (Cassini).

`/cassiniVirtualDirectory:<cassiniVirtualDirectory> (Alias: /cvd)`

The virtual directory to use for the ASP.NET web development server (Cassini).

`/cassiniNetVersion:<cassiniNetVersion> (Alias: /cnv)`

If Visual Studio 2010 is installed, this is the version of .NET to use when running the web development server (Cassini). Default: 0

`/service:<service>`

The name of a Windows service to profile.

`/complus:<complus>`

The name of a COM+ server to profile.

`/silverlight:<silverlight>`

The URL of a site containing a Silverlight application to profile.

`/RecordSqlIo (Alias: /rs)`

The profiler should try to record SQL and File I/O events.

`/profileSubprocesses (Alias: /sp)`

The profiler should profile both the target and any child processes it spawns.

`/timeout:<timeout> (Alias: /t)`

The number of seconds to wait before terminating the target process. Set to 0 to indicate that no timeout should be used. The default is 120 seconds.

`/lineLevel (Alias: /ll)`

The profiler should record line-level timings as well as method-level timings.

`/methodLevel` (Alias: `/ml`)

The profiler should record only method-level timings (the default).

`/onlyWithSource` (Alias: `/ows`)

The profiler should only record values for methods which have source code files specified in their debugging data (pdb) files.

`/sampling` (Alias: `/sm`)

The profiler should use sampling to produce approximate results quickly.

`/includeSource:<includeSource>` (Alias: `/is`)

Whether or not the results should include the source code. Permitted values are *on* and *off*. The default is *on*.

`/inlining:<inlining>` (Alias: `/in`)

Default: *on*

Whether or not the profiler should allow .NET to inline functions. Turning this off will produce results for more methods, at the expense of a less accurate reflection of the processes performance. Permitted values are *on* and *off*. The default is *on*.

`/compensate:<compensate>` (Alias: `/comp`)

Whether or not the profiler should adjust results to account for its own overhead. Permitted values are *on* and *off*. The default is *on*.

`/simplify:<simplify>` (Alias: `/simp`)

Whether or not the profiler should simplify certain complicated stack traces to reduce resource requirements. Permitted values are *on* and *off*. The default is *on*.

`/avoidTrivial:<avoidTrivial>` (Alias: `/notriv`)

Whether or not the profiler should avoid extremely trivial functions to reduce resource requirements. These functions have a low hit count and a running time of only a few processor cycles. Permitted values are *on* and *off*. The default is *on*.

`/aspxPages:<aspxPages>` (Alias: `/aspx`)

Whether or not the profiler should profile the compiled contents of ASPX pages as well as the code that lies behind them. Turning this option on may considerably increase the amount of time the application spends in the JIT. Permitted values are *on* and *off*. The default is *off*.

`/threshold:<threshold>`

The threshold time in percent that a method must have used in order to be included in the report. Set to 0 to include all results. The default is 0.1.

`/csv:<csv>`

The name of a file to write a summary of the profiler results as CSV data to.

`/xml:<xml>`

The name of a file to write a summary of the profiler results as XML data to.

`/htmlreport:<htmlreport>` (Alias: `/h`)

The name of a file to write a summary of the profiler results as an HTML report.

`/calltree:<calltree>`

The name of a file to write a summary of the call tree profiler results as XML data to.

`/calltreehtml:<calltreehtml>` (Alias: `/cth`)

The name of a file to write a summary of the call tree profiler results as an HTML report.

`/data:<data>`

The name of a file to save the profiler results to. The contents of this file can be inspected using the ANTS Performance Profiler desktop application.

## Exit Codes

If an error occurs, the following exit codes may be displayed:

- 0 Success.
- 1 General error code.

3 Illegal argument duplication. Some arguments may not appear more than once in a command-line. If such arguments appear more than once this exit code will be returned.

8 Unsatisfied argument dependency or violated exclusion when user runs command line.

For example, /arg2 depends on /arg1 but you have specified /arg2 without specifying /arg1, or alternatively /arg2 cannot be used with /arg1 but you have tried to use them both.

32 Value out of range. Numeric value supplied for an argument that is outside the range of valid values for that argument.

33 Value overflow. The magnitude of a value supplied for an argument is too large and causes an overflow.

34 Invalid value. The value supplied for an argument is invalid.

35 No / invalid software license or trial period has expired.

64 General command-line usage error.

65 Data error. Some input data required by the tool is invalid or corrupt.

69 A resource or service required to run the tool is unavailable.

73 Failed to create report

74 IO error occurred. Generally returned if the program attempts to write to a file that already exists without the user having specified the /force option.

77 Action cannot be completed because the user does not have permission.

126 Execution failed because of an error.

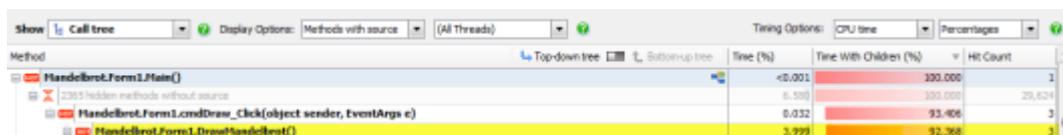
130 Execution stopped because Ctrl+Break.

## Examples of CSV and XML results files

This section exemplifies the format of the CSV and XML results files that can be saved when profiling from the command line.

The examples are for the `Mandelbrot.Form1.DrawMandelbrot()` method. For comparison, the following screenshots show the data for the same method when displayed in the GUI.

With **CPU time** selected (i.e. excluding time elapsed while a thread was blocked) :



Method	Time (%)	Time With Children (%)	Hit Count
Mandelbrot.Form1.Main()	<0.001	300.000	1
2385 hidden methods without source	6.500	300.000	29,621
Mandelbrot.Form1.cmdDraw_Click(object sender, EventArgs e)	0.032	93.406	3
Mandelbrot.Form1.DrawMandelbrot()	3.999	92.368	3

With **Wall-clock time** selected (i.e. showing total time elapsed, including blocking):

Method	Time (%)	Time With Children (%)	Hit Count
(Waiting for synchronization)	51.17%	51.17%	1
Mandelbrot.Form1.Main()	<0.001%	48.82%	1
Mandelbrot.Form1.OnDraw_Click(object sender, EventArgs e)	0.004%	11.30%	3
Mandelbrot.Form1.DrawMandelbrot()	0.994%	11.18%	4

In **XML** format:

```
<Method class="Mandelbrot.Form1" name="DrawMandelbrot" PID="3276" has-source="yes">
  <HitCount>4</HitCount>
  <CPU ticks="3943998009" millisecs="1274.7710" percent="31.1234" />
  <Wallclock ticks="3944321017" millisecs="1274.8746" percent="9.2591" />
  <WithSelf ticks="46780075" millisecs="18.0000" percent-cpu="0.3692" percent-wallclock="0.1098" />
</Method>
```

In **CSV** format (with headings row):

(Note that spaces have been added between each field to ensure that the lines in this example wrap.)

```
Method type, Class, Method, Hit count, CPU %, CPU milliseconds, CPU ticks, Wallclock %, Wallclock milliseconds, Wallclock ticks, CPU % time with self, Wallclock % time with self, Milliseconds with self, Ticks with self
, Mandelbrot.Form1, DrawMandelbrot(), 4, 31.1234417769152, 1274.77098086476, 3943998009, 9.25905031694744, 1274.87463378906, 3944321017, 0.369157625652905, 0.109813340848462, 18.0000198185444, 46780075
```

## Integrating ANTS Performance Profiler in a test procedure

---

Integrating ANTS Performance Profiler in your existing automated test framework ensures that you are alerted whenever a change is made that would adversely affect your application's performance.

To integrate ANTS Performance Profiler in automated tests, you perform three general steps:

1. Profile your application from the command line, saving the results to a CSV or XML file.
2. Read the results into the test harness.
3. Make assertions about the data you have read.

This topic describes these three steps in more detail.

### 1. Profiling your application from the command line

For automated tests, you do not need much detail in the results. We recommend that you either profile using method-level timing only, or that you profile in sampling mode.

To profile your application from the command line:

1. Choose whether you only need method-level timings, or whether you want to run the tests in sampling mode.
2. Choose whether you prefer results in CSV or XML format.

To assist your decision, see [Examples of CSV and XML results files \(/supportcenter/Content.aspx?p=ANTS Performance Profiler&c=ANTS\\_Performance\\_Profiler/help/6.2/app\\_commandline.htm&toc=ANTS\\_Performance\\_Profiler/help/6.2/toc1408011.htm#o14306\)](#).

3. Set the command line arguments to reflect your choices.

For example, to profile *SimpleApp.exe*, using method-level timings (the default) and saving results to a CSV file, run:

```
Profile.exe /e:"C:\testing\SimpleApp.exe" /csv:"C:\testing\results.csv" /data:"C:\testing\results.app6results"
```

To profile *SimpleApp.exe*, using sampling and saving results to a XML file, run:

```
Profile.exe /e:"C:\testing\SimpleApp.exe" /sm /xml:"C:\testing\results.xml" /data:"C:\testing\results.app6results"
```

Note that in these examples, the *.app6results* file is also saved. The *.app6results* file is not used for integrating with the test procedure, but saving it ensures that you can investigate any problems without needing to profile your application again.

For a full list of arguments you can use when profiling at the command line, see [Profiling from the command line \(API\)](#).

## 2. Read the results into the test harness

Write a simple command line application to read the CSV or XML data.

For information on how to read XML data in C# using the `XMLTextReader` class, see How to read XML from a file by using Visual C# (<http://support.microsoft.com/kb/307548>) (Microsoft.com)

## 3. Make assertions about the data you have read

Using your automated test framework, ensure that the performance data is within a reasonable bound for each method that you are interested in.

For example, if you know from experience that `exampleMethod()` normally takes about 0.02 seconds of CPU time, you assert that the time taken by `exampleMethod()` must not be longer than 0.02 seconds + 20% (0.024 seconds).

In the NUnit test framework:

```
Assert.Greater(0.024, double exampleMethodTime);
```

## Using the Visual Studio add-in

---

The ANTS Performance Profiler Visual Studio add-in allows you to:

- Launch ANTS Memory Profiler from your IDE.
- Switch straight to your source code from ANTS Performance Profiler.

### Launching ANTS Performance Profiler from Visual Studio

Installing the add-in adds a new **ANTS** menu in Visual Studio. If you also have ANTS Memory Profiler installed, both profilers will be available under this menu.

Build your solution in Visual Studio and then select **Profile Performance** to profile the build.

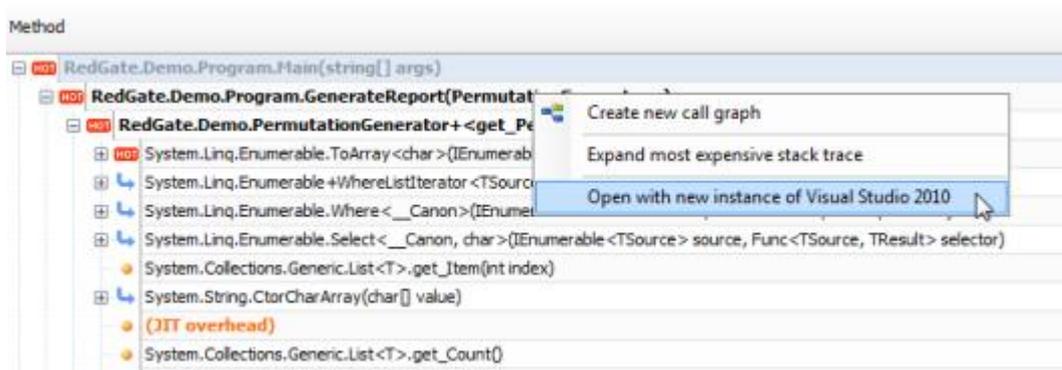


### Switching to your source code from ANTS Performance Profiler

So that ANTS Performance Profiler can identify classes with source code, you must ensure that the *.pdb* file is in the same directory as the application. See [Resolving .PDB problems](#) for more details.

You can switch to source code from the call tree and the methods grid. In both, classes with source code are shown in bold. You can use the **Find** bar to search for your class's namespace.

Right-click a class with source code to show the context menu.



To open only the source code associated with that class, select **Open with new instance of Visual Studio 20xx**.

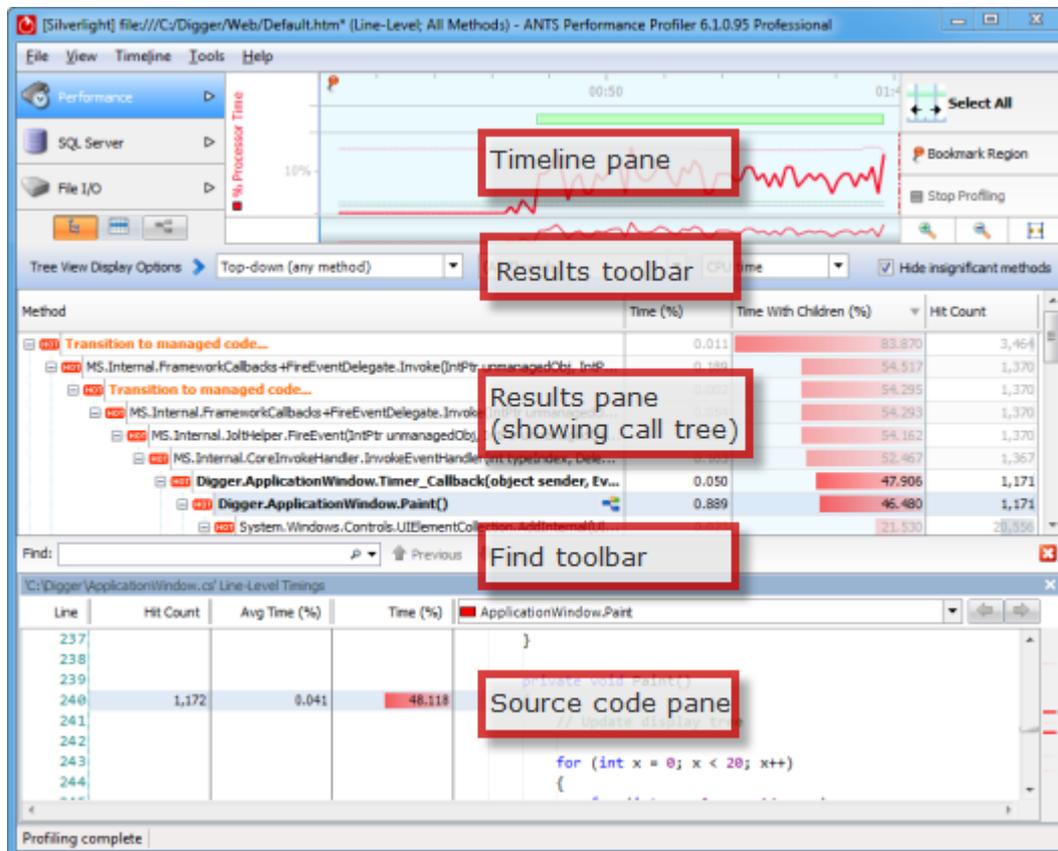
It is often more useful to open the source code inside its solution. To do this, open the solution in Visual Studio. Return to ANTS Performance Profiler and open the context menu. On the context menu, select **Open with (Solution Name) - Microsoft Visual Studio (Visual Studio 20xx)**.

### Troubleshooting

- If the path to the source code in the *.pdb* file is invalid, the class is still shown in bold, but the context menu does not display the Visual Studio options. Recompile the application on the computer you are using to profile it.
- If the solution was opened with elevated privileges (Visual Studio is running as administrator), the option for opening the source code inside the solution might not be shown. Restart Visual Studio under the same credentials as ANTS Performance Profiler.
- The Visual Studio add-in is a separate program from ANTS Performance Profiler, and is installed by default as part of Red Gate's .NET Developer Bundle. If you purchased ANTS Performance Profiler as a standalone product, you might not have the add-in. In that case, download the free trial of the .NET Developer Bundle (<http://www.red-gate.com/products/dotnet-development/dotnet-developer-bundle/>) from the Red Gate website and install **ANTS Profiler Visual Studio Add-in 1.0**. You do not need a new license for the add-in, and the add-in will not expire when the bundle's trial period ends.

## The ANTS Performance Profiler user interface

Once you have run a profiling session and displayed some profiling results you can start analyzing the results in the results pane using the three main display types: call tree, methods grid, and call graph.

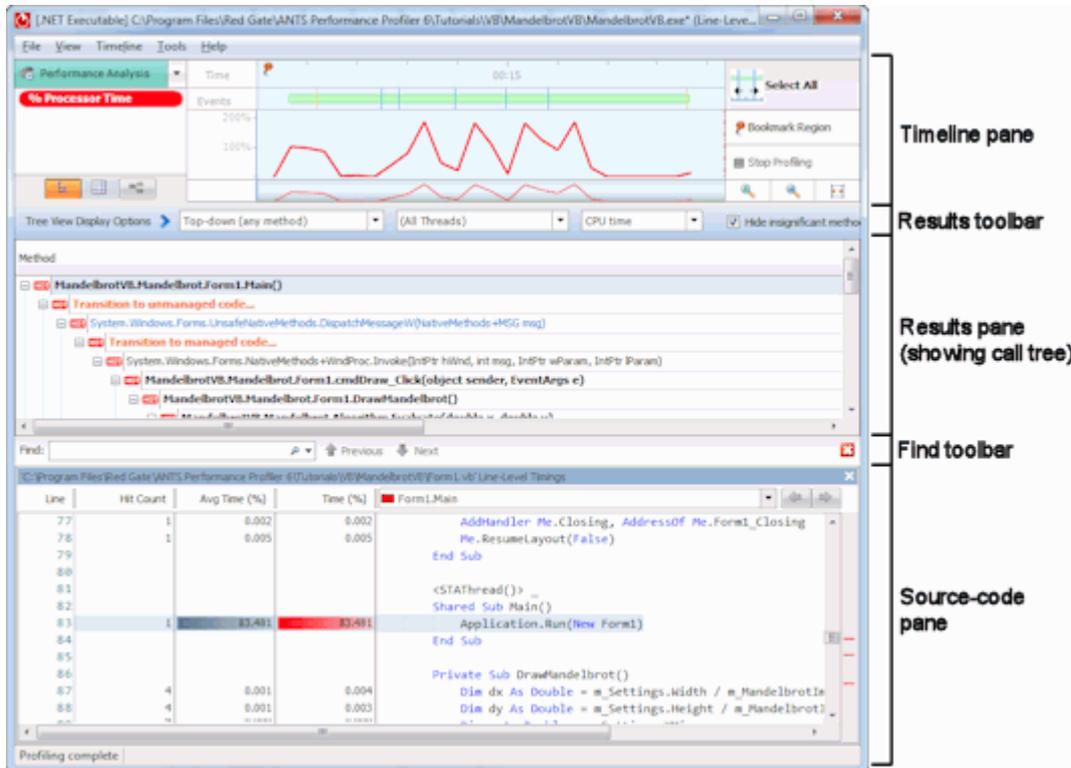


Use the buttons on the timeline pane to switch between display types:

-  Call tree: shows stack traces that were executed by your application during the time period you have selected.
-  Methods grid: lists each method that was executed by your application during the time period you have selected.
-  Call graph: shows the calling relationships between methods executed by your application, for the time period you have selected. (The call-graph button is disabled until you have created a new call graph.)

## Working with profiling results

Once you have run a profiling session and displayed some profiling results you can start analyzing the results in the results pane using the three main display types: call tree, methods grid, and call graph.



The screenshot displays the ANTS Performance Profiler interface. The top section is the **Timeline pane**, showing a graph of processor time usage over a 15-second period. Below this is the **Results toolbar**, which includes buttons for 'Select All', 'Bookmark Region', and 'Stop Profiling'. The main **Results pane (showing call tree)** displays a hierarchical view of the execution stack, starting with 'MandelbrotVB.MandelbrotForm1.Main()'. Below the call tree is the **Find toolbar** with 'Previous' and 'Next' buttons. The bottom section is the **Source-code pane**, which shows the source code for the selected method, including the 'Main' method and the 'DrawMandelbrot' method. A table of line-level timings is visible above the source code.

Line	Hit Count	Avg Time (%)	Time (%)	Form1.Main
77	1	0.002	0.002	AddHandler Me.Closing, AddressOf Me.Form1_Closing
78	1	0.005	0.005	Me.ResumeLayout(False)
79				End Sub
80				
81				<STAThread()> _
82				Shared Sub Main()
83	1	83.481	83.481	Application.Run(New Form1)
84				End Sub
85				
86				Private Sub DrawMandelbrot()
87	4	0.001	0.004	Dim dx As Double = m_Settings.Width / m_MandelbrotI
88	4	0.001	0.003	Dim dy As Double = m_Settings.Height / m_MandelbrotI

Use the buttons on the timeline pane to switch between display types:

-  **Call tree:** shows stack traces that were executed by your application during the time period you have selected.
-  **Methods grid:** lists each method that was executed by your application during the time period you have selected.
-  **Call graph:** shows the calling relationships between methods executed by your application, for the time period you have selected. (The call-graph button is disabled until you have created a new call graph.)

## Working with the timeline

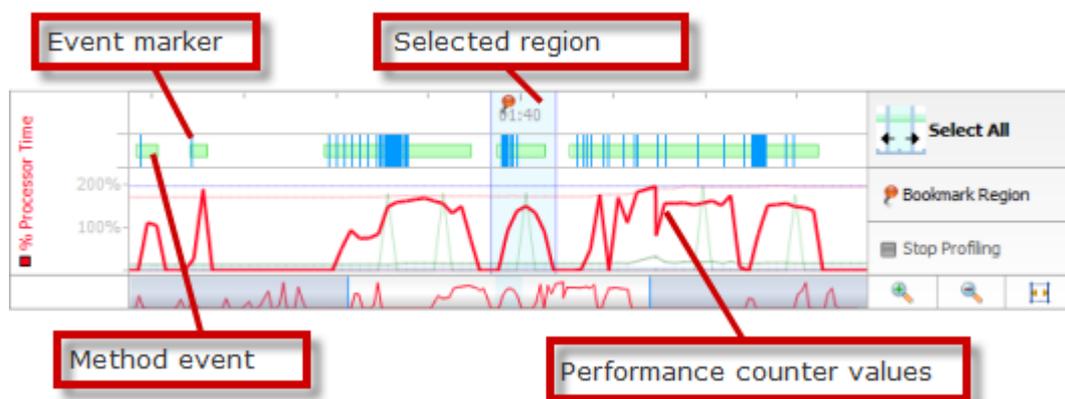
---

The timeline is visible throughout a profiling session, and provides a frequently updated display of performance-counter values and instances of events related to the application you are profiling. You can use this overview of application activity to isolate performance-profiling results for specific time periods.

The timeline enables you to select a region (corresponding to a time period during execution of your application) for which you wish to display profiling results. You can select and reselect any region as often as you need to, both during profiling and after you have stopped profiling and closed your application. You can also create bookmarks for selected regions, enabling you to define multiple regions and switch between them to look at data for different periods during a profiling session.

The main section of the timeline shows the values for a selection of Windows performance counters. You can choose which performance counters to display before you start profiling your application. See [Setting up performance counters](#) for more information.

The event bar on the timeline shows event markers. These indicate when certain types of event occur within your application, for example, button clicks, window activations, and exceptions. When you move the mouse pointer over an event marker a tooltip provides more information about the event. See [Working with event markers and method events](#) for more information.



### Working with regions on the timeline

You can select, clear, and bookmark regions on the timeline. Whenever you select a region, profiling results are displayed that relate to the selected period only.

#### Selecting a region

To select a region, click and drag the mouse pointer  across the timeline. The results pane (beneath the timeline) updates to show profiling results for the selected region.

## Resetting a region

To reset the currently selected region to cover the whole timeline, click **Select All**. The results pane updates to show profiling results for the entire time period for which your application was running.

## Bookmarking a region

You can create a bookmark on the timeline, for a selected region. This is useful if there are several periods for which you want to view or compare profiling results: you can easily switch between bookmarked regions to redisplay profiling results.

To bookmark a selected region, click  within the selected region. This region is now bookmarked (this is indicated by a highlighted bar at the top of the timeline). To select this region again later, click within the highlighted bar.

You can name a bookmark to make it easier to identify. To name a bookmark, click the highlighted bar for the bookmark and click . The name you type will be shown on the bookmark's tooltip.

To delete a bookmark, click the highlighted bar for the bookmark and click .



## Adjusting the time scale

You can change the time scale to view performance-counter data in more or less detail by rotating the mouse wheel, or by using the zoom-control buttons (zoom in , zoom out , and zoom to fit ). You can also use the following keyboard shortcuts: CTRL+PLUS to zoom in; CTRL+MINUS to zoom out.

To pan the main timeline, move the mouse pointer over the highlighted area in the overview timeline (, and drag to the left or right.

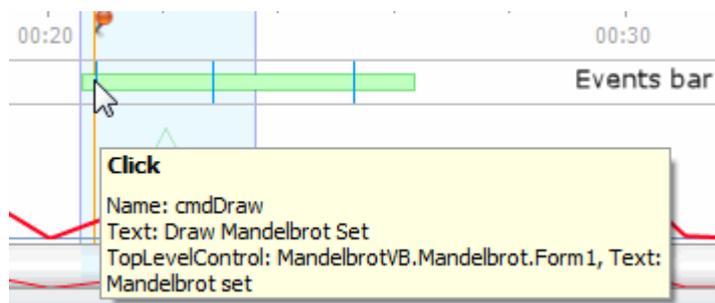
## Working with performance counters

The performance counters available for the current profiling session are listed to the left of the timeline. You can choose which performance counters to display when you set up a new profiling session (see Setting up performance counters).

To highlight a particular performance counter on the timeline, click its description in the **Performance Counters** list. Values for the selected performance counter are shown on a tooltip when you move your mouse pointer over the main timeline.

## Working with event markers and method events

The events bar (directly above the main timeline) indicates when certain events occurred within the application you are profiling. To display more information about the event, move your mouse pointer over an event marker.



Events are displayed using colored markers:

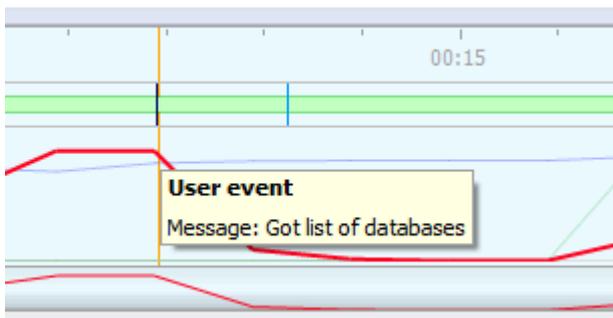
Event type	Color
Exception	Red
Click	Blue
Window activated and window closed	Yellow
All other events	Dark Blue

If you click on a method in the call tree or on the methods grid (and when you display the call graph for a method), the times when that method executed are shown as light green areas in the events bar. These areas are called 'method events'. If the method called unmanaged code, the period when the unmanaged code ran is shown as dark green.

## Adding custom event markers

It is possible to add your own event markers to the timeline. For example, you might want to know when a certain item is loaded into memory or when an ASP.NET page is loaded or unloaded. To add a custom marker:

1. Copy  
`%ProgramFiles%/Red Gate/ANTS Performance Profiler 6/RedGate.Profiler.UserEvents.dll`  
to your application's *bin* folder. (Create the *bin* folder using Windows Explorer if it does not already exist.)
2. Reference the DLL from your application.
3. Add the following line to your application wherever you want to see an event marker on the profiler's timeline. (Replace 'The message' with a short description of the event.)  
`RedGate.Profiler.UserEvents.ProfilerEvent.SignalEvent("The message");`
4. When you profile your application, the event is shown in the ANTS Performance Profiler timeline with a black event marker. Move your mouse over the event marker to display the message.

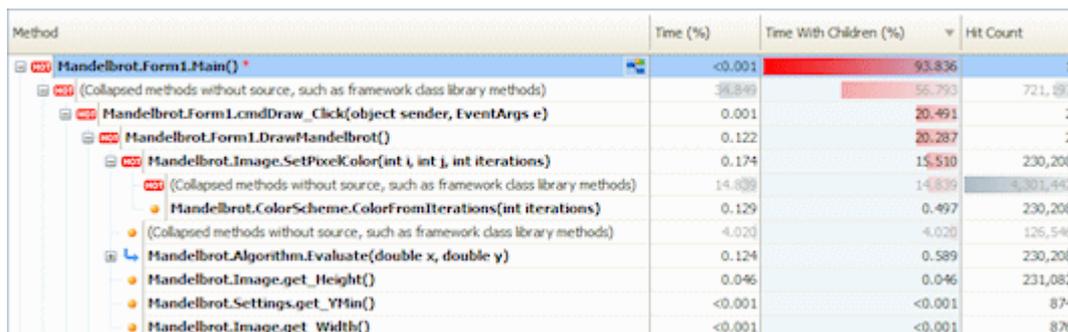


Note that you cannot use custom event markers when profiling Silverlight applications.

## Working with the call tree

The call tree shows the stack traces that were executed by your application during the time period you have selected. By default, stack traces are displayed top-down (calling method above called method). The "hottest" stack trace (the one that took the most time to run) is displayed at the top of the call tree, and is automatically expanded. If a method was called in several contexts, it is displayed once for each context in the call tree.

See Tips on using the call tree for more information on how to use the call tree effectively.



Method	Time (%)	Time With Children (%)	Hit Count
Mandelbrot.Form1.Main() *	<0.001	93.836	1
(Collapsed methods without source, such as framework class library methods)	56.899	56.793	721,997
Mandelbrot.Form1.cmdDraw_Click(object sender, EventArgs e)	0.001	20.491	2
Mandelbrot.Form1.DrawMandelbrot()	0.122	20.287	2
Mandelbrot.Image.SetPixelColor(int i, int j, int iterations)	0.174	15.510	230,208
(Collapsed methods without source, such as framework class library methods)	14.839	14.839	4,301,440
Mandelbrot.ColorScheme.ColorFromIterations(int iterations)	0.129	0.497	230,208
(Collapsed methods without source, such as framework class library methods)	4.020	4.020	126,546
Mandelbrot.Algorithm.Evaluate(double x, double y)	0.124	0.589	230,208
Mandelbrot.Image.get_Height()	0.046	0.046	231,082
Mandelbrot.Settings.get_YMin()	<0.001	<0.001	874
Mandelbrot.Image.get_Width()	<0.001	<0.001	876

The following data is shown for each method within the stack trace, for the selected time period:

- **Time:** the total execution time for the method within this stack trace.
- **Time With Children:** the total execution time for the method and all its children within this stack trace.
- **Hit Count:** the number of times the method was called within this stack trace.

When time is shown as a percentage, the **Time (%)** for each method shows the proportion of the total execution time that the method contributed during the selected period. The total percentage for all methods can sum to over 100 on machines using multiple CPU cores.

Each method is shown with one of the following icons:

- Root method or leaf method. Root methods are not called by any other method; leaf methods do not call any other method.
- ↳ Indicates call flow when the call-tree direction is top-down (calling method above called method).
- ↑ Indicates call flow when the call-tree direction is bottom-up (called method above calling method).
- HOT Method is part of the hottest (longest running) stack trace. Used instead of the root/leaf or call-flow icons.
- \* A method name followed by an orange asterisk: Method is optimizable.

Methods listed in **bold** have source code available. To display the method's source code, click any bold method. Line-level timings are also available in the source-code pane if you use one of the *Line-level ...* profiling modes.

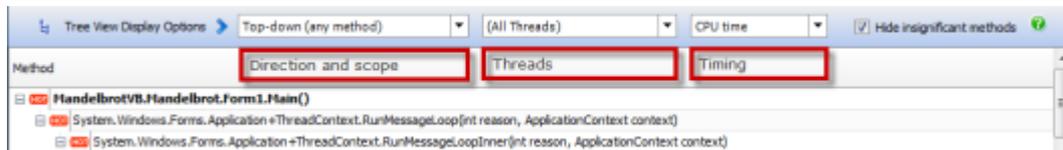
You may also see the following items in the call tree. These are shown in bold orange text, and represent time spent in your application that is in addition to time spent executing specific methods:

- **Waiting for synchronization:** A thread was waiting. For example, if you have called the `Monitor.Wait` method, the thread will wait for synchronization until the lock is reacquired. Another cause is that the finalizer thread spends most of its time waiting. Expand the call graph to see what caused the wait. Items that are *Waiting for synchronization* only contribute to timings in the call tree when the **Timing** display option is set to *Wall-clock time*. To exclude time due to *Waiting for synchronization* items, select *CPU time* from the **Timing** display option.
- **Thread blocked:** The executing thread was blocked. For example, the thread may have been sleeping, or waiting for access to a shared resource. *Thread blocked* items only contribute to timings in the call tree when the **Timing** display option is set to *Wall-clock time*. To exclude time due to *Thread blocked* items, select *CPU time* from the **Timing** display option. The call-tree display options are described below.
- **Waiting for I/O to complete:** The executing thread was blocked waiting for file I/O. *Waiting for I/O to complete* items only contribute to timings in the call tree when the **Timing** display option is set to *Wall-clock time*. To exclude time due to *Waiting for I/O to complete* items, select *CPU time* from the **Timing** display option. The call-tree display options are described below.
- **Transition to unmanaged code:** A transition from managed code to unmanaged code occurred at this point in the stack trace. In general, line-level and method-level timings are not available for the unmanaged code. However, for unmanaged methods that are declared with `extern` within managed code, method-level timings are available.
- **Transition to managed code:** A transition from unmanaged code to managed code occurred at this point in the stack trace.
- **JIT overhead:** JIT compilation occurred at this point in the stack trace during execution of your application. The method that needed to perform the compilation is shown as the parent of a *JIT overhead* item.
- **Profiler overhead:** Additional overhead introduced by ANTS Performance Profiler. This is unlikely to be seen when the option to adjust timings to compensate for overhead added by the profiler is enabled.
- **Assembly load or unload:** A .NET assembly was loaded or unloaded.
- **Module load or unload:** A .NET module was loaded or unloaded.

To create a new call graph based on a particular method, select the method in the call tree, and click the new call graph button  in the **Method** column.

## Changing the call-tree display options

You can change the way data is displayed in the call tree, using the display options on the results toolbar:



- **Direction and scope:** controls whether the call tree is displayed top-down (calling methods above called methods) or bottom-up (called methods above calling methods), and also whether any method, or only methods with source, are shown. If you choose an option that shows any method, the call tree will include details for .NET Framework class-library methods.
- **Threads:** filters the display of stack traces by thread.
- **Timing:** controls the way in which method timings are calculated. You can choose from *Wall-clock time* which includes blocking such as waiting for I/O, or *CPU time* which excludes blocking.
- **Hide insignificant methods:** select this check box to hide methods that contribute less than 1% of the total execution time for the currently selected time period.

You can also:

- Change the time unit. On the **View** menu, click **Percentages**, **Ticks**, **Milliseconds**, or **Seconds**.
- Reorder the call tree. To change the stack-trace order, click the **Time With Children (%)** column heading.

## Tips on using the call tree

To locate methods that may be good candidates for optimization:

1. Order the call tree with the slowest stack traces at the top (top-down). If necessary, click the **Time With Children** column heading to change the stack-trace order.
2. Starting with the slowest stack traces, look for method pairs where subsequent values for **Time With Children** reduce substantially as you move down the stack trace. Methods with higher values in such pairs may be good candidates for optimization.

ANTS Performance Profiler can optionally suggest methods that may be good candidates for optimization. To show suggested methods, on the **Tools** menu, click **Suggest methods to optimize**. Suggested method names are marked with an asterisk (\*).

In general, the better you understand the structure and meaning of your code, the more easily you will be able to interpret the data collected by the profiler.

To reduce the number of methods shown, you can do any of the following:

- Choose a "(methods with source)" option from the **Direction and scope** list in the display options.

iterations(int iterations)	0.027	0.027	115,104
double y)	0.017	0.131	115,104
oubles(double x, double y)	0.113	0.113	115,104

Display: **Top-down (methods with source)** (All Threads) CPU time  **Hide insignificant methods**

Direction and scope      Threads      Timing      Hide insignificant methods

- Select the **Hide insignificant methods** check box in the display options.
- Select a shorter region on the timeline.

To find a particular method:

1. On the **Tools** menu, click **Find**.  
The **Find** bar is displayed beneath the call tree.
2. Type all or part of the method name you are looking for, and press ENTER.  
The first matching row in the call tree is highlighted.

Click **↑ Previous** or **↓ Next** to move between matching method names.

## Working with the methods grid

The methods grid lists each method that was called by your application during the time period you have selected. Even if a given method is called in several contexts, it is shown only once in the methods grid, with aggregated data that accounts for all contexts. You can order the data by any column by clicking the column heading. Data is ordered by **Time With Children** by default.

Namespace	Method Name	Time (%)	Time With Children (%)	Hit Count	Source File
Mandelbrot	<b>Form1.Main()</b>	0.000	94.495	1	Form1.cs
Mandelbrot	<b>Form1.ctor()</b>	0.003	40.333	1	Form1.cs
Mandelbrot	<b>Form1.cmdDraw_Click(object sender, EventArgs e)</b>	0.001	21.003	2	Form1.cs
Mandelbrot	<b>Form1.DrawMandelbrot()</b>	0.082	20.746	2	Form1.cs
Mandelbrot	<b>Image.SetPixelColor(int i, int j, int iterations)</b>	0.168	16.267	230,208	Image.cs
Mandelbrot	<b>Form1.InitializeComponent()</b>	0.007	7.750	1	Form1.cs
Mandelbrot	<b>Form1.Dispose(bool disposing)</b>	0.000	1.520	1	Form1.cs
Mandelbrot	<b>Algorithm.EvaluateUsingDoubles(double x, double y)</b>	0.393	0.393	115,104	Algorithm.cs
Mandelbrot	<b>ColorScheme.ColorFromIterations(int iterations)</b>	0.107	0.308	230,208	ColorScheme.cs
Mandelbrot	<b>Image.ctor(int width, int height)</b>	0.001	0.116	1	Image.cs
Mandelbrot	<b>Algorithm.Evaluate(double x, double y)</b>	0.117	0.113	230,208	Algorithm.cs
Mandelbrot	<b>Image.get_Height()</b>	0.032	0.032	231,082	Image.cs

The following data is shown for each method, for the time period you have selected:

- **Time:** the total execution time for the method (in all contexts).
- **Time With Children:** the total execution time for the method and all its children.
- **Hit Count:** the number of times the method was called.

Methods listed in **bold** have source code available. To display the method's source code, click any bold method. Line-level timings are also available in the source-code pane if you used the *Line-level and method-level timings; all methods* or *Line-level and method-level timings; only methods with source* profiling mode.

### Changing the methods-grid display options

You can change the way data is displayed in the methods grid, using the display options on the results toolbar.

Namespace	Method Name	Scope (%)	Threads	With Children (%)	Hit Count	Timing mode	Timing units
	133 hidden methods without source		99.500	99.877	7,977,120		
RedGate.Demo	<b>Program.GenerateReport(PermutationGener...</b>	0.027		49.782	1		
RedGate.Demo	<b>Program.Main(string[] args)</b>	0.000		49.782	1		

- **Scope:** controls whether any method, or only methods with source, are shown. If you choose to display any method, the methods grid will include details for .NET Framework class-library methods.
- **Threads:** filters the display of stack traces by thread.
- **Timing:** controls the way in which method timings are calculated. You can choose from **Wall-clock time** which includes blocking such as waiting for I/O, or **CPU time** which excludes blocking.
- **Hide insignificant methods:** select this check box to hide methods that contribute less than 1% of the total execution time for the currently selected time period.

### Tips on using the methods grid

To find particular methods:

1. On the **Tools** menu, click **Find**.

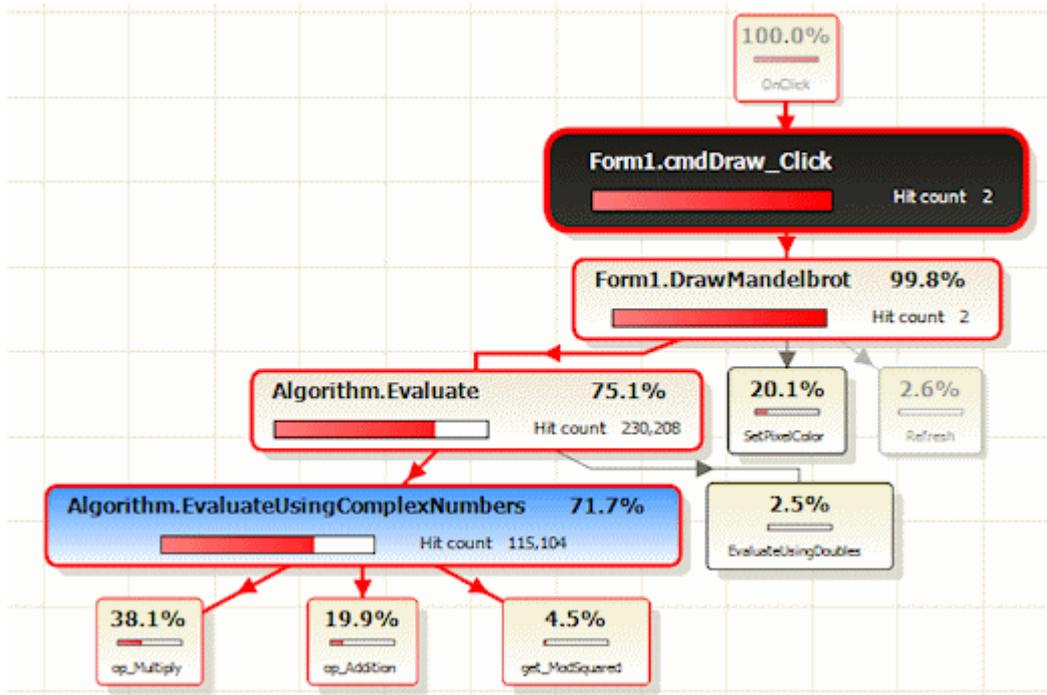
The **Find** bar is displayed beneath the methods grid.

2. Type all or part of the method name you are looking for.

As you type, the methods are filtered to display only those that match your text.

## Working with the call graph

The call graph shows the calling relationships between methods during the execution of your application, and is focused on a method of your choice (the *base* method; shown in black in the example below). If a given method is called in several contexts, it is shown once for each context in the call graph. The base method is shown only once in the call graph, unless it is called recursively.



Selecting a base method makes it easy for you to visualize all the callers and callees for that method.

The percentage value shown in each method is calculated with respect to the base method as follows:

- For a method called by the base method, this is the percentage of the base method's execution time that the method accounts for, relative to the base method's total execution time.
- For a method that calls the base method, this is the percentage of the base method's total execution time that is due to the calling method.

Calculations are always made with respect to the selected region on the timeline, or the whole profiling period if you have not selected a region.

## Creating a new call graph

Every instance of a call graph is based on a particular method, so you must first select a method in the call tree, methods grid, or source code, then click the create new call graph button :

	0.007	9.358	1
ibrary methods)	1.591	45.253	1,718
<b>EventArgs e)</b>	0.001	13,302	1
	0.063	13,101	1
<b>int iterations)</b>	Create a new call graph for this method.		9.711 115,104
work class library methods)	1.280	2.593	810
	<0.001	7.405	1

Alternatively, right-click the method and select **Create new call graph** on the short-cut menu.

The call graph is displayed in the results pane.

## Changing the call-graph display options

You can change the way data is displayed in the call graph, using the display options on the results toolbar:



- **Weighting:** controls the way that methods are drawn on the call graph. *Equal weighting to all methods* is useful when you need to see how methods without source code (for example, .NET Framework library methods) affect the execution times in your application. *Emphasize methods with source* draws the call graph with much smaller boxes for those methods that do not have source code available. This allows you to concentrate on the timings for those methods for which you have the source code.
- **Timing:** controls the way that method timings are calculated. You can choose from **Wall-clock time** which includes blocking such as waiting for I/O, or **CPU time** which excludes blocking.
- **Hide insignificant methods:** select this check box to hide methods that contribute less than 1% of the total execution time (for the currently selected time period).

You can also change the time unit. On the **View** menu, click **Percentages**, **Ticks**, **Milliseconds**, or **Seconds**.

## Navigating the call graph

You can resize the call graph by rotating the mouse wheel, or by using the zoom controls to the left of the call graph. You can pan the call graph by clicking and dragging on a blank part of the graph.

To expand a method on the call graph (that is, to show the method's immediate children or parents), click the method.

To expand the most expensive path from a particular method, hold down the CTRL key and click the method. Alternatively, right-click and select **Expand most expensive stack trace**.

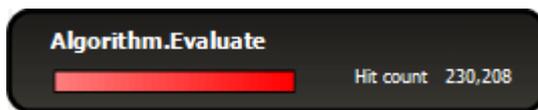
To expand the most expensive path for all children from a particular method, hold down the SHIFT key and click the method. Alternatively, right-click and select **Expand most expensive stack trace for all callees**.

To expand the most expensive path for all parents of a particular method, hold down the SHIFT key and click the method. Alternatively, right-click and select **Expand most expensive stack trace through all callers**.

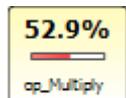
To collapse a method on the call graph, double-click the method.

## More about call graphs

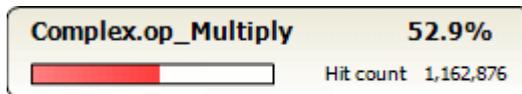
Methods are drawn in several different styles in a call graph:



Base method (the method you chose when creating the call graph). Execution-time percentages are calculated with respect to this method.



(Minimized)



(Maximized)

Method with source code.



(Minimized)



(Maximized)

Method without source code. This style is used when all methods have equal weighting.



Method without source code. This style is used when methods with source code are emphasized.



(Minimized)



(Maximized)

Selected method. When a method is selected, *all* methods in that stack trace are also outlined in red.



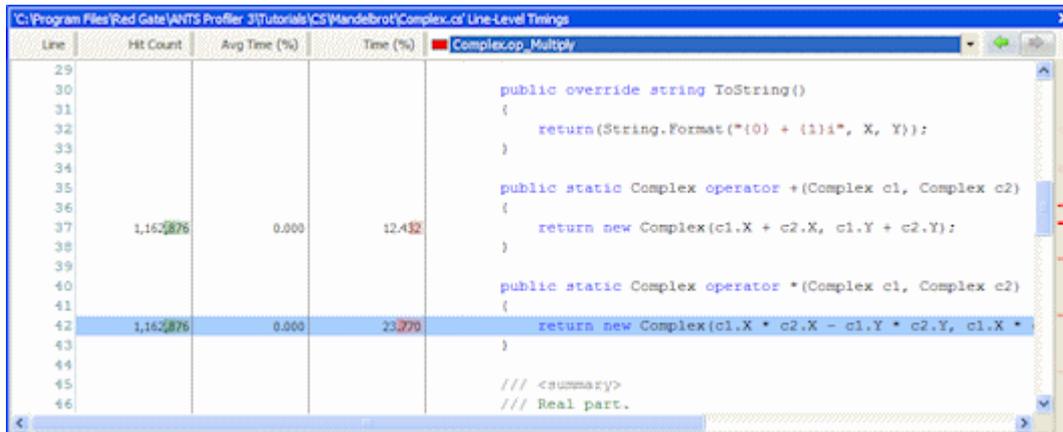
Recursive method. The  symbol is added to any method that is called recursively within your application.

Call graphs always include methods for which no source code is available (for example, methods from the .NET Framework class library) and methods for all threads running in your application during profiling.

It is not possible to change the time period covered by an existing call graph. To create a call graph for a different time period, return to the call-tree or methods-grid display, reselect the required period on the timeline, and create a new call graph.

## Working with source code

When you select a method in the results pane and source code is available for that method, the code is displayed in the source-code pane with the first line of the method body highlighted. You can also profile and navigate code decompiled using Red Gate's .NET Reflector VSPro (<http://www.reflector.net/>) in the same way as the original source code.



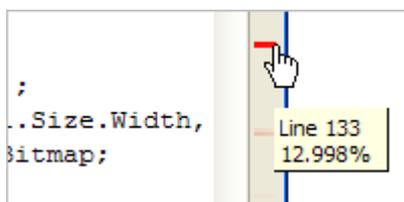
Line	Hit Count	Avg Time (%)	Time (%)
29			
30			
31			
32			
33			
34			
35			
36			
37	1,162,876	0.000	12.432
38			
39			
40			
41			
42	1,162,876	0.000	23.270
43			
44			
45			
46			

If you used *Line-level and method-level timings*; *all methods* or *Line-level and method-level timings*; *only methods with source* profiling mode, line-level timings are shown for each line of code (as well as average time and hit count). Note that line-level timings are unavailable for Silverlight applications.

### Navigating through source code

You can navigate through the source code for a particular file in several ways:

- To jump directly to lines of code that accounted for the most execution time, use the heat map alongside the vertical scroll bar. Colored bars indicate the location of the slowest lines of code in the source-code file:



Click a bar to jump to the relevant line. The heat map is not available if you did not collect line-level timings.

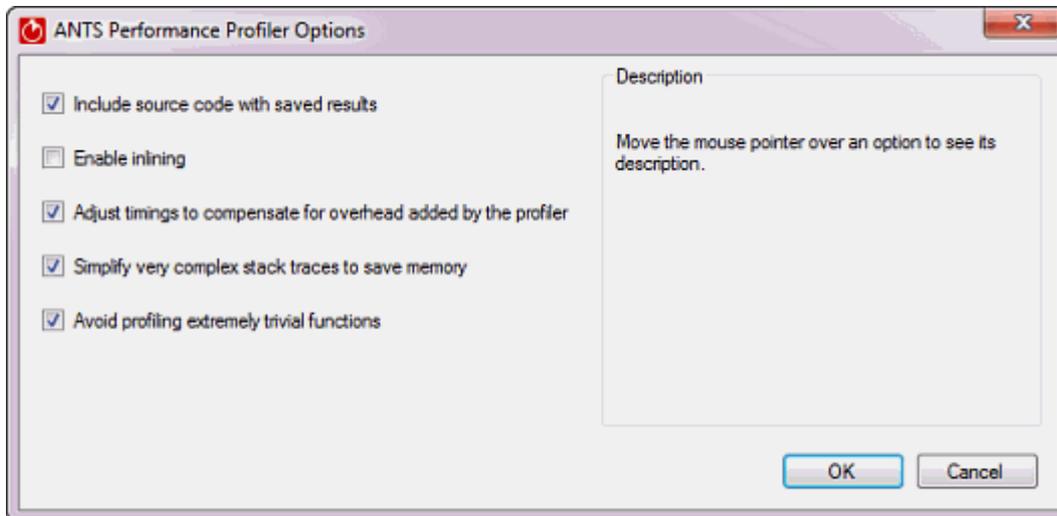
- To jump directly to a particular method, select the method from the list directly above the source code. A colored bar against each method indicates the relative time spent within the method.
- To jump between methods from within the source code, click the method call. This click-through navigation works for most method calls where the called method has source code available.
- Use the forward  and back  history buttons.

You can also create a new call graph from within the source-code pane: right-click the method that you want to base the call graph on, and select **Draw Call Graph** > *<method name>*.

## ANTS Performance Profiler options

---

ANTS Performance Profiler includes a number of options that are applied to all profiling sessions. To access these options, on the **Tools** menu, click **Options**.



Unless you have a particular need to adjust the options, leave them at their default settings. Changing the default setting for certain options may cause problems during profiling.

Include source code with saved results

Enable inlining

Adjust timings to compensate for overhead added by the profiler

Simplify very complex stack traces to save memory

Avoid profiling extremely trivial functions

### **Include source code with saved results**

Includes the contents of source files when you save profiling results. This means that you can review line-level performance data in saved results, without having to restore your source files to their original state.

You may want to clear this option if, for example, you need to distribute performance profiling results for an application that has confidential source code.

By default, this option is selected.

## **Enable inlining**

Enables inlining of methods by the .NET JIT compiler, for the process being profiled.

If you are profiling the release build of an application, selecting this option will produce a profile that is closer to the "real-world" performance. However, the accuracy of the results will be reduced. In particular, line-level timings will be distorted, hit counts will not be recorded for inlined methods, and time spent in inlined methods will be reported as part of the calling method.

By default, this option is not selected.

## **Adjust timings to compensate for overhead added by the profiler**

Adjusts timings by estimating the influence the profiler has had on the process being profiled, and subtracts this from the profiling results. This estimate is most accurate when you use a profiling mode that does not collect line-level timings.

The design of modern processors means that this estimate may not always be accurate, especially for short function calls.

By default, this option is selected.

## **Simplify very complex stack traces to save memory**

Summarizes complex stack traces in profiling results. This conserves resources on the machine you are using for profiling. The stack traces that are summarized are unlikely to be important to your profiling results. However, if you wish to see these summarized results, you can clear this option.

Clearing this option can significantly increase the memory required by the profiler. Depending on the application you are profiling, the profiler may become unstable if you clear this option.

By default, this option is selected.

## **Avoid profiling extremely trivial functions**

Prevents profiling of methods that have a running time measured in tens of nanoseconds, and which contribute to less than one-billionth of the run time in total. Typically, these methods do not produce very relevant performance data. Ignoring these methods reduces the amount of memory required to store and process profiling results.

By default, this option is selected.

## Troubleshooting application crashes

---

If the application you are profiling stops responding, fails to load, or closes unexpectedly, the profiling process may have caused the profiled application to crash. If this happens, you may not receive profiling results for your session, or you may receive incomplete results captured before the crash.

This topic describes ways to troubleshoot crashes during profiling.

If your profiling session returns no results, or incomplete results, but does not hang or crash, see [Troubleshooting missing results](#).

### Troubleshooting error messages

The application may crash with the following error messages:

- Operation could destabilize the runtime
- StackOverflow Exception  
(This exception may be recorded only in the log file.)

#### Operation could destabilize the runtime

If your application crashes with the error "Operation could destabilize the runtime" during line-level profiling, then the assembly or method you are profiling may have been marked as SecurityTransparent (or, in .NET 3.5 and earlier, marked as partially trusted).

To prevent crashes of this type:

- In **.NET 4**: Set the application's code to **SecurityCritical** or **SecuritySafeCritical** mode. (This requires the code to have full trust.)

For instructions on setting these attributes, see [SecurityCriticalAttribute Class](http://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute.aspx) (<http://msdn.microsoft.com/en-us/library/system.security.securitycriticalattribute.aspx>) or [SecuritySafeCriticalAttribute Class](http://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute.aspx) (<http://msdn.microsoft.com/en-us/library/system.security.securitysafecriticalattribute.aspx>).

- In **.NET 3.5** and earlier: Set the application's code trust level to **FullTrust**.

For details, see [Named Permission Sets](http://msdn.microsoft.com/en-us/library/4652tyx7%28v=VS.85%29.aspx) (<http://msdn.microsoft.com/en-us/library/4652tyx7%28v=VS.85%29.aspx>).

#### StackOverflow Exception

During line-level profiling, if the application you are profiling is very large or the profiling session lasts a long time, a stack overflow can occur. If this happens, the application will crash and may record a "StackOverflow Exception" error in the log. (For instructions on finding the ANTS Performance Profiler logs, see [Log files](http://www.red-gate.com/supportcenter/content/ANTS_Performance_Profiler/knowledgebase/ANTS_log_files) ([http://www.red-gate.com/supportcenter/content/ANTS\\_Performance\\_Profiler/knowledgebase/ANTS\\_log\\_files](http://www.red-gate.com/supportcenter/content/ANTS_Performance_Profiler/knowledgebase/ANTS_log_files)).

To reduce the risk of an overflow, open the **Tools** menu, click **Options**, and select **Simplify very complex stack traces to save memory**.

If this does not prevent the crash, try profiling in sampling mode: on the **Application Settings dialog box**, set the **Profiling mode** to **Sample method-level timings (fastest, least detail, no hit counts)**.

## Troubleshooting all crashes

If the application crashed but did not return either of the above error messages, try the following approaches. They are listed in order of how likely they are to resolve common causes of crashes.

### Force your application to use .NET 4

Many crashes can be avoided by rebuilding the application as a .NET 4 executable, and profiling it using the **Attach to a .NET 4 process** option in Application Settings.

For instructions on reconfiguring your application for use with this profiling method, see Forcing your application to use .NET 4 ([http://www.red-gate.com/SupportCenter/Content/ANTS\\_Performance\\_Profiler/knowledgebase/net4](http://www.red-gate.com/SupportCenter/Content/ANTS_Performance_Profiler/knowledgebase/net4)).

The **Attach to a .NET 4 process** approach profiles your application when it is already running, rather than launching a new instance of the application. Because attaching to a running process does not change the way the application's code executes, it is less likely to cause a crash.

### Run ANTS Performance Profiler from the Visual Studio menu

If the crash occurs in a profiling session launched directly from ANTS Performance Profiler, try profiling using the Visual Studio ANTS Performance Profiler add-in instead.

For instructions see Using the Visual Studio add-in.

### Delete corrupt third-party PDBs

The following third-party assemblies sometimes contain corrupt *.pdb* files, which can cause crashes during profiling:

- Microsoft.Practices library
- AjaxControlToolkit

For instructions on finding and deleting corrupt *.pdb* files, see Troubleshooting PDB problems.

## Troubleshooting web application crashes

If the application you are profiling is a web application, try the following two steps.

## Profile using another web server application

Some applications that crash under profiling can be profiled successfully if run on another type of server application:

- If you are profiling in IIS, switch to the ASP.NET built-in web development server.
- If you are profiling in the ASP.NET web development server, switch to IIS.

Note that changing environments will work only if the server application you select supports all your application's features:

- Applications using integrated pipeline mode, impersonation, or HTTPS will not run on the built-in web-development server.
- Applications that cannot operate in a restricted security context may not run in IIS.

Profiling results obtained in from applications running on the web development server may also differ from performance in a production environment.

## IIS: ensure you are profiling on the correct port

- If you are using **IIS 6** or **IIS 7**, select **Unused port** and choose a port that is not used by IIS.

Note that this will not work if your application's code binds to a specific port.

- If you are using **IIS 5**, or if you are using **IIS 6 or 7** and your application binds to a specific port, select **Original port**.

IIS will restart so that the profiler can attach to the port.

## Getting more help

If the steps above do not prevent the profiled application from crashing, please contact Red Gate support (<https://www.red-gate.com/supportcenter/ContactSupport?q=purchaseevaluation>), including, if possible, a log of the application crash.

## Troubleshooting missing results

---

ANTS Performance Profiler may show no profiling results, may show no results for some methods, or may show the error message "The profiler did not find any methods with source code", when:

- ANTS Performance Profiler cannot find usable *.pdb* files for your application
- ANTS Performance Profiler cannot find any code to profile
- ANTS Performance Profiler cannot read a performance counter

This topic describes how to resolve each of these problems.

You may also receive no results, or incomplete results, if the application you are profiling crashes. For more information, see [Troubleshooting application crashes](#).

### **ANTS Performance Profiler cannot find usable *.pdb* files for your application**

See [Troubleshooting \*.pdb\* problems](#).

### **ANTS Performance Profiler cannot find any code to profile**

ANTS Performance Profiler may be unable to find your code if:

- The application you are profiling is not installed on the same computer as ANTS Performance Profiler.  
It is not possible to profile remotely using ANTS Performance Profiler. Ensure that the profiler is installed on the computer running the application you want to profile.
- You are profiling with 'Hide insignificant methods' selected.  
This setting hides any method that contributes less than 1% of your application's total CPU time. To display all profiled methods, clear the 'Hide insignificant methods' checkbox.
- Your application contains no managed code.  
Line-level and method-level timings are not available for unmanaged code. If all your application's code is unmanaged, it cannot be profiled.  
Method-level timings for unmanaged methods can be shown if the methods are declared with `extern` within your managed code.

### **For web applications and WCF services running in IIS:**

- The application is using an unprofiled port.  
If you are profiling on an unused port, configure your application to communicate on the same port that you selected on the Application Settings dialog in ANTS Performance Profiler. By default this is port 8013. For more information, see [Profiling ASP.NET applications running on IIS](#).

## ANTS Performance Profiler cannot read a performance counter

If any performance counters are missing from profiling results, rebuild the counters and try profiling again.

To rebuild performance counters:

- For Windows 2000, Windows XP and Windows Server 2003: see How to manually rebuild Performance Counter Library values (<http://support.microsoft.com/kb/300956>)
- For Windows Vista, Windows Server 2008 and Windows 7: see How to rebuild performance counters (<http://blogs.technet.com/b/yongrhee/archive/2009/10/06/how-to-rebuild-performance-counters-on-windows-vista-server2008-7-server2008r2.aspx>)

## Contacting Red Gate support

If you are still unable to resolve this problem, contact Red Gate support (<https://www.red-gate.com/supportcenter/ContactSupport?q=purchaseevaluation>).

Ensure that you include in the **Description** field:

- the version of ANTS Performance Profiler you are using
- your computer's operating system
- the steps you have already tried
- any error messages ANTS Performance Profiler has generated, including any in the log files.

More information about locating log files for ANTS Performance Profiler can be found here ([http://www.red-gate.com/supportcenter/content/ANTS\\_Performance\\_Profiler/knowledgebase/ANTS\\_log\\_files](http://www.red-gate.com/supportcenter/content/ANTS_Performance_Profiler/knowledgebase/ANTS_log_files)).

## Troubleshooting PDB problems

---

If ANTS Performance Profiler cannot locate usable *.pdb* (debugging symbols) files for the application you are profiling, it cannot display method source code or show line-level timings. You may receive the message "The profiler did not find any methods with source code."

This can indicate one or more of the following:

- No *.pdb* file exists for the application.
- A *.pdb* file exists, but ANTS Performance Profiler cannot locate it.
- A *.pdb* file exists, but is out of date.
- A corrupt *.pdb* file causes ANTS Performance Profiler to crash.

This topic explains how to resolve each of these issues.

To resolve other issues causing missing results, see [Troubleshooting missing results](#).

### No *.pdb* file exists for the application

The method for creating a PDB depends on the type of application you are profiling.

Note that, if the application's namespace includes multiple DLLs, ANTS Performance Profiler will look for a PDB for each of them. Line-level profiling results and source-code display will be available only for methods in those DLLs for which ANTS Performance Profiler can locate the PDB.

#### ASP.NET web applications:

1. Close ANTS Performance Profiler.
2. Open the application's *web.config* file.

For instructions, see [How to enable debugging for ASP.NET applications](http://msdn.microsoft.com/en-us/library/e8z01xdh.aspx) (<http://msdn.microsoft.com/en-us/library/e8z01xdh.aspx>)

3. Find the `compilation` tag and set its `debug` attribute to `true`.
4. Restart ANTS Performance Profiler and try to profile your application again.

Note that simply rebuilding the assembly will not work unless debugging is enabled. Once the *.pdb* file has been created, you can set `debug` to `false` without affecting it.

#### Other types of application:

See [Debug settings and preparation](http://msdn.microsoft.com/en-us/library/d0b8xh0y.aspx) (<http://msdn.microsoft.com/en-us/library/d0b8xh0y.aspx>)

### A *.pdb* file exists, but ANTS Performance Profiler cannot locate it

For each of an application's assemblies, ANTS Performance Profiler looks for PDBs in the directory where the assembly's DLL is stored. (For ASP.NET web applications, this is by

default the *bin* or *app\_bin* folder.) If the PDB for a profiled assembly is located somewhere else, move it into this folder to enable line-level profiling and source code display.

### Using a global PDB directory

If you are unable to move the PDBs for all your DLLs into a folder where ANTS Performance Profiler can find them (for example, if your application uses assemblies from the Global Assemblies Cache), you can bypass the problem by creating a global debugging symbols (PDB) directory ([http://www.red-gate.com/supportcenter/Content?p=ANTS%20Performance%20Profiler&c=knowledgebase\ANTS\\_Performance\\_Profiler\KB200806000270.htm](http://www.red-gate.com/supportcenter/Content?p=ANTS%20Performance%20Profiler&c=knowledgebase\ANTS_Performance_Profiler\KB200806000270.htm)).

### A *.pdb* file exists, but is out of date

If the assembly has been changed since its *.pdb* file was generated, the PDB may be out of date. To update it:

1. Enable debugging by following the steps under "No *.pdb* file exists for the application" above.
2. Recompile your assembly.
3. Restart ANTS Performance Profiler and try to profile your application again.

Note that simply rebuilding the assembly will not work unless debugging is enabled.

### A corrupt *.pdb* file causes ANTS Performance Profiler to crash

If ANTS Performance Profiler encounters a corrupt PDB during profiling, the profiler may crash and fail to return results. The corrupt PDB might belong to a third-party DLL or to the application you want to profile.

There are two ways to profile an application with a corrupt PDB:

- Delete the corrupt PDB.  
This will prevent you from viewing source code referenced in the deleted PDB, but will display line-level code for all other methods.
- On the Application Settings dialog, in the **Profiling mode** dropdown menu, choose **Method-level timings; all methods (faster)**.  
This will prevent the corrupt PDB from being read, avoiding the cause of the crash, but will also prevent you from viewing source code referenced in the deleted PDB. Line-level timings are also unavailable in this profiling mode.

### Getting help

If you are still unable to resolve this problem, please contact Red Gate support (<https://www.red-gate.com/supportcenter/ContactSupport?q=purchaseevaluation>). Supply as much information as you can in the **Description** box, including:

- the type of application you are trying to profile
- the version of ANTS Performance Profiler you are using
- your computer's operating system
- the steps you have already tried
- any error messages ANTS Performance Profiler has generated, including any in the log files. To locate log files for ANTS Performance Profiler, see Log files ([http://www.red-gate.com/supportcenter/content/ANTS\\_Performance\\_Profiler/knowledgebase/ANTS\\_log\\_files](http://www.red-gate.com/supportcenter/content/ANTS_Performance_Profiler/knowledgebase/ANTS_log_files)).

## Troubleshooting SharePoint Profiling

---

There are two main problems that you may encounter while profiling SharePoint:

- ANTS Performance Profiler cannot read from the directory which SharePoint writes data to.
- ANTS Performance Profiler cannot profile SharePoint because of security restrictions in ASP.NET.

### **ANTS Performance Profiler cannot read from the directory which SharePoint writes data to**

The most likely cause of problems while profiling Sharepoint is that ANTS Performance Profiler cannot read from the directory which SharePoint writes data to. To fix this:

1. Create a temporary directory
2. If you are not on a sensitive system, allow full read/write access to this temporary directory to all users. If you are on a sensitive system, ensuring that the local system account has read/write access should suffice.
3. Use **Control Panel** to add a new environment variable. The variable must be called *RGIISTEMP* and the value is the path to the temporary directory you just created.

For more information, see Chris Allen's blog post, 'Profiling SharePoint with ANTS Performance Profiler 5.2 (<http://www.simple-talk.com/dotnet/.net-tools/profiling-sharepoint-with-ants-performance-profiler-5.2/>)' (This blog post is also valid for ANTS Performance Profiler 6.3.)

### **ANTS Performance Profiler cannot profile SharePoint because of security restrictions in ASP.NET**

Security features in ASP.NET may cause problems on some systems. To fix this:

1. Obtain the required information.
2. Grant permissions.
3. Ensure that compilation with be in DEBUG configuration.
4. Copy PDBs and web part DLLs to the app\_bin directory.

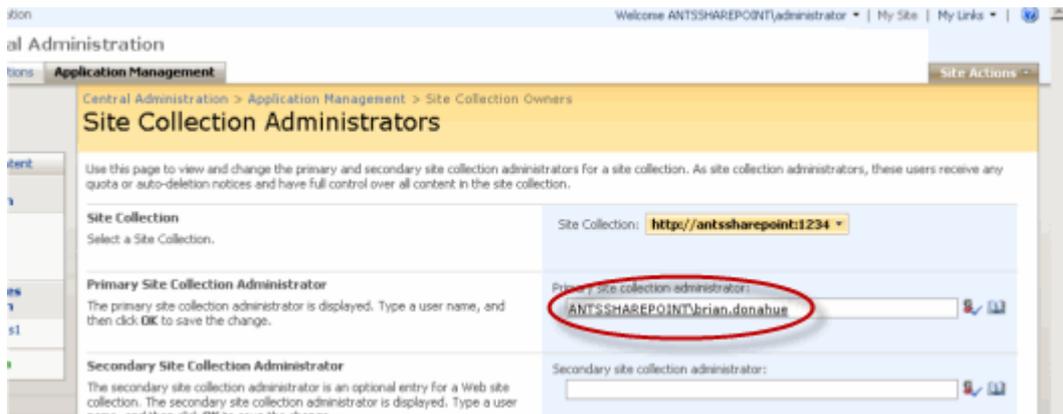
#### **1. Obtain the required information**

Before you start, you need to know the following details for the site collection which you are profiling:

- the URL
- the TCP port
- the name of the primary site collection administrator
- the primary site collection administrator's password

To find the name of the primary site collection administrator:

1. Open the SharePoint Central Administration website using the Start menu item.
2. Click the **Application Management** tab.
3. Under the **SharePoint Site Management** heading, click **Site Collection Administrators**.
4. Select the name of the site collection hosting your web part from the dropdown list.
5. Make a note of the account set in the **Primary Site Collection Administrator** box.

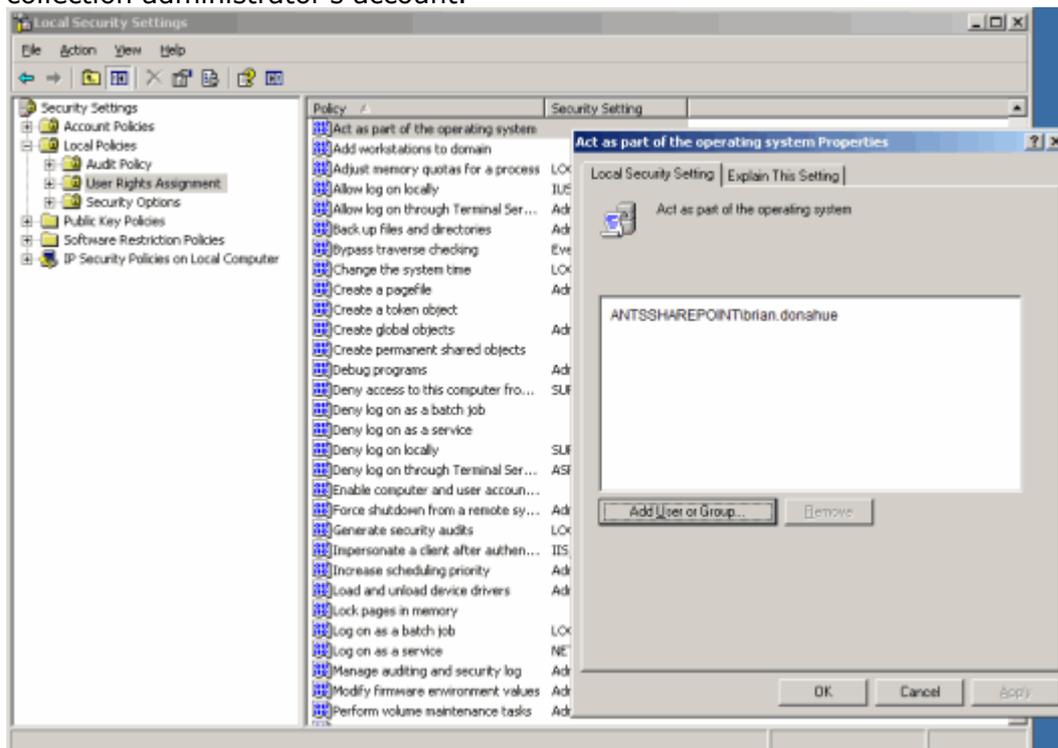


## 2. Grant permissions

The primary site collection administrator must have permission to launch an IIS worker process. To grant this permission:

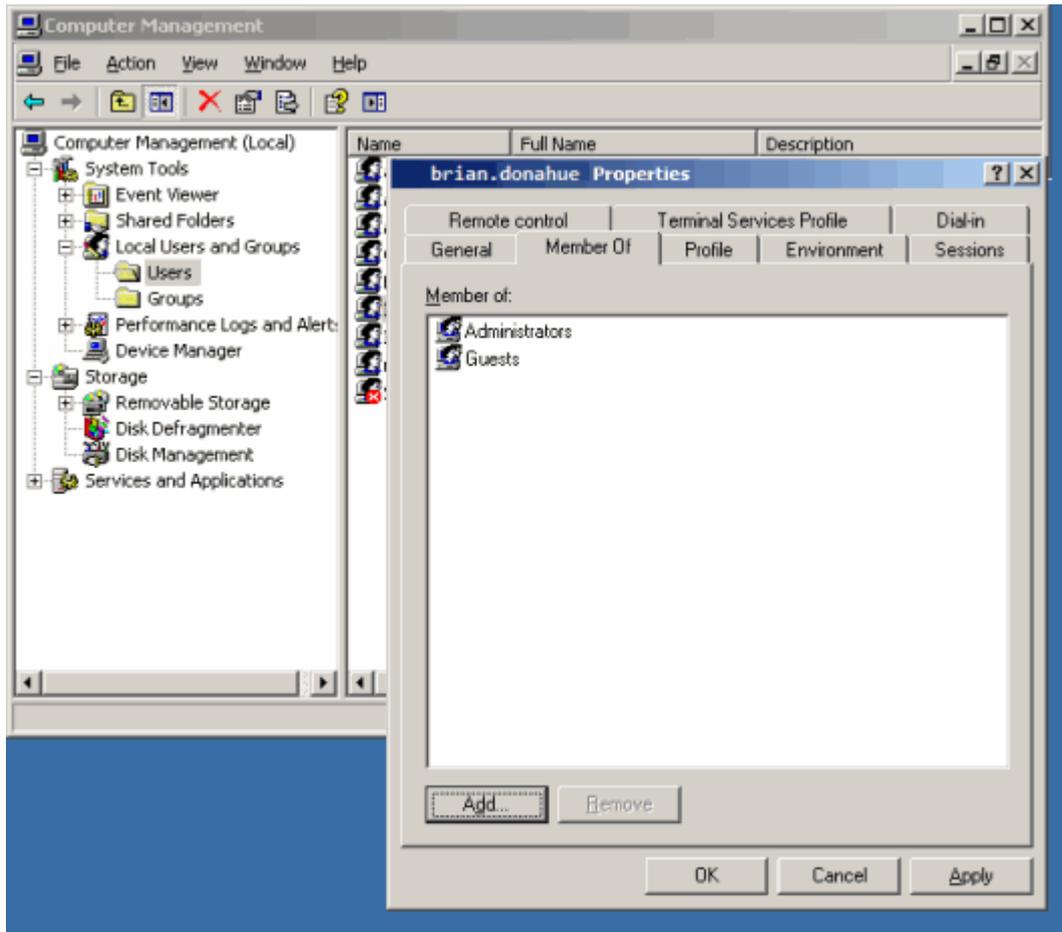
1. Open **Administrative Tools** then open **Local Security Policy**.

2. Under **Local Policies**, click **User Rights Assignment**.
3. Double-click **Act as part of the operating system** and add the primary site collection administrator's account.



4. In the same way, double-click **Impersonate a client after authentication** and add the primary site collection administrator's account.
5. Open a command prompt and run `gpupdate /force` to enforce the new settings.
6. Open **Administrative Tools** and go to **Computer Management**.
7. Under **Local Users and Groups**, open **Users**.
8. Double-click the primary site collection administrator's account and open the **Member Of** tab.

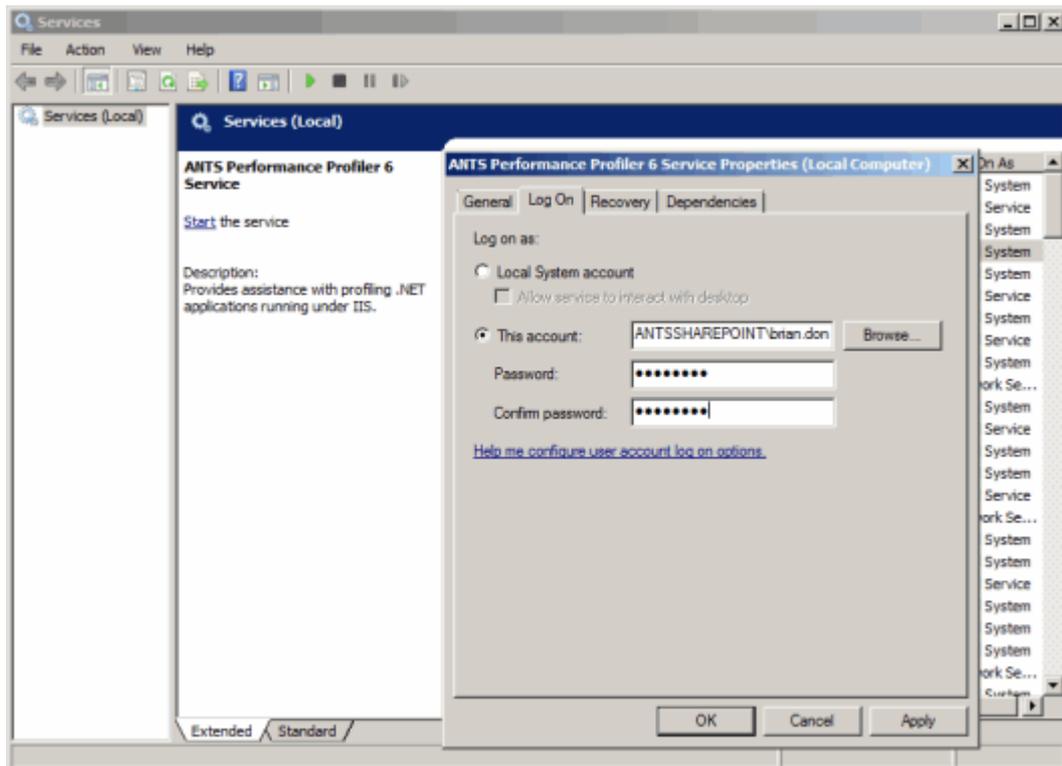
9. Add the *Administrators* group.



The ANTS Performance Profiler 6 Service must use the primary site collection administrator's account when it starts. To configure this:

1. Open **Administrative Tools** then open **Services**.
2. Double-click **ANTS Performance Profiler 6.3 Service**.
3. Click the **Log On** tab.
4. Select **This Account** and enter the primary site collection administrator's username and password.
5. Click **OK**.

6. If the status of ANTS Performance Profiler 6 Service is *Started*, right-click the service and click **Restart**.



### 3a. Ensure that compilation will be in DEBUG configuration (IIS 7)

(Instructions for IIS 6 are below)

To set the application in DEBUG configuration in IIS 7:

1. Load **IIS Manager**.
2. Click the web application you want to profile.
3. Click the **.NET Compilation** option.
4. Under the **Behavior** group, set **Debug** to *True*.

Continue reading the instructions in section "4. Copy PDBs and web part DLLs to the app\_bin directory", below.

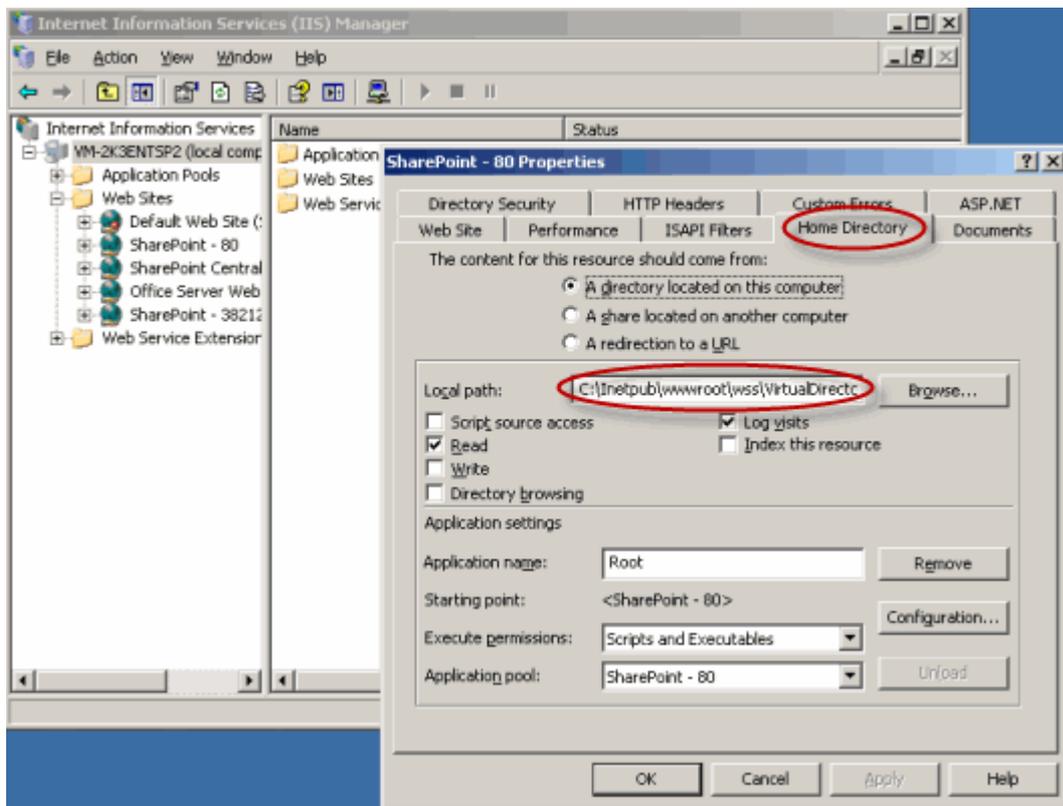
### 3b. Ensure that compilation will be in DEBUG configuration (IIS 6)

To profile a SharePoint collection, the ASP .NET compilation must be done in DEBUG configuration. This will allow ANTS Performance Profiler to locate the source code for any web parts or other extensions you have written for the site collection. DEBUG configuration will also relax some unmanaged code restrictions that prevent profiling and stop the site from timing out.

To set DEBUG configuration, you must know the physical path to the root of the site collection website.

To find this path:

1. Open **Administrative Tools**.
2. Open **Internet Information Server (IIS) Manager**.
3. Right-click the website containing the site collection then click **Properties**.
4. Open the **Home Directory** tab.
5. Note the path in the **Local path** box.

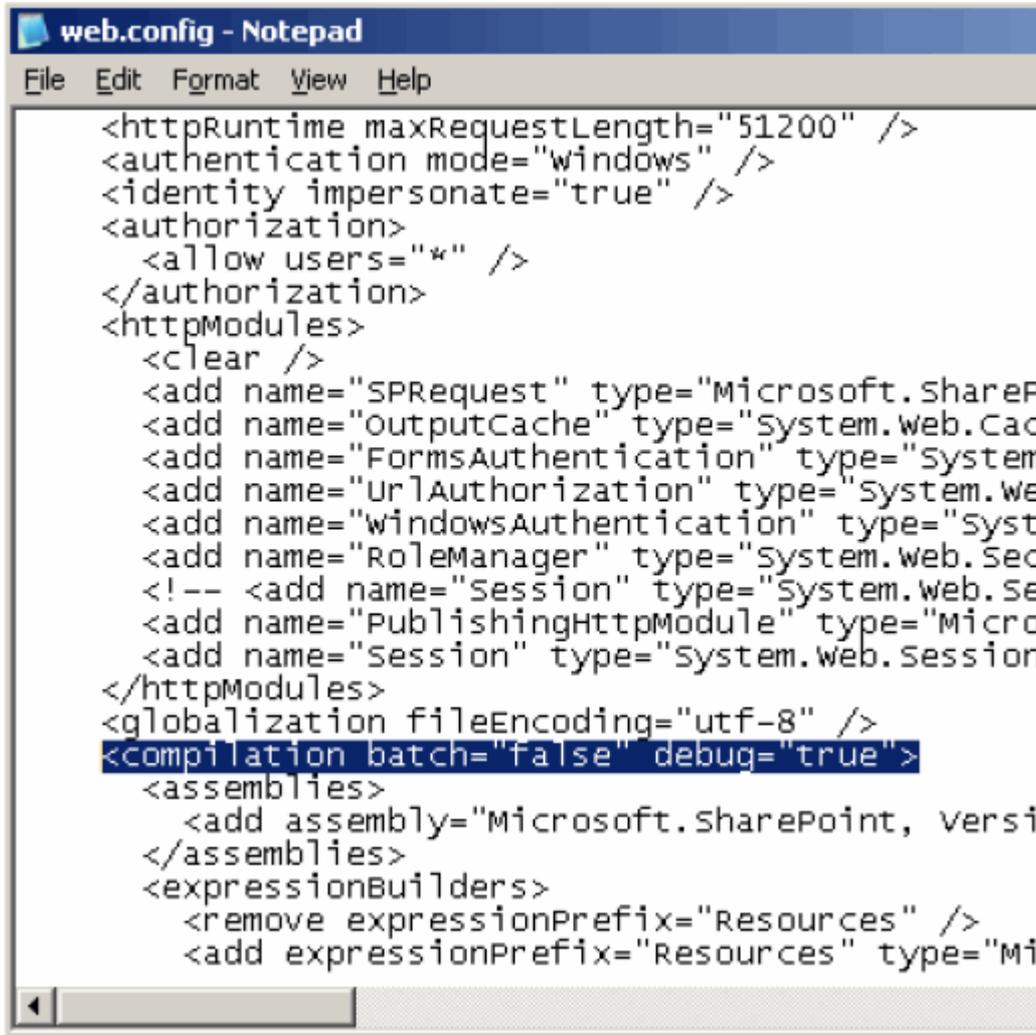


You must now locate and edit the *web.config* file for the site collection using an XML editor.

1. Use Windows Explorer to navigate to the site collection root's physical path.
2. Right-click the *web.config* file.
3. Open the *web.config* file using Notepad. Search for the text **debug**.

#### 4. Change

```
<compilation batch="false" debug="false">  
to  
<compilation batch="false" debug="true">  
and save the file.
```



```
web.config - Notepad  
File Edit Format View Help  
<httpRuntime maxRequestLength="51200" />  
<authentication mode="windows" />  
<identity impersonate="true" />  
<authorization>  
  <allow users="*" />  
</authorization>  
<httpModules>  
  <clear />  
  <add name="SPRequest" type="Microsoft.ShareP  
  <add name="OutputCache" type="System.Web.Cac  
  <add name="FormsAuthentication" type="System  
  <add name="UrlAuthorization" type="System.We  
  <add name="windowsAuthentication" type="syst  
  <add name="RoleManager" type="system.web.Sec  
  <!-- <add name="session" type="System.Web.Se  
  <add name="PublishingHttpModule" type="Micro  
  <add name="session" type="System.Web.Session  
</httpModules>  
<globalization fileEncoding="utf-8" />  
<compilation batch="false" debug="true">  
  <assemblies>  
    <add assembly="Microsoft.SharePoint, versi  
  </assemblies>  
  <expressionBuilders>  
    <remove expressionPrefix="Resources" />  
    <add expressionPrefix="Resources" type="Mi
```

#### 4. Copy PDBs and web part DLLs to the app\_bin directory

If you want to be able to filter out all methods except the ones run by your code when viewing the results, you must copy the relevant files into the site's app\_bin directory. To do this:

1. Copy all PDB files and any web part DLLs used by your site to the Clipboard.
2. Use Windows Explorer to navigate to the site collection root's physical path.
3. Open the *app\_bin* directory.
4. Paste all PDB files and any web part DLLs used by your site into this directory.

Continue following the instructions in Setting up the Performance Profiler, above.

## Troubleshooting IIS profiling

---

When you click **Start Profiling** for an ASP.NET web application (IIS), profiling may not start, and a "Cannot start IIS" error may be displayed. This indicates one or more of the following:

- An early access build of the ANTS Performance Profiler continuous profiling tool prevents ANTS Performance Profiler from accessing IIS.
- The logged-in user has insufficient account permissions to run the web application.
- Internet Explorer is running in protected mode.
- IIS is unable to resolve the web application's URL.
- ANTS Performance Profiler encounters a conflict with another performance profiler installed on your computer.

This topic describes how to resolve each of these issues.

### **An early access build of the ANTS Performance Profiler continuous profiling tool prevents ANTS Performance Profiler from accessing IIS**

The continuous profiling tool currently runs as a separate tool from the main ANTS Performance Profiler product. If you have installed an early access build of the ANTS Performance Profiler continuous profiler IIS module, other profilers - including the desktop ANTS Performance Profiler product - will be unable to profile applications running in IIS on this computer. To re-enable other profilers with IIS, uninstall the IIS Profiler Module:

1. From your computer's Start menu, launch the **Continuous Profiling Configuration Tool**.
2. Click **Uninstall**.

For more information on configuring continuous profiling, see [Setting up continuous profiling](#).

### **The logged-in user has insufficient account permissions to run the web application**

ANTS Performance Profiler starts the IIS application pool with permissions inherited from the **currently logged-in user account**, rather than using the IIS application settings.

- If possible, run ANTS Performance Profiler as an administrator.
- If you are unable to run ANTS Performance Profiler as an administrator, grant the logged-in user account permissions to access the IIS configuration system and write to the ASP.NET temporary files. For more information on how to do this, see [Assign ASP.NET Permissions to the New Account \(http://msdn.microsoft.com/en-us/library/ms998297.aspx#paght000009\\_step2\)](http://msdn.microsoft.com/en-us/library/ms998297.aspx#paght000009_step2) (MSDN).

### Manually specifying the ASP.NET account

The error may also occur when using "Manually specify ASP.NET account details". Check that the specified account is a valid user, has administrator privileges, and has read access to `%ProgramFiles%\Red Gate\ANTS Performance Profiler 6\ProfilerCore.dll`.

### Internet Explorer is running in protected mode

If Protected Mode is turned on in Internet Explorer, when you click **Start Profiling**, a browser session may invisibly start and quickly terminate. If this occurs, Internet Explorer may launch with the message "Internet Explorer cannot display the webpage" or "Could not connect to the remote server", and no profiling results are displayed.

To prevent this error:

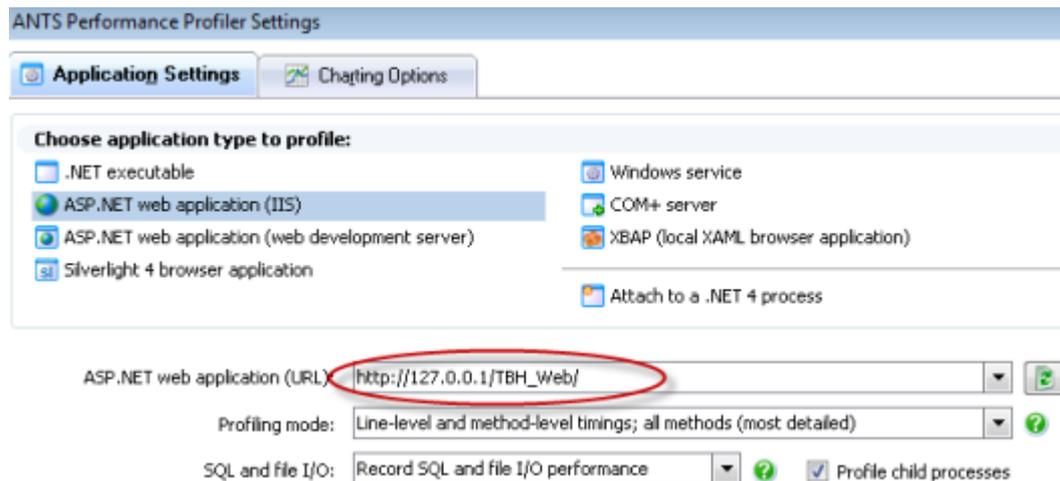
- If possible, turn off Protected Mode in Internet Explorer before starting a profiling session.
- If you need to use Protected Mode, add `localhost` to the list of trusted sites, and try to profile your application again. For instructions on adding a site to Internet Explorer's Trusted Sites list, see [Security zones: adding or removing websites \(http://windows.microsoft.com/en-US/windows7/Security-zones-adding-or-removing-websites\)](http://windows.microsoft.com/en-US/windows7/Security-zones-adding-or-removing-websites) (Microsoft).

### IIS cannot resolve the web application's URL

If the bindings in IIS have been changed from the default, ANTS Performance Profiler may be unable to resolve your site's hostname. If this problem occurs, the following error message is usually shown:

"Couldn't determine the IIS Site associated with URL 'http://< URL>:port'. Please check that the URL is serviced by the instance of IIS running on this machine."

In the **ASP.NET web application (URL)** field, enter *localhost* or the loopback IP address (*127.0.0.1*) and try to profile your application again:



### ANTS Performance Profiler encounters a conflict with another performance profiler installed on your computer

IIS can fail to start if ANTS Performance Profiler encounters a conflict with another performance profiler. We recommend uninstalling other profilers while profiling with ANTS Performance Profiler.

#### If you are running ANTS Performance Profiler Version 5.2 and earlier:

If the error persists after uninstalling other profilers, you may need to remove environment variables left behind by an earlier profiling session:

1. Close ANTS Performance Profiler and IIS.
2. In regedit.exe, locate the following registry key:  
If you are running IIS version 6.0 or earlier:  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\IISADMIN`  
If you are running IIS version 7.0 or later:  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC`
3. Expand the key and modify the **Environment** subkey to delete the following values:  
`COR_ENABLE_PROFILING=1` and `COR_PROFILER={a GUID}`.  
**Note:** if the **Environment** subkey does not exist, please contact Red Gate support (see below for more details).
4. Close the Registry Editor, restart ANTS Performance Profiler, and try to profile your application again.

### Contacting Red Gate support

If you are unable to resolve this problem using the information in this topic, please contact support (<https://www.red->

gate.com/supportcenter/ContactSupport?q=purchaseevaluation), and supply as much information as you can in the **Description** box, including:

- the versions of IIS and ANTS Performance Profiler you are using
- your computer's operating system
- the steps you have already tried
- any error messages ANTS Performance Profiler has generated, including any in the log files. More information about locating log files for ANTS Performance Profiler can be found here ([http://www.red-gate.com/supportcenter/content/ANTS\\_Performance\\_Profiler/knowledgebase/ANTS\\_log\\_files](http://www.red-gate.com/supportcenter/content/ANTS_Performance_Profiler/knowledgebase/ANTS_log_files)).

## 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 Log Rescue, SQL Multi Script, SQL Packager, SQL Prompt, SQL Refactor, SQL Response, SQL Toolbelt, 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.

### 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.