ANTS Memory Profiler 7 documentation

About ANTS Memory Profiler

ANTS Memory Profiler enables you to profile memory usage 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 improve memory usage by identifying the objects and classes that use most memory, and objects that remain live the longest.

You can use ANTS Memory 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.

For more information, see the ANTS Memory Profiler product page.

Quick start guide

The exact procedure you use to find your memory problem depends on your application and what you are trying to achieve. In all cases, there are five general steps:

1. Choose the application that you want to profile
2. Identify what kind of memory problem you have
3. Take one or more memory snapshots
4. Follow an appropriate strategy for the type of problem
5. Analyze and interpret the profiling results

Some knowledge of .NET memory management is useful before you start. If you need a refresher on this, see our Memory management primer.

Requirements

You can use ANTS Memory 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)

Some filters and functionality are not available when profiling .NET 1.1 applications; see the List of object filters for more details.

Installing

Most Redgate products are available as part of a bundle. You can select which individual products to install when you run the installer.

When you install a non-free product, you have 14 days to evaluate the product. For the DLM Automation Suite, DLM Automation Suite for Oracle, SQL Source Control, Schema Compare for Oracle, Data Compare for Oracle, and Source Control for Oracle, you have 28 days. For more information, see Licensing.

To install a Redgate product:

1. Download the product from the website.
2. Run the installer and follow the instructions.

The product is listed on the **Start** menu under **Red Gate**.

Licensing

When you install most Redgate products (apart from free ones), you have **14 days** to evaluate them without purchase.

For a few products, you have 28 days: DLM Automation Suite, DLM Automation Suite for Oracle, SQL Prompt, SQL Source Control, Source Control for Oracle.

If you need more time to evaluate a product, email licensing@red-gate.com.

Finding your serial number

When you buy a license for a product, we'll send you an invoice that contains your serial number to activate the product. Your invoice shows how many instances of a product the serial number can be used to activate. For information about how to activate, see Activating.

If you can't find your invoice, you can view your serial numbers at red-gate.com/myserialnumbers. You'll need to log in to your Redgate account with the email address and password you provided when you bought the product.

> If you need to reinstall products on the same computer (eg after installing a new operating system), you can reactivate them using the same serial number. This doesn't affect the number of distinct activations for the serial number. For information about moving a serial number to a different computer, see below.

Serial numbers for bundles and suites

If you've bought a bundle or suite of products, your serial number activates all the products in the bundle or suite. For bundles containing both server and client tools (such as the SQL DBA Bundle) you will have two serial numbers.

If you deactivate a bundle or suite serial number, all products using that serial number will be deactivated.

For information on which products are included in a bundle, see Bundle history.

Changing the serial number used to activate a product

To change the serial number used to activate a product, on the **Help** menu, select **Enter Serial Number**. For some products, you will need to deactivate the old serial number first.

Moving a serial number to a different computer

To move a serial number to a different computer, deactivate the serial number on the old computer, then use it to activate the product on the new computer.

To deactivate a serial number, on the **Help** menu, select **Deactivate Serial Number**. If the Deactivate Serial Number menu item isn't available, use the deactivation tool.

If you can't deactivate a serial number, use the Request Extra Activations page to request more activations for your serial number. You'll need to provide your serial number and the reason for the additional activations.

Activating

This page applies to a number of Redgate products, so the screenshots below may not match your product.

When you activate a product with your serial number, the licensing and activation program sends an activation request to the Redgate activation server, using checksums of attributes from your computer. The checksums sent to the activation server do not contain any details that might pose a security risk. The activation server returns an activation response and an encrypted key to unlock the software. The licensing and activation program should activate your product within a few seconds.

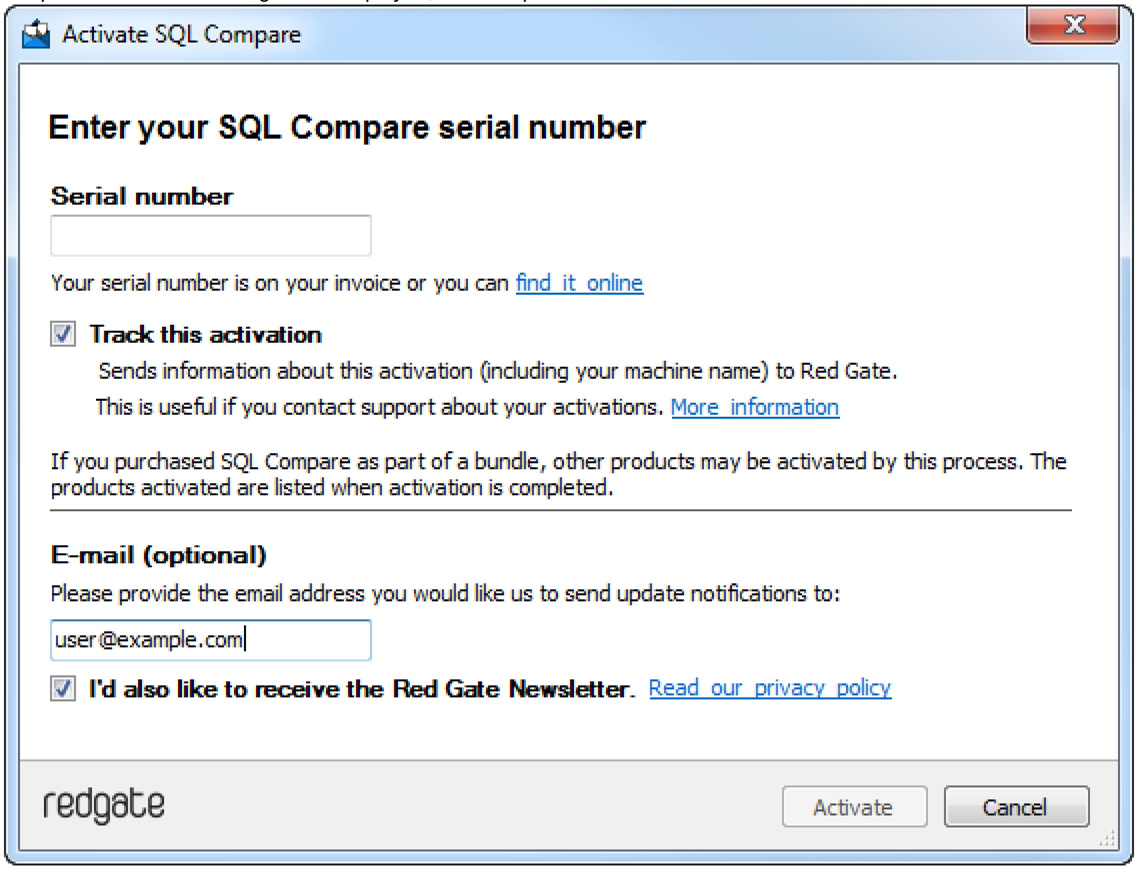
If you experience problems with activating your products, you'll be directed to activate manually.

- Activating using the GUI
- Activating using the command line
- Manual activation

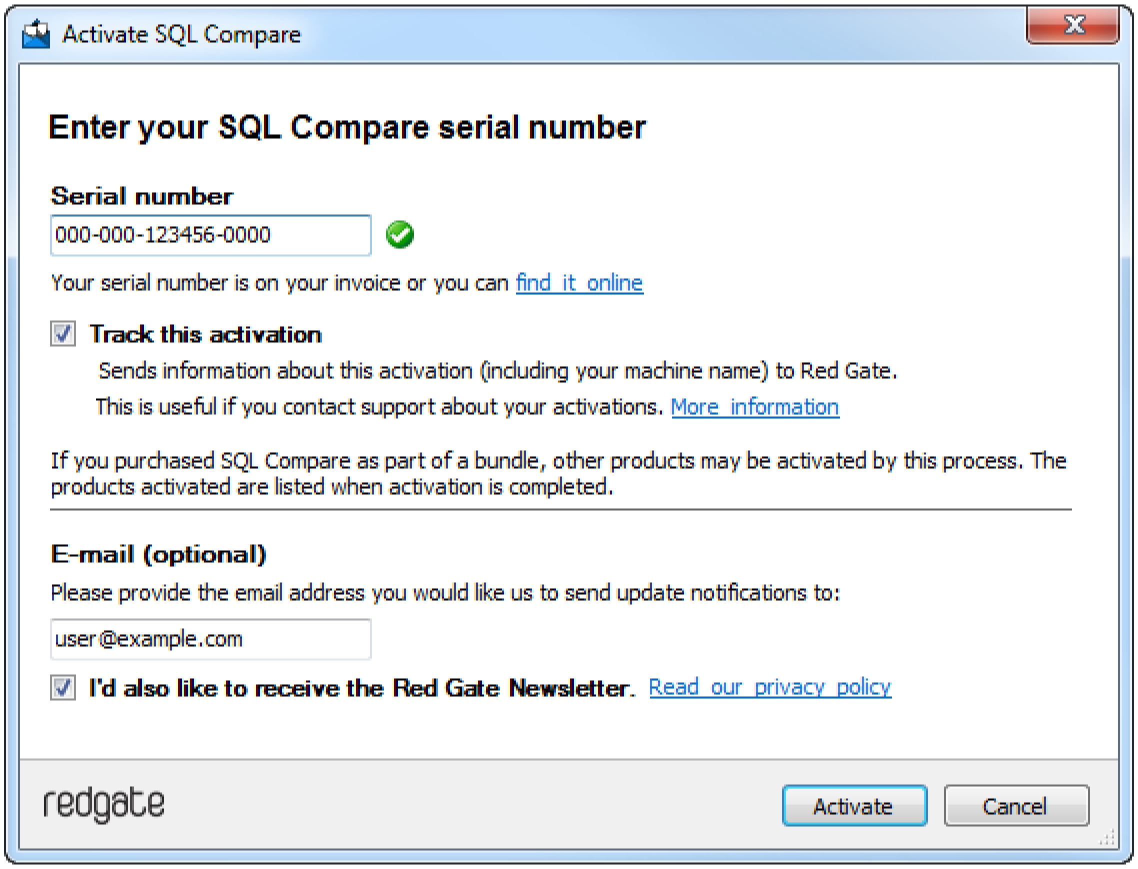
Activating using the GUI

These instructions apply to a number of Redgate products, so the screenshots below may not match your product.

To activate your products:

1. On the **Help** menu, click **Enter Serial Number**.
   The product activation dialog box is displayed, for example:
   
   Activate SQL Compare
   Enter your SQL Compare serial number
   Serial number
   Your serial number is on your invoice or you can find it online
   Track this activation
   Sends information about this activation (including your machine name) to Red Gate.
   This is useful if you contact support about your activations. More information
   If you purchased SQL Compare as part of a bundle, other products may be activated by this process. The products activated are listed when activation is completed.
   E-mail (optional)
   Please provide the email address you would like us to send update notifications to:
   user@example.com
   I'd also like to receive the Red Gate Newsletter. Read our privacy policy
   redgate
   Activate
   Cancel

2. Enter your serial number.
   When you have entered a valid serial number,
   ✓
   is displayed next to the serial number box:


Activate SQL Compare

Enter your SQL Compare serial number

Serial number

000-000-123456-0000 ✓

Your serial number is on your invoice or you can find it online

[✓] Track this activation

Sends information about this activation (including your machine name) to Red Gate.

This is useful if you contact support about your activations. More information

If you purchased SQL Compare as part of a bundle, other products may be activated by this process. The products activated are listed when activation is completed.

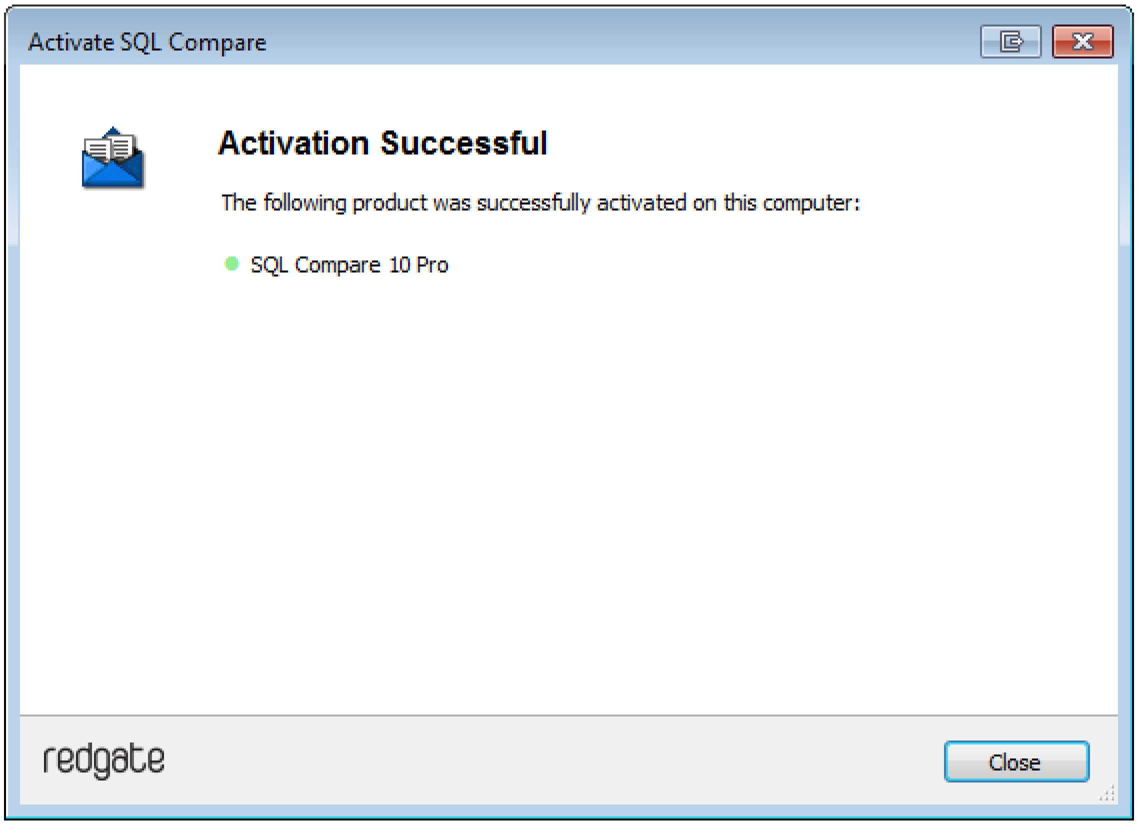
E-mail (optional)

Please provide the email address you would like us to send update notifications to:

user@example.com

[✓] I'd also like to receive the Red Gate Newsletter. Read our privacy policy

3. If you want to receive email updates from Redgate, enter your email address.
   The list of identifiers and your email address may already be populated using information available to the licensing client from the Windows installation on your computer. No information is sent back to Redgate when the fields are populated.
   When you activate your product, the optional information you entered is recorded by Redgate with your serial number. Your email address is not linked to the data collected should you consent to participate in the Quality Improvement Program provided with some Red Gate products.
4. Click **Activate**.
   Your activation request is sent to the Red Gate activation server.
   When your activation has been confirmed, the **Activation successful** page is displayed, for example:


Activate SQL Compare

Activation Successful

The following product was successfully activated on this computer:

● SQL Compare 10 Pro

If there is a problem with your activation request, an error dialog box is displayed. For information about activation errors and what you can do to resolve them, see Troubleshooting licensing and activation errors. Depending on the error, you may want to try manual activation.

5. Click **Close**.
   You can now continue to use your product.

Activating using the command line

Open a command prompt, navigate to the folder where your product executable file is located and run a command with the following syntax:

```
<name of productEXE> /activateSerial:<serialNumber>
```

For example:

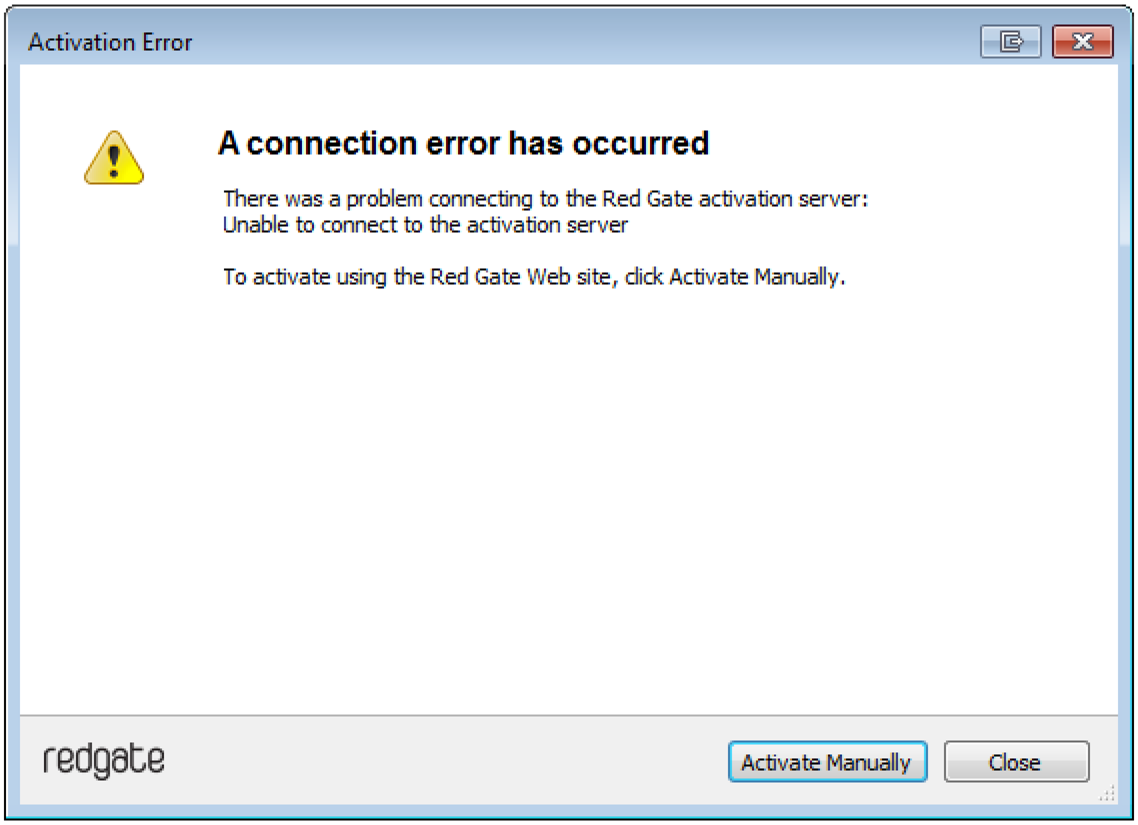
```
sqlcompare /activateSerial:123-456-789012-ABCD
```

The product activation dialog box is displayed. Follow the instructions below.

Manual activation

Manual activation enables you to activate products when your computer does not have an internet connection or your internet connection does not allow SOAP requests. You will need access to another computer that does have an internet connection.

You can use manual activation whenever the **Activation Error** dialog box is displayed and the **Activate Manually** button is available, for example:


Activation Error

A connection error has occurred

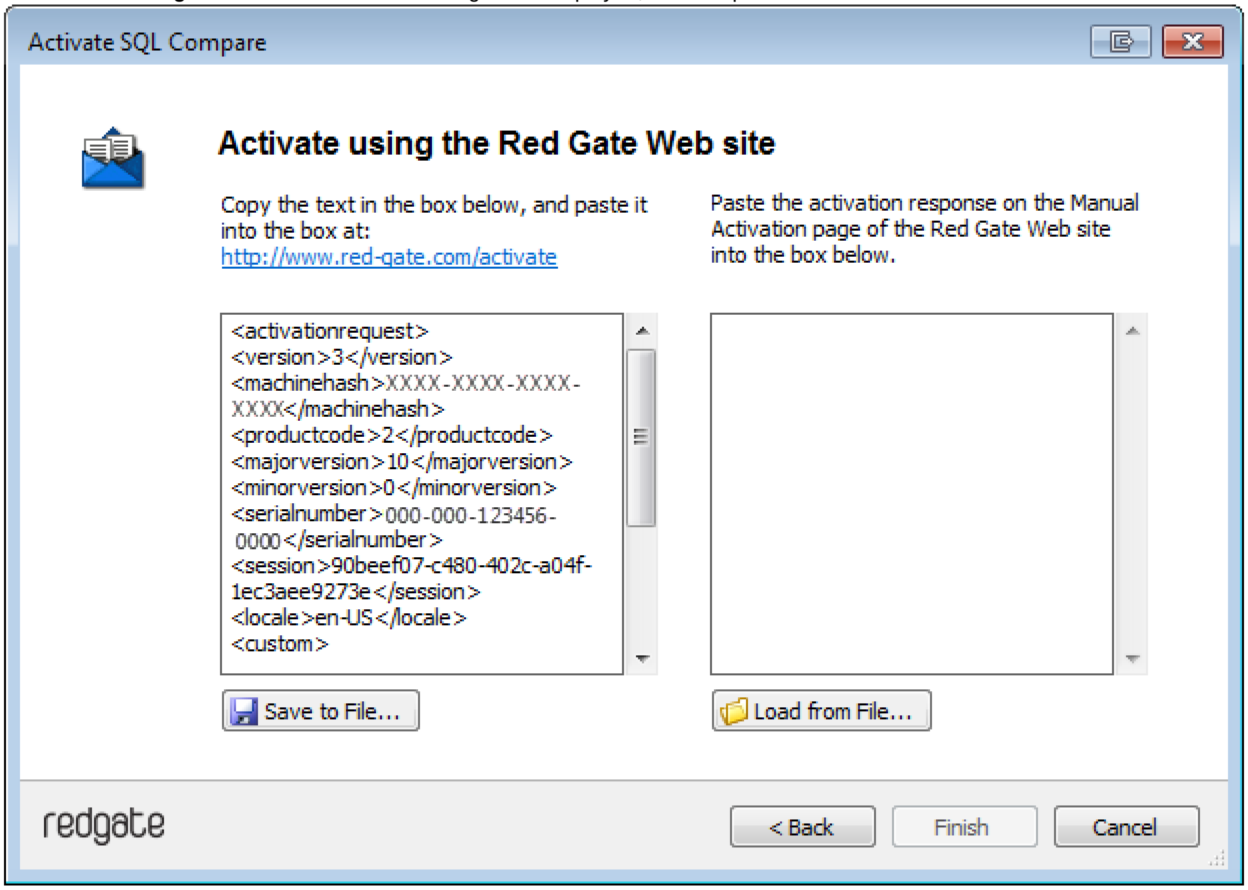
There was a problem connecting to the Red Gate activation server:
Unable to connect to the activation server

To activate using the Red Gate Web site, click Activate Manually.

To activate manually:

1. On the error dialog box, click **Activate Manually**.

   The **Activate using the Red Gate Web site** dialog box is displayed, for example:


Activate SQL Compare

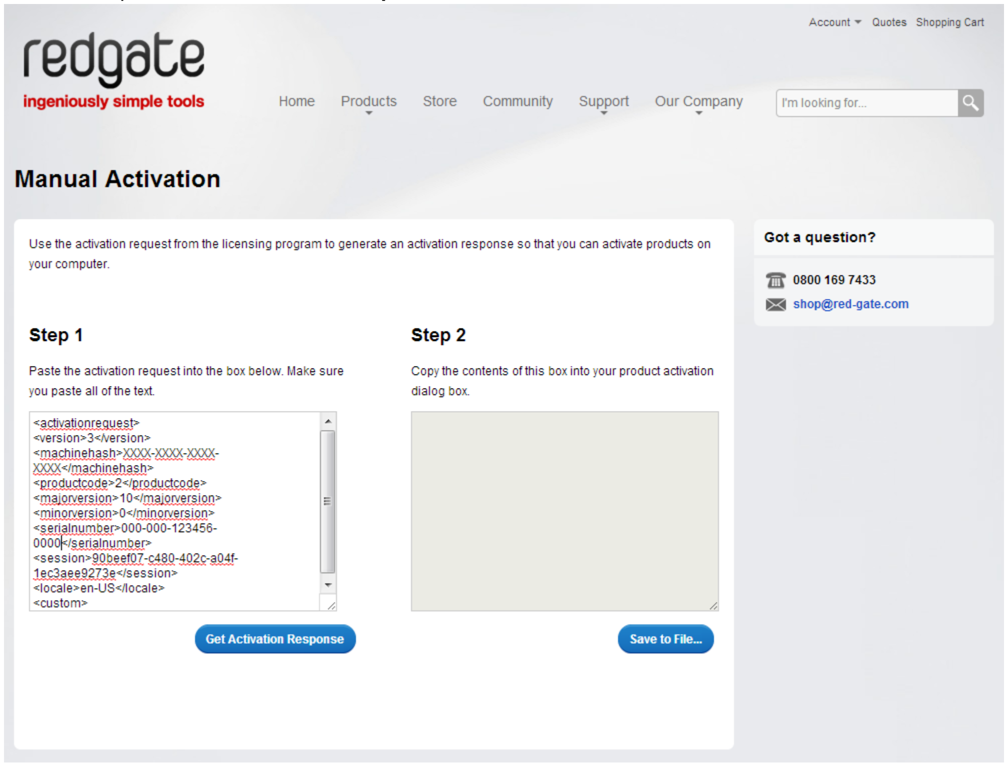
Activate using the Red Gate Web site

Copy the text in the box below, and paste it into the box at:
http://www.red-gate.com/activate

Paste the activation response on the Manual Activation page of the Red Gate Web site into the box below.

```
<activationrequest>
<version>3</version>
<machinehash>XXXX-XXXX-XXXX-
XXXX</machinehash>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<serialnumber>000-000-123456-
0000</serialnumber>
<session>90beef07-c480-402c-a04f-
1ec3aee9273e</session>
<locale>en-US</locale>
<custom>
```

2. Copy all of the activation request, and **leave this dialog box open** (if you close the dialog box, you may have to start again).
   Alternatively you can save the activation request, for example to a location on your network or to a USB device.
3. On a computer that has an Internet connection, go to the **Manual Activation** page at http://www.red-gate.com/activate and paste the activation request into the box under **Step 1**.



Manual Activation

Use the activation request from the licensing program to generate an activation response so that you can activate products on your computer.

Got a question?

0800 169 7433

shop@red-gate.com

Step 1

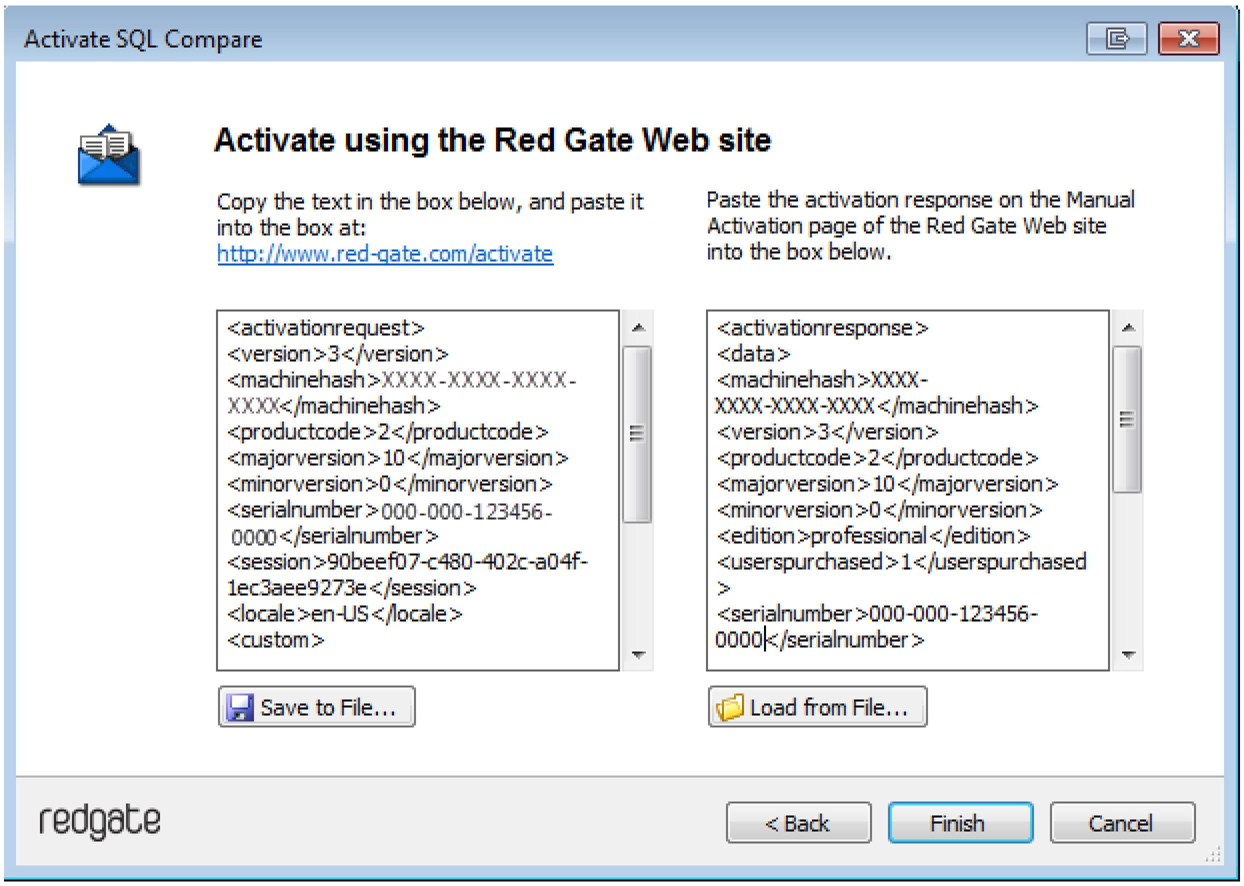
Paste the activation request into the box below. Make sure you paste all of the text.

```
<activationrequest>
<version>3</version>
<machinehash>XXXX-XXXX-XXXX-
XXXX</machinehash>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<serialnumber>000-000-123456-
0000</serialnumber>
<session>90beef07-c480-402c-a04f-
1ec3aee9273e</session>
<locale>en-US</locale>
<custom>
```

Step 2

Copy the contents of this box into your product activation dialog box.

4. Click **Get Activation Response**.
5. When the activation response is displayed under **Step 2**, copy all of it.
   Alternatively you can save the activation response to a .txt file.
6. On the computer where the licensing and activation program is running, paste the activation response or if you saved it, load it from the file.



Activate using the Red Gate Web site

Copy the text in the box below, and paste it into the box at: http://www.red-gate.com/activate

```
<activationrequest>
<version>3</version>
<machinehash>XXXX-XXXX-XXXX-
XXXX</machinehash>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<serialnumber>000-000-123456-
0000</serialnumber>
<session>90beef07-c480-402c-a04f-
1ec3aee9273e</session>
<locale>en-US</locale>
<custom>
```

Paste the activation response on the Manual Activation page of the Red Gate Web site into the box below.

```
<activationresponse>
<data>
<machinehash>XXXX-
XXXX-XXXX-XXXX</machinehash>
<version>3</version>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<edition>professional</edition>
<userspurchased>1</userspurchased
>
<serialnumber>000-000-123456-
0000</serialnumber>
```

7. Click **Finish**.
   The **Activation successful** page is displayed.
8. Click **Close**.
   You can now continue to use your product.

Deactivating

This page applies to several Redgate products, so the screenshots below may not match your product.

Download deactivation tool

You can use the deactivation tool to deactivate a serial number so you can reuse it on another computer. You can also use it to deactivate serial numbers for products you've uninstalled.

When you deactivate a serial number for a bundle of products, all the products in the bundle are deactivated. For information about what products are in your bundle, see Bundle history.

To deactivate a serial number, your computer must have an internet connection. If you can't deactivate a serial number, you can request additional activations for that serial number. You may need to do this if:

- your computer doesn't have an internet connection
- your network uses a proxy server that interrupts contact between the product and the Redgate activation server
- your serial numbers aren't displayed in the deactivation tool (eg if the product installation is corrupted)

Deactivating using the command line

Open a command prompt, navigate to the folder where your product executable file is located and run a command with the following syntax:

```
<productEXE> /deactivateSerial
```

For example:

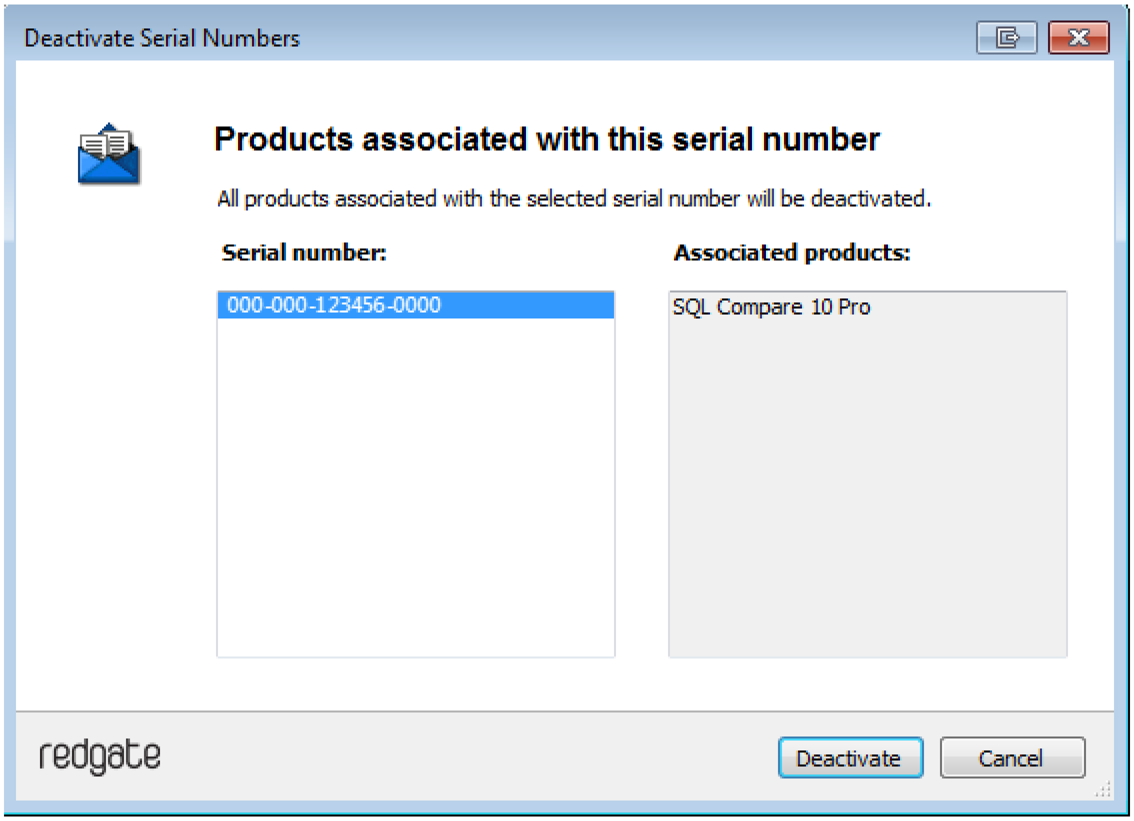
```
sqlcompare /deactivateSerial
```

The **Deactivate Serial Numbers** dialog box is displayed. Follow the instructions below.

Deactivating using the GUI

To deactivate your products:

1. Start the deactivation tool. To do this, either download the tool and run the executable file, or on the **Help** menu of the product, click **Deactivate Serial Number**.
   The **Deactivate Serial Numbers** dialog box is displayed. For example:


Deactivate Serial Numbers

Products associated with this serial number

All products associated with the selected serial number will be deactivated.

Serial number:

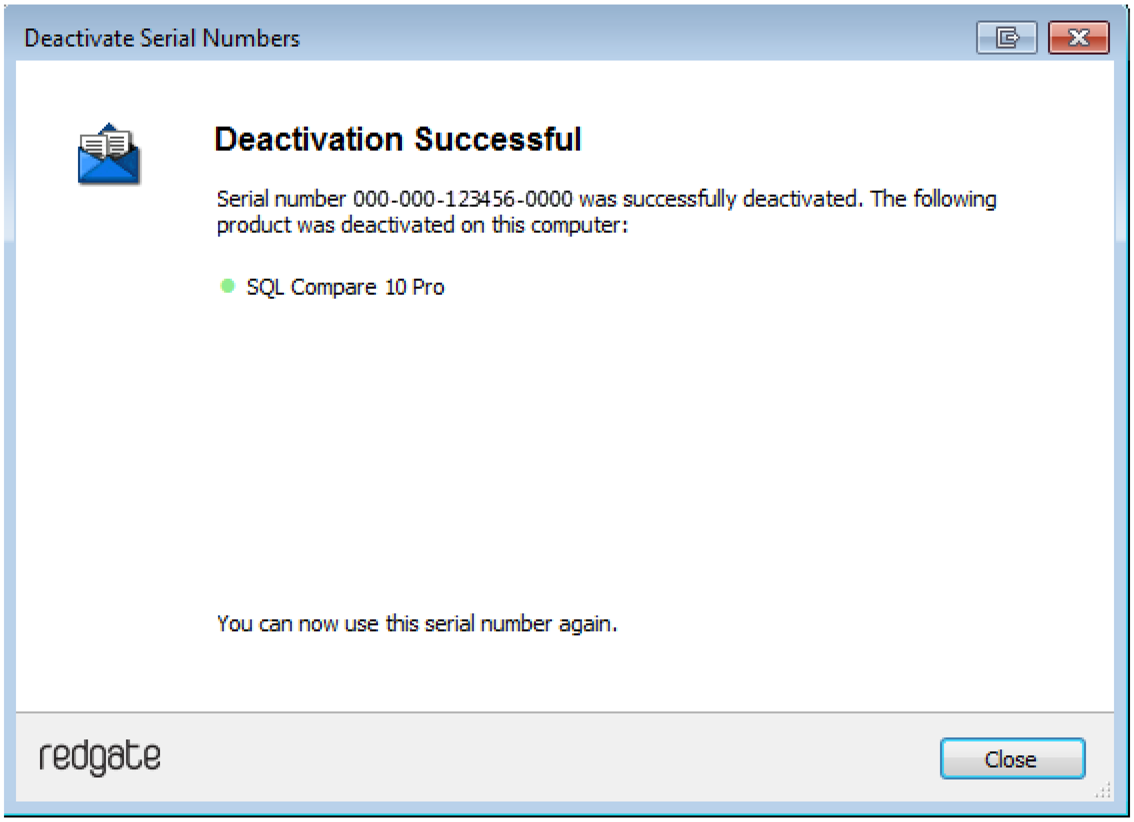
000-000-123456-0000

Associated products:

SQL Compare 10 Pro

If you're running the executable file, the dialog box displays all the serial numbers for Red Gate products that have been activated on your computer.
If the serial number is for a bundle, all the products in the bundle are displayed under **Associated products**.

2. Select the serial number you want to deactivate and click **Deactivate**.
   Your deactivation request is sent to the Red Gate activation server.
3. When your deactivation has been confirmed, the **Deactivation successful** page is displayed. For example:


Deactivate Serial Numbers

Deactivation Successful

Serial number 000-000-123456-0000 was successfully deactivated. The following product was deactivated on this computer:

- SQL Compare 10 Pro

You can now use this serial number again.

If there's a problem with your deactivation request, an error dialog box is displayed. For information about deactivation errors and how to resolve them, see Troubleshooting licensing and activation errors.

4. Click **Close**. You can now use this serial number on a different computer.

Troubleshooting licensing and activation

This page provides information about errors you may encounter when you activate Redgate products:

The number of activations for this serial number has been exceeded

This error message is displayed when a serial number is activated on more computers than the number of licenses that were purchased for that serial number.

When you purchase products from Redgate, we send you an invoice that includes your serial numbers. The serial numbers enable you to activate the software a number of times, depending on how many licenses you purchased and the terms in the license agreement. When this limit is reached, you will see this error message.

To fix the problem, you can:

- deactivate the product on another computer to free up a license
- purchase more licenses
- request additional activations for your serial number

This serial number has been disabled

This error message is displayed when you try to activate a product using a serial number that Redgate has disabled.

When you upgrade a product, your existing serial numbers will be disabled and we will issue new ones with your invoice. If you cannot find your new serial numbers, you can review them at http://www.red-gate.com/myserialnumbers

Redgate will also disable serial numbers for non-payment of invoices or breach of the terms in the license agreement. If you think we have disabled your serial numbers in error, email licensing@red-gate.com

This serial number was for a trial extension

This error message is displayed when you have requested a trial extension and you try to reuse the serial number that was provided for the trial extension; trial extensions can be used one time only.

To continue using the product, you need to purchase it.

This serial number is not registered with the activation server

This error message is displayed when the serial number you entered does not exist on the Redgate activation server.

To find out your serial numbers, check your invoice or go to http://www.red-gate.com/myserialnumbers

This serial number is not for <product name>

This error message is displayed when the serial number you entered is not for the product you are trying to activate.

To find out your serial numbers, check your invoice or go to http://www.red-gate.com/myserialnumbers

This serial number is not for this version

This error message is displayed when the serial number you entered is for a different version of the product you are trying to activate.

If the serial number is for an older version of the product, and you don't have that version installed on your computer, you can download it from the Release notes and other versions page.

If you want to upgrade to the latest version of the product, go to the Upgrade center to get a quote or purchase an upgrade, or email sales@red-gate.com.

The activation request is in the wrong format

This error message is displayed:

- if your internet connection does not allow SOAP requests.
  Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed.
- if you are activating by email and there is a problem with the format of the activation request.
  Check that you copied and pasted all of the activation request.
  Alternatively, try using manual activation. Go to http://www.red-gate.com/activate and paste your activation request under **Step 1**.
- when you are using manual activation and there is a problem with the format of the activation request. If the format is incorrect, for example part of the request is missing, the Redgate activation server cannot process the request.
  Check that you copied and pasted all of the activation request.

For more information about activating manually, see Manual activation.

The activation request contains an invalid machine hash

This error message is displayed:

- if your internet connection does not allow SOAP requests.
  Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see Manual activation.
- when you are using manual activation and there is a problem with the format of the *machinehash* element in the activation request. The *machinehash* is a checksum of attributes from your computer. We use the *machinehash* to identify computers on which our products have been activated. If the format of the *machinehash* element is incorrect, the Redgate activation server cannot process the request.
  Check that you copied and pasted the activation request correctly.

The activation request contains an invalid session

This error message is displayed:

- if your internet connection does not allow SOAP requests.
  Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see Manual activation.
- when you are using manual activation and there is a problem with the format of the activation request. If the format of the *session* element is incorrect, the Redgate activation server cannot process the request.
  Check that you copied and pasted the activation request correctly.

The activation request contains an invalid serial number

This error message is displayed:

- if your internet connection does not allow SOAP requests.
  Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see Manual activation.
- when you are using manual activation and there is a problem with the format of the activation request. If the format of the serial number is incorrect, the Redgate activation server cannot process the request.
  Check that you copied and pasted the activation request correctly.

The activation request contains an invalid product code or version number

This error message is displayed:

- if your internet connection does not allow SOAP requests.
  Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see Manual activation.
- when you are using manual activation and there is a problem with the format of the activation request. If the product code or version numbers are incorrect, the Redgate activation server cannot process the request.
  Check that you copied and pasted the activation request correctly.

There's a problem deactivating your serial number

This error message is displayed if your computer is not connected to the internet or your internet connection does not allow SOAP requests. You cannot deactivate a serial number if your computer does not have an internet connection.

Try deactivating again later. If the problem persists, contact your system administrator.

If you require more activations because you cannot deactivate your serial number, you can request them on the Request Extra Activations page.

This serial number is not activated on this computer

This error message is displayed when you try to deactivate a serial number that has not been activated on your computer.

If you think the product installation on your computer is corrupt, you can try re-activating the product, and then deactivating the product again.

If you require more activations because you cannot deactivate your serial number, you can request them on the Request Extra Activations page.

Products not activated on this computer

This error message is displayed when you try to deactivate a serial number for a bundle of Redgate products and those products were not activated on your computer.

If you think the product installation on your computer is corrupt, you can try re-activating the product, and then deactivating the product again.

If you require more activations because you cannot deactivate your serial number, you can request them on the Request Extra Activations page.

Upgrading

Minor releases are free for all users. For example, if you have a license for version 7.0 of a product, you can upgrade to version 7.1 at no cost. W hen you download and install a minor release, the product is licensed with your existing serial number automatically.

Major releases are free for users with a current Support and Upgrades contract. For example, if you have a license for version 7 of a product, you can upgrade to version 8 at no cost. When you download and install a major release, the product is licensed with your existing serial number automatically.

If you don't have a current Support and Upgrades contract, installing a major release will start a free 14-day trial. You'll need to buy a new license and activate the product with your new serial number.

To check whether you have a current Support and Upgrades contract or see the cost of upgrading to the latest major version of a product:

- visit the Upgrade Center
- email sales@red-gate.com
- call:
    - 1 866 733 4283 (toll free USA and Canada)
    - 0800 169 7433 (UK freephone)
    - +44 (0)870 160 0037 (rest of world)

To check the latest version of a product, see Current versions.

How to upgrade

You can download the latest version of a product using Check for Updates, the Upgrade Center, or the Redgate website.

- If you download the latest version from the Upgrade Center or our website, you need to run the installer to upgrade the product.

    Some Redgate products are available as part of bundle. You can select which products you want to upgrade when you run the installer.

- If you use Check for Updates, the installer runs automatically.

You can install the latest *major* version of any product (other than SQL Backup Pro) on the same machine as the previous version. For example, you can run version 9 and version 10 in parallel. However, installing a *minor* release will upgrade the existing installation.

To revert to an earlier version, uninstall the later version, then download and install the version you want from the Release notes and other versions page. You can use a serial number for a later version to activate an earlier version.

Upgrading to ANTS Memory Profiler 7

Summary

The new Summary in ANTS Memory Profiler 7 includes information which we think you will find more useful when trying to trace memory problems. This means that some of the information shown on the ANTS Memory Profiler 6 summary is no longer available, or has been moved.

The following list explains where to find information shown on the Summary in previous versions of ANTS Memory Profiler, in ANTS Memory Profiler 7:

Filters

The filter panel has been removed from the summary because filters have no effect on the summary (they didn't in version 6, either). The filter panel is available on other parts of the workflow, however.

Classes with largest size (in the current snapshot)

This information is available in the **Largest classes** pie chart.

Classes with most instances (in the current snapshot)

Switch to the **Class List** and then sort it by **Live Instances**.

Large object heap details

- **Free space on all .NET heaps:** On the Summary, in the .NET and unmanaged memory section, see Unused memory allocated to .NET
- **Largest free block:** On the Summary, in the Memory fragmentation section, see Largest fragment.
- **Max. size of new object (approx.):** This information is no longer available, because it is not generally useful.

Largest growth in size (comparison)

Switch to the **Class List**, and then sort it by **Size diff**.

Largest growth in instances (comparison)

Switch to the **Class List**, and then sort it by **Instance diff**.

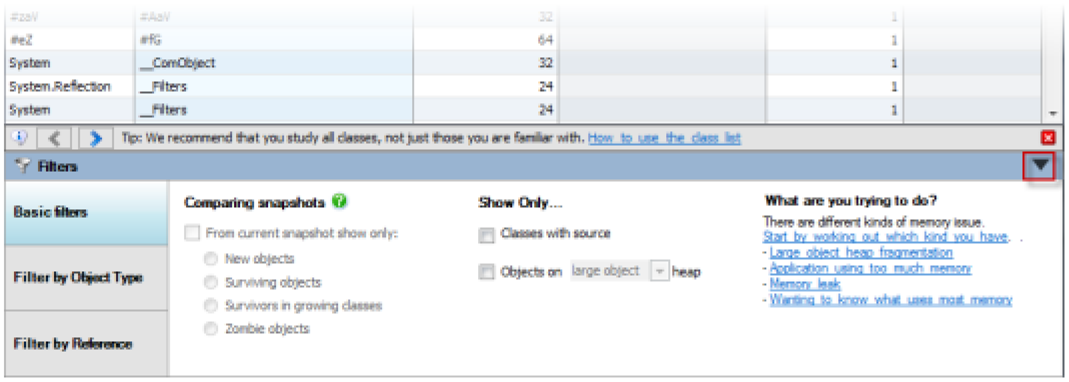
Large object heap changes

- **Change in free space:** This information is no longer available from ANTS Memory Profiler because it is rarely useful, but you can calculate it.
  1. Switch the current snapshot to an earlier snapshot (which you will use as baseline).
  2. On the **Summary**, in the **.NET and unmanaged memory** section, note the value given for **Unused memory allocated to .NET**.
  3. Switch to the later (comparison) snapshot.
  4. Subtract the **Unused memory allocated to .NET** from the first value.
- **Change in largest free block:** This information is no longer available from ANTS Memory Profiler because it is rarely useful, but you can calculate it.
  1. Switch the current snapshot to an earlier snapshot (which you will use as baseline).
  2. On the **Summary**, in the **Memory fragmentation** section, note the value given for **Largest fragment**.
  3. Switch to the later (comparison) snapshot.
  4. Subtract the **Largest fragment** from the first value.
- **Change in max. size of new object:** This information is no longer available, because it is not generally useful.

Filters

Filter panel

This is now shown at the bottom of the Class List, Instance Categorizer, and Instance List. The filter panel can be collapsed when not needed using the arrow.



#zaV	#AaV	32	1
#eZ	#fG	64	1
System	__ComObject	32	1
System.Reflection	__Filters	24	1
System	__Filters	24	1

Tip: We recommend that you study all classes, not just those you are familiar with. How to use the class list

Filters

Basic filters

Filter by Object Type

Filter by Reference

Comparing snapshots

- [ ] From current snapshot show only:
  - ○ New objects
  - ○ Surviving objects
  - ○ Survivors in growing classes
  - ○ Zombie objects

Show Only...

- [ ] Classes with source
- [ ] Objects on large object heap

What are you trying to do?

There are different kinds of memory issue.
Start by working out which kind you have.

- Large object heap fragmentation
- Application using too much memory
- Memory leak
- Wanting to know what uses most memory

Behavior when multiple filters are selected

In ANTS Memory Profiler 6 and earlier, when you enable more than one filter, there was normally an 'AND' relationship between most filters: only objects matching all of the filters were shown. There was one exception, however: when filtering by reference from more than one class or namespace, objects matching either filter were shown (an 'OR' relationship).

In ANTS Memory Profiler 7, there is an AND relationship between all filters. If you filter by reference from more than one class or namespace, only objects which are referenced by all of the selected classes/namespaces are shown.

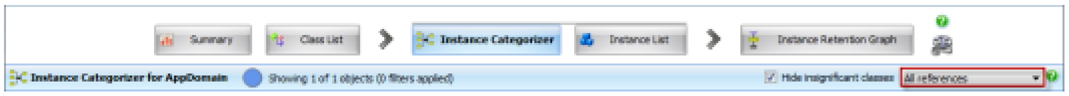
Process filter

For applications using more than one process, the **Process** filter is no longer on the Filters panel. On the Summary and the Class List, select the process you want to view by using the dropdown list beneath the timeline. The process selected by default is the process that started first.

Class Reference Explorer

The former Class Reference Explorer has been replaced in the workflow by the **Instance Categorizer**. The Instance Categorizer's default **Categorized references** view takes the objects which account for 60% of the memory usage by the class, and then categorizes them by the path which keeps those objects in memory. We hope that you will find this more useful.

If you would prefer to use the old Class Reference Explorer, switch the **Instance Categorizer** to **All references** view.



Instance Categorizer for AppDomain   Showing 1 of 1 objects (0 filters applied)   ✓ Hide insignificant classes   All references

Object Retention Graph

The Object Retention Graph is now called the **Instance Retention Graph**. This name change is only for technical accuracy. The only change in the Instance Retention Graph is that field values are now available in the graph.

WPF support

In ANTS Memory Profiler 6, a property that is set as a dependency property was not shown in the properties for the class it was associated with. In ANTS Memory Profiler 7, dependency properties are shown on the list of properties.

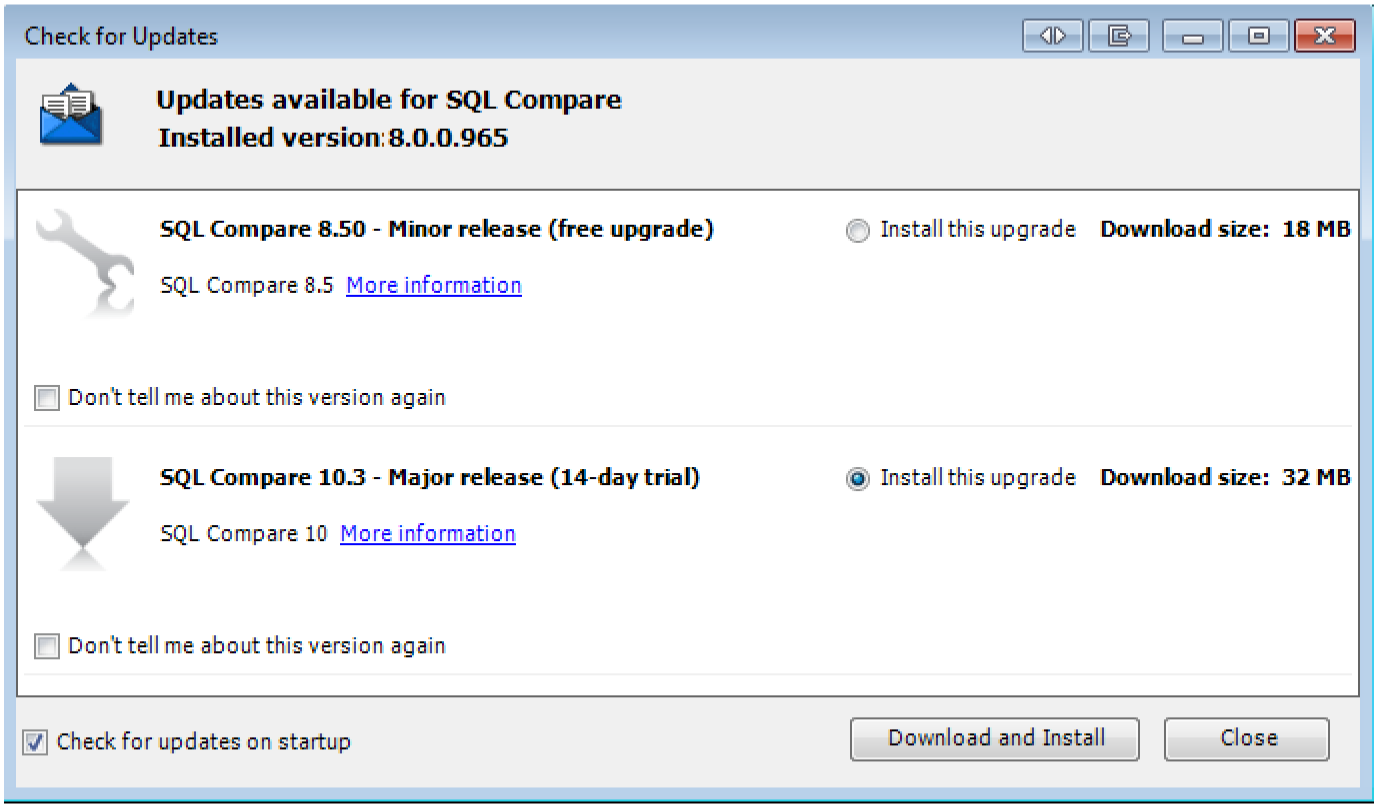
Using Check for Updates

This page applies to several Redgate products, so the screenshots below may not match your product.

The Check for Updates service checks whether a more recent version of the product is available to download. To use the service, your computer must have a connection to the internet. If your internet connection uses a proxy server, make sure your web browser connection settings are configured correctly.

The Check for Updates service doesn't work with automatic configuration scripts.

To check for updates for a Redgate product, on the **Help** menu, click **Check for Updates**. Any available updates are listed:


Check for Updates

Updates available for SQL Compare
Installed version: 8.0.0.965

SQL Compare 8.50 - Minor release (free upgrade) — Install this upgrade — Download size: 18 MB
SQL Compare 8.5 More information
Don't tell me about this version again

SQL Compare 10.3 - Major release (14-day trial) — [✓ Install this upgrade] — Download size: 32 MB
SQL Compare 10 More information
Don't tell me about this version again

[✓ Check for updates on startup]

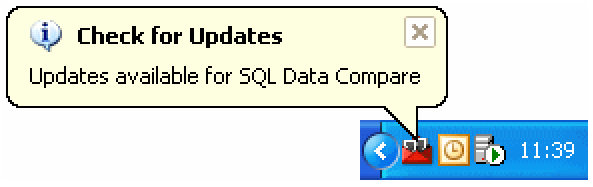
To view the full release details in your default web browser, click **More information**.

To get the update, click **Download and Install**. If you have a choice of updates, choose by selecting **Install this upgrade**, and then click **Download and Install**.

The installer will ask you to close the program. If you're upgrading an add-in, you'll also be asked to close the host program (SQL Server Management Studio, Visual Studio or Query Analyzer).

About the Check for Updates service

When you start the application, the Check for Updates service informs you automatically when there are updates available:


Check for Updates
Updates available for SQL Data Compare

If you don't want to receive these notifications for the product, clear the **Check for updates on startup** check box.

If you don't want the Check for Updates service to inform you about a particular update again, select the **Don't tell me about this version again** check box. The Check for Updates service will still inform you of new updates when they become available.

Troubleshooting Check for Updates errors

For details about how to use the Check for Updates service, see Using Check for Updates.

Error: There is a problem saving the download file to your computer

This error message is displayed if:

You don't have enough disk space

The Check for Updates service downloads the updates to the location defined by the *RGTEMP* environment variable, or the *TMP* variable if the *RGTEMP* variable doesn't exist.
If you don't have enough disk space, you can change the environment variable to a location with more space.

> Changing the *RGTEMP* or the *TMP* variables will affect other programs that use those variables. The *RGTEMP* variable affects only Redgate programs. For information about environment variables, see your Windows documentation.

There's a problem with permissions on your computer

The Check for Updates service downloads the updates to the location defined by the *RGTEMP* environment variable, or the *TMP* variable if the *RGTEMP* variable does not exist. If your user account doesn't have permissions to write to the location specified by these environment variables, contact your system administrator.

There's a problem with the download file on the Redgate web server

Contact Redgate support.

Error: There is a problem with the network connection

This error message is displayed if:

Your internet connection dropped while the Check for Updates service was downloading the updates

Try checking for updates again later.

Proxy authentication failed

Check your user name and password.

Your computer can't connect to the Check for Updates service.

Contact your system administrator. If you're using a proxy server, check it's configured correctly (see Control Panel > Internet Options > Connections).

> The Check for Updates service doesn't work with automatic configuration scripts.

There's a problem with the download file on the Redgate web server

Contact Redgate support.

Understanding memory problems

Most types of memory problem that you encounter are one of the following kinds. Working out which kind of problem you have may help you start to locate and resolve your problem.

- **Managed memory leaks**
  The typical symptom of a memory leak is that the performance degrades while the program runs; performance recovers on restart and then degrades again. The amount of bytes in the .NET heap increases.
- **Unmanaged object leaks**
  The performance degrades while the program runs; this recovers on restart and then degrades again. The number of private bytes (the amount of real and paged memory requested by the program) in use often increases at a greater rate than the number of bytes in the .NET heap.
- **Fragmentation of the large object heap**
  The main symptom of fragmentation is that *OutOfMemory* exceptions are thrown even though there is sufficient space on the .NET heap.
- **Large RAM footprint**
  Some programs need to use a lot of memory. There is little specific advice that can be given about this, since the best approach to reducing the amount of memory used will depend on your program's requirements. Using a profiler may give you suggestions for where to optimize code by showing which parts of your program are using the most memory.

If you are not familiar with memory profiling, you might find our .NET Memory Primer useful before you start.

Memory management primer

This page is an overview of .NET memory management. It should give you enough information about how .NET allocates objects into memory to start using ANTS Memory Profiler.

Large and small objects

.NET handles large objects and small objects differently:

- Large objects are all objects over 85k, and double (multidimensional) arrays over 8k.
- All other objects are considered small objects.

The small object heaps

Small objects are stored in the small object heaps.

Objects are allocated first to the generation 0 (Gen 0) heap.

When the Gen 0 heap is full, the .NET garbage collector runs. The garbage collector performs two tasks:

1. It disposes of all objects on the Gen 0 heap that are no longer needed.
2. It moves objects that are still needed to the generation 1 (Gen 1) heap.

This leaves the Gen 0 heap empty, ready for new objects to be allocated on it.

Eventually, the Gen 1 heap becomes full. When this happens, the garbage collector runs again, and it performs three tasks:

1. It disposes of all objects on the Gen 1 heap that are no longer needed.
2. It moves objects that are still needed to the generation 2 (Gen 2) heap.
3. It performs a garbage collection on Gen 0, as before.

If Gen 2 becomes full, a full garbage collection takes place. When this happens, the garbage collector performs four tasks:

1. It disposes of all objects on the Gen 2 heap that are no longer needed.
2. It performs a garbage collection on Gen 1, as before.
3. It performs a garbage collection on Gen 0, as before.
4. It disposes of objects on the large object heap that are no longer needed (see below).

If insufficient memory has been freed at the end of a full garbage collection, an OutOfMemory exception is thrown.

Note that whenever the garbage collector runs on any of the small object heaps, the heaps are compacted, so that the available space is used efficiently.

By using this generational approach, recently-allocated objects should be disposed sooner than older objects, which are presumed to be being kept in memory for a reason. It also ensures that full garbage collections, which have the greatest impact on an application's performance, happen as infrequently as possible.

ANTS Memory Profiler forces a garbage collection when you take a snapshot. This is because some of the memory counters that ANTS Memory Profiler obtains from Windows are only accurate immediately after a garbage collection. For this reason, you will generally not see any objects on the Gen 0 heap during profiling. This behavior will not affect the results that you obtain, because an object which only exists for a short time on Gen 0, and which is never promoted to Gen 1, will never be the cause of a .NET memory problem.

(If you do see objects on the Gen 0 heap, it is because the .NET garbage collector does not always behave in the way described in this simplified overview. For example, pinned objects, objects on the finalizer queue and objects created during the garbage collection might remain on the Gen 0 heap after a snapshot. Again, this is never a problem that would affect the results obtained.)

The large object heap

The large object heap is where large objects are stored.

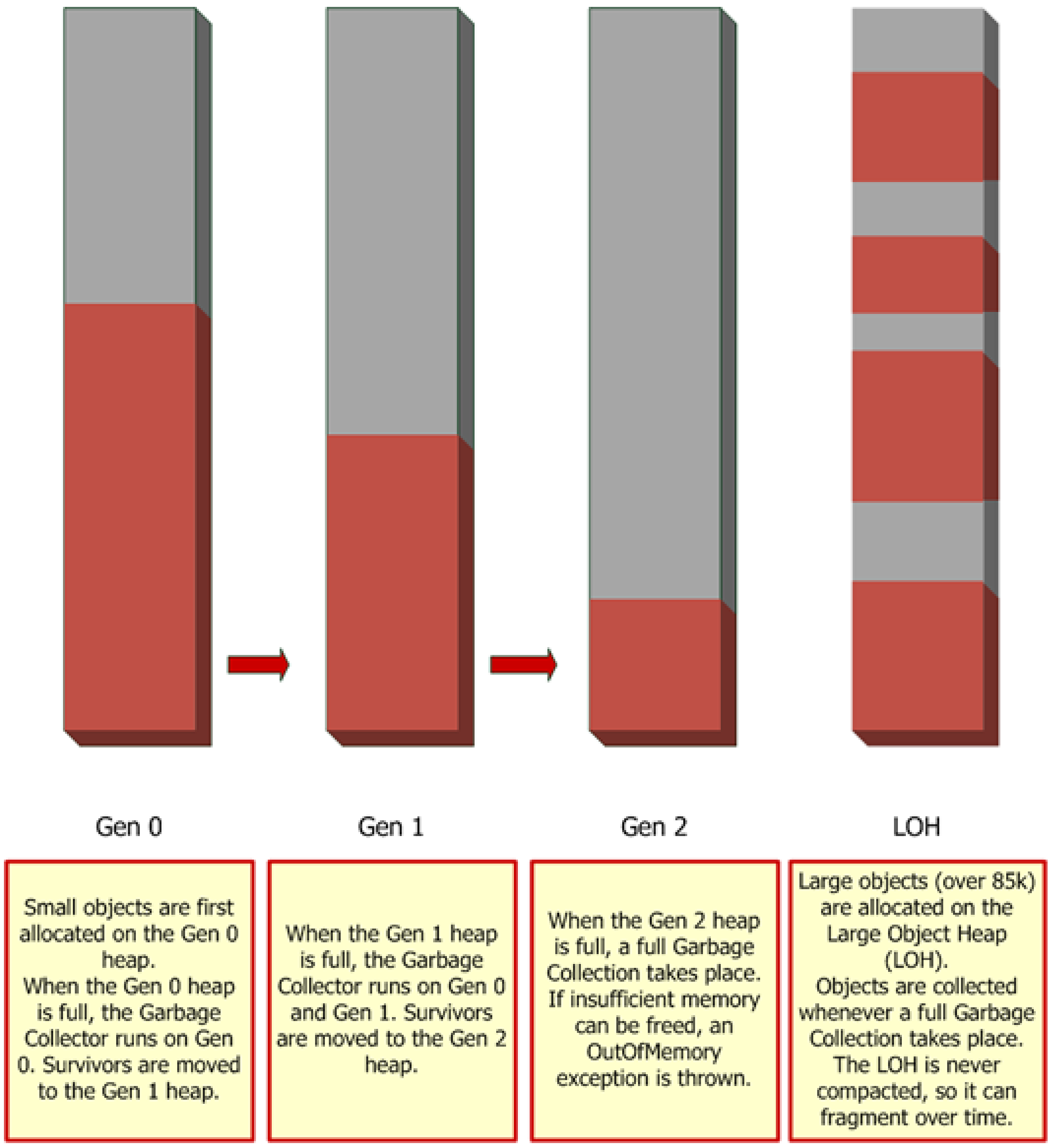
There is only one large object heap, and objects stored on it are only tidied-up during a full garbage collection.

Importantly, because compacting the large object heap would adversely affect your application's performance, the large object heap is never compacted. This means that the heap can fragment over time. If a new object, which is bigger than an existing fragment, needs to be allocated on the heap, the large object heap will grow so that the new object can be allocated to the end of the heap. Eventually, the fragmentation may mean that there is insufficient memory to allocate new large objects.

This is often a major cause of .NET memory problems, and it can be one of the most difficult issues to solve. For more information, see how to check for large object heap fragmentation.

Further information on large object heap fragmentation

- The dangers of the large object heap
  http://www.simple-talk.com/dotnet/.net-framework/the-dangers-of-the-large-object-heap/
- Large object heap uncovered (MSDN)
  http://msdn.microsoft.com/en-us/magazine/cc534993.aspx
- Understanding garbage collection in .NET
  http://www.simple-talk.com/dotnet/.net-framework/understanding-garbage-collection-in-.net/

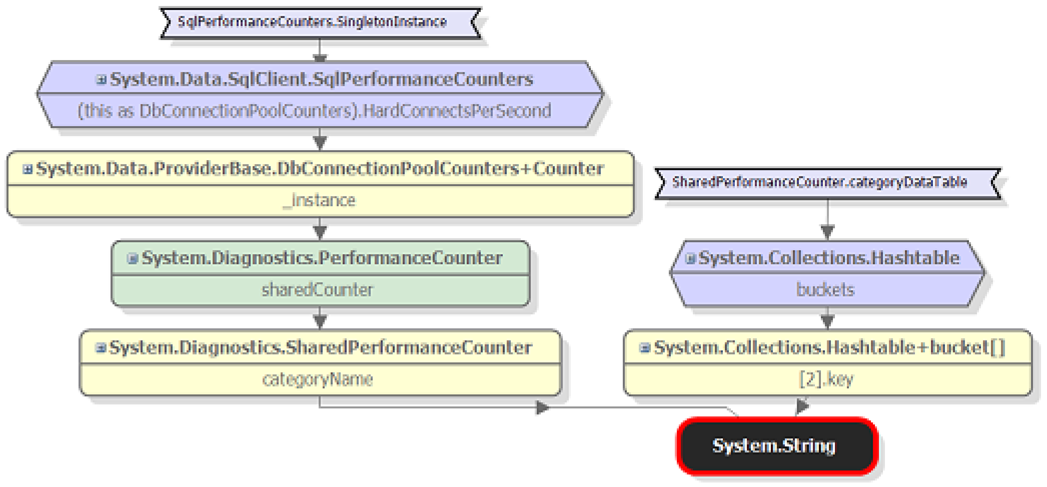

Gen 0
Gen 1
Gen 2
LOH
Small objects are first allocated on the Gen 0 heap. When the Gen 0 heap is full, the Garbage Collector runs on Gen 0. Survivors are moved to the Gen 1 heap.
When the Gen 1 heap is full, the Garbage Collector runs on Gen 0 and Gen 1. Survivors are moved to the Gen 2 heap.
When the Gen 2 heap is full, a full Garbage Collection takes place. If insufficient memory can be freed, an OutOfMemory exception is thrown.
Large objects (over 85k) are allocated on the Large Object Heap (LOH). Objects are collected whenever a full Garbage Collection takes place. The LOH is never compacted, so it can fragment over time.

Managed memory leaks

The garbage collector calculates whether or not an object can be disposed from one of the heaps by creating a graph (similar to the Instance Retention Graph in ANTS Memory Profiler), which shows how the object is referenced by other objects.

The graph starts from the GC roots. A GC root can be any storage slot to which the running program has access, such as a local variable, static variables, or even a CPU register. (Strictly speaking, the object itself is not the GC root; the storage slot that holds the reference to the object is the GC root.)

The GC roots will reference objects, which in turn typically reference other objects (through a member variable). If an object is not being referenced by anything, it cannot be reached by your program, and will be removed by the garbage collector.


SqlPerformanceCounters.SingletonInstance
System.Data.SqlClient.SqlPerformanceCounters
(this as DbConnectionPoolCounters).HardConnectsPerSecond
System.Data.ProviderBase.DbConnectionPoolCounters+Counter
_instance
System.Diagnostics.PerformanceCounter
sharedCounter
System.Diagnostics.SharedPerformanceCounter
categoryName
SharedPerformanceCounter.categoryDataTable
System.Collections.Hashtable
buckets
System.Collections.Hashtable+bucket[]
[2].key
System.String

In the graph above, `System.String` is referenced by:

1. The key of `System.Collections.Hashtable+bucket[2]`.
   a. `System.Collections.Hashtable+bucket[]` is referenced by the `buckets` property of `System.Collections.Hashtable`.
   b. `System.Collections.Hashtable` is a GC root because it is referenced by the static variable `SharedPerformanceCounter.categoryDataTable`.
2. The `categoryName` property of `System.Diagnostics.SharedPerformanceCounter`.
   a. `System.Diagnostics.SharedPerformanceCounter` is referenced by the `sharedCounter` property of `System.Diagnostics.PerformanceCounter`. (The green background means that `System.Diagnostics.PerformanceCounter` is disposable, but not yet disposed.)
   b. `System.Diagnostics.PerformanceCounter` is referenced by the `_instance` property of `System.Data.ProviderBase.DbConnectionPoolCounters+Counter`.
   c. `System.Data.ProviderBase.DbConnectionPoolCounters+Counter` is referenced by the `HardConnectsPerSecond` property of the base class `DbConnectionPoolCounters` in `System.Data.SqlClient.SqlPerformanceCounters`.
   d. `System.Data.SqlClient.SqlPerformanceCounters` is a GC root because it is referenced by the static variable `SqlPerformanceCounters.SingletonInstance`.

Because references hold objects in memory, you have to be careful to ensure that references to objects are removed when no longer needed. A common mistake, for example, is to leave an event handler (such as a timer) referencing an object. This will stop the garbage collector from disposing the object. The result is that an object which should have been in memory only briefly can end up leaked onto the Gen 2 heap. Importantly, in this scenario, all of the objects that the object references will also be on the Gen 2 heap. This has significant implications on how you use ANTS Memory Profiler: the objects referenced by the leaked object will often be more obvious than the leaked object itself, so you need to start by investigating those referenced objects (which are most commonly strings or byte arrays).

The typical symptom of a memory leak is that the performance degrades while the program runs but the amount of memory used recovers on restart and then degrades again.

See finding and fixing a memory leak.

Further information on finding leaks in managed code

- Identify and prevent memory leaks in managed code (MSDN)
- Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework (MSDN)
- Memory Management in .NET

Unmanaged memory leaks

Managed .NET code may interoperate with unmanaged code when some COM objects or native-code DLLs are invoked.

Leaks caused by unmanaged objects are more complex to identify, because ANTS Memory Profiler cannot provide detailed information about unmanaged usage. By understanding and interpreting the data shown by ANTS Memory Profiler, however, you might gain some ideas about where to look for these leaks.

The main indication of an unmanaged memory leak is when the number of private bytes (the amount of real and paged memory requested by the program) increases while the number of bytes in the .NET heap does not grow as quickly. You may be able to find the source of the problem by looking for .NET objects whose instance count is increasing, or by looking for objects on the finalizer queue that have not been disposed.

See checking for unmanaged memory usage.

Further information on finding leaks in unmanaged code

- Finding unmanaged memory leaks

Webinar

This page should have given you enough knowledge to start using ANTS Memory Profiler. If you are interested in learning more detail than this brief overview can provide, see our webinar 5 Misconceptions about .NET Memory Management.

Understanding ANTS Memory Profiler

Recommendations

Finding your memory problem may require some detective work; we recommend that, before using ANTS Memory Profiler, you should have the following:

1. Good knowledge of your code.
2. An idea of where in your code your problem might be occurring.
3. An understanding of how .NET memory management works.

If you have a leak in managed memory, it is likely that you are leaking more than one object:

- Your object, which is kept in memory because of this leak.
- Objects in classes from the CLR itself, which are referenced by your object.

If your leaked object contains text, for example, at least one System.String object will also be kept in memory. Your object may only be a few bytes large, but it may be holding a significant amount of memory in classes from the CLR.

Conversely, this means that when you are looking for leaked objects in your code using ANTS Memory Profiler, it is often useful to start by looking at these classes inherited from the CLR, rather than starting from your own classes.

Remember also that no program can decide whether or not an object is a leak, because it depends on the nature of the program you are profiling. If a large number of strings are kept in memory for a long time in word processing software, this may not be an indicator of a problem. However, if strings from a dialog box that has been closed are kept in memory in a graphics program, this could be a good place to start the investigation. Ultimately, therefore, you will need to use knowledge of your code to interpret the information shown in ANTS Memory Profiler.

Setting up and running a profiling session

When you start ANTS Memory Profiler, or when you start a new profiling session, the ANTS Memory Profiler Settings dialog box is displayed.

Application settings

On the **Application settings** tab, select the type of application you want to profile from the **Choose application type to profile** list. The settings available change depending on your selection:

- .NET executable
- ASP.NET web application (hosted in IIS or web development server)

  > Special requirements apply to SharePoint.

- Silverlight 4 or later browser application (ANTS Memory Profiler 7 only supports Silverlight 4)
- Windows service
- COM+ server
- XBAP (XAML Browser Application)
- Attach to a .NET 4 process

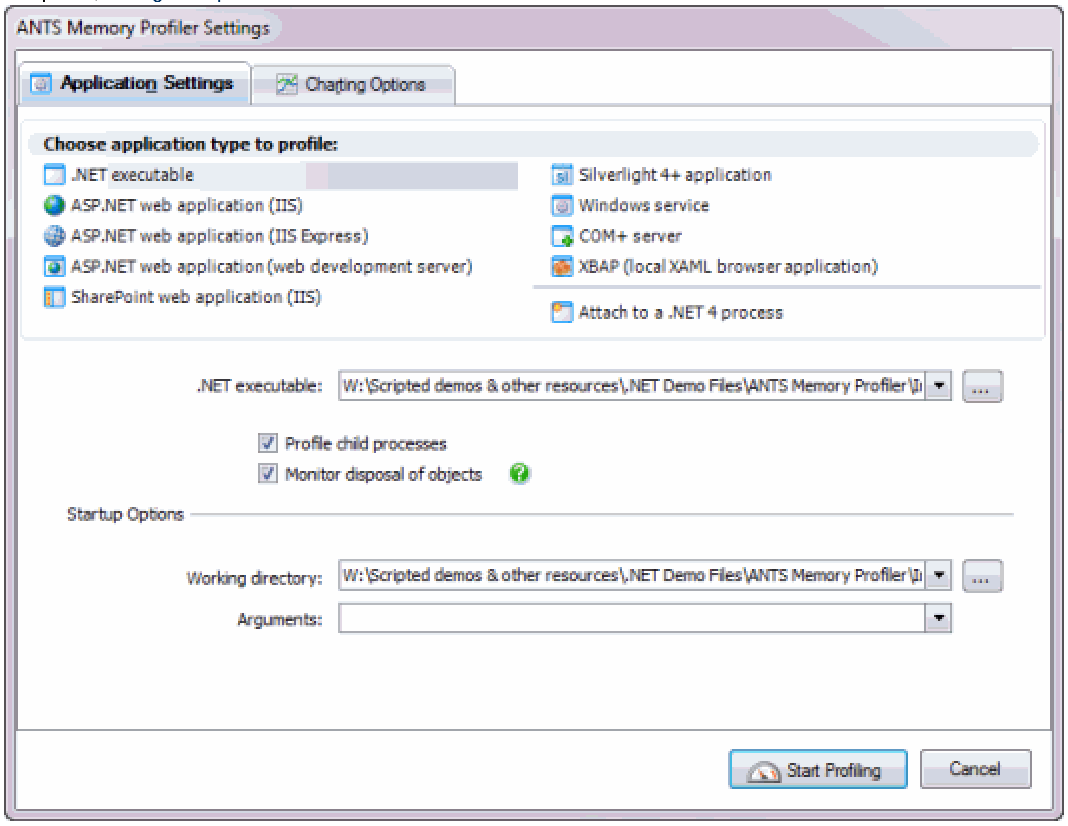
You can also start ANTS Memory Profiler from the Visual Studio add-in.

Charting options

Use the **Charting options** tab to choose which performance counters you would like to record. See Setting up performance counters for more information.

Profiling an executable

1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **.NET executable**.
3. Enter the path to the .NET executable that you want to profile.
4. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may reduce the performance of your application.
5. If required by your application, specify its **Working directory**. By default, the directory containing the .NET executable is used.
6. If required by your application, specify any **Arguments** that should be supplied to the .NET executable at launch.
7. If required, change the performance counters to record.


ANTS Memory Profiler Settings
Application Settings
Charting Options
Choose application type to profile:
.NET executable
ASP.NET web application (IIS)
ASP.NET web application (IIS Express)
ASP.NET web application (web development server)
SharePoint web application (IIS)
Silverlight 4+ application
Windows service
COM+ server
XBAP (local XAML browser application)
Attach to a .NET 4 process
Start Profiling
Cancel

.NET executable: W:\Scripted demos & other resources\.NET Demo Files\ANTS Memory Profiler\

[✓] Profile child processes
[✓] Monitor disposal of objects

Startup Options

Working directory: W:\Scripted demos & other resources\.NET Demo Files\ANTS Memory Profiler\

Arguments:

8. Click **Start Profiling**.
9. Check whether there are any memory problems.

Profiling a web application

ANTS Memory Profiler can profile ASP.NET web applications. The application you want to profile must be on the same machine on which you are running the profiler.

Web applications are profiled in Microsoft Internet Explorer, even if that is not your preferred browser. This is because ANTS Memory Profiler uses the low-level data exposed by Internet Explorer.

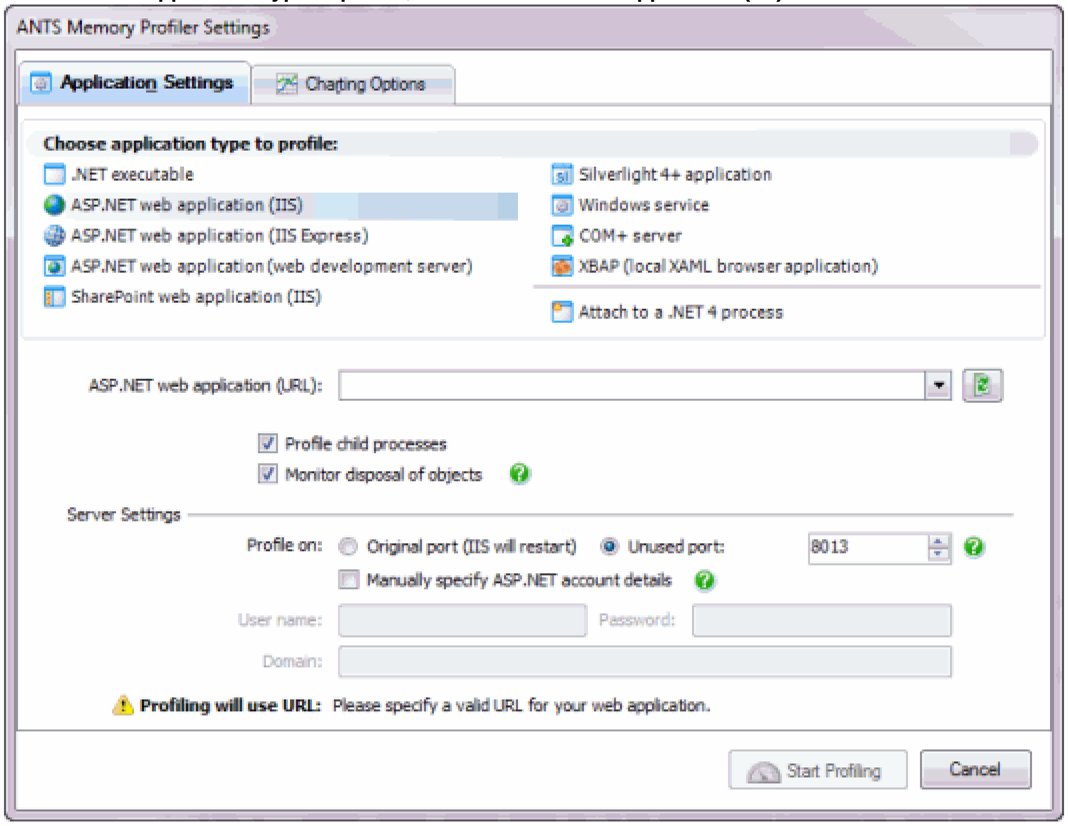
The set-up steps depend on what you are profiling:

- an ASP.NET application in IIS
- an ASP.NET application in IIS Express
- an ASP.NET application in web development server
- SharePoint

Profiling on IIS

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

1. Start ANTS Memory Profiler. If it's already running, on the **File** menu, click **New Profiling Session**.
2. Under **Choose application type to profile**, select **ASP.NET web application (IIS)**.



ANTS Memory Profiler Settings

Application Settings | Charting Options

Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (IIS Express)
- ASP.NET web application (web development server)
- SharePoint web application (IIS)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

ASP.NET web application (URL):

[✓] Profile child processes
[✓] Monitor disposal of objects

Server Settings

Profile on: ( ) Original port (IIS will restart) (●) Unused port: 8013

[ ] Manually specify ASP.NET account details

User name: Password:

Domain:

Profiling will use URL: Please specify a valid URL for your web application.

Start Profiling | Cancel

3. Next to **ASP.NET web application (URL)**, click the

   refresh icon.
   ANTS Memory Profiler displays a list of applications currently running on IIS. Choose the application you want to profile.
4. Normally, you'll want to keep **Profile child processes** and **Monitor disposal of objects** selected, but this can affect your application's performance.
5. If needed, choose an **Unused port** to profile on.
   This means that IIS won't need to restart, but it won't work if you've hard-coded a specific port in your web application.
   (This option is not available in IIS 5.)
6. With IIS 6 and 7, your web application will run under the Windows Local System user by default.
   If needed, change the user account your application will run under by selecting **Manually specify ASP.NET account details**. The specified user must have administrator privileges and permission to read from *%ProgramFiles%\Red Gate\ANTS Memory Profiler 7\RedGate.Memory.Core.dll*
   With IIS 5, your web application will always run under the ASPNET account. Make sure that the ASPNET account has permission to read from *%ProgramFiles%\Red Gate\ANTS Memory Profiler 7\RedGate.Memory.Core.dll*
7. The **Profiling will use URL** notice confirms the URL that will be used for profiling, including any unused port you have set.
8. If needed, change the performance counters to record.
9. Click

   Start Profiling

   .
10. Check whether there are any memory problems.

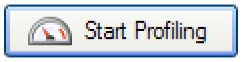
Profiling on IIS Express

This page only applies to ANTS Memory Profiler 7.3 and later.

To profile ASP.NET applications running on IIS Express:

1. Start ANTS Memory Profiler. If it's already running, on the **File** menu, click **New Profiling Session**.
2. Under **Choose application type to profile**, select **ASP.NET web application (IIS Express)**.
3. Set the **Web application path or config file** for the web application that you want to profile.You can specify the path directly to the web application, or to the site's ApplicationHost.config file.
   If you choose the ApplicationHost.config file, a list of available applications configured in that file is displayed. Select the application you want to profile.

   Normally, you'll want to keep **Profile child processes** and **Monitor disposal of objects** selected. However, monitoring the disposal of objects can affect the performance of your application.

4. If needed, Setting up performance counters.
5. Click
   

   .
   IIS Express starts, and the web application opens in Internet Explorer.

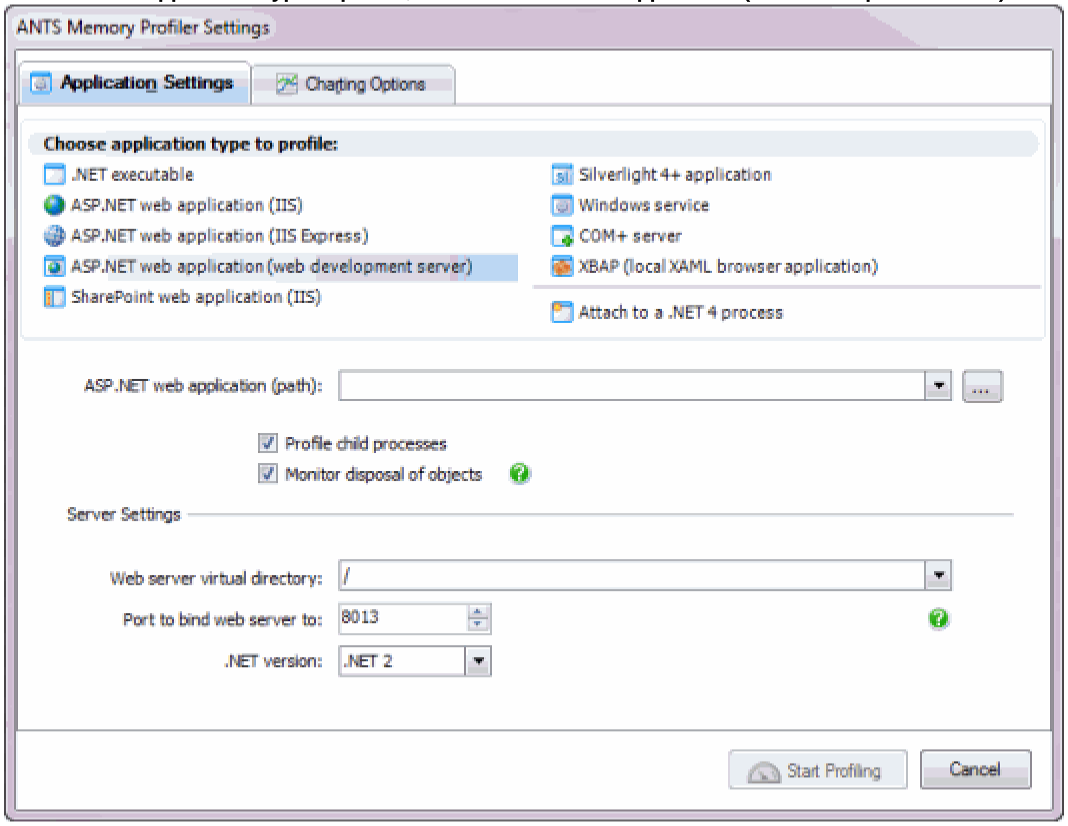
   You can interact with the application using any web browser, but closing the Internet Explorer window opened by ANTS Memory Profiler will end your profiling session.

6. Check whether there are any memory problems.

Profiling on web development server

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

1. Start ANTS Memory Profiler. If it's already running, on the **File** menu, click **New Profiling Session**.
2. Under **Choose application type to profile**, select **ASP.NET web application (web development server)**.



ANTS Memory Profiler Settings

Choose application type to profile:
- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (IIS Express)
- ASP.NET web application (web development server)
- SharePoint web application (IIS)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

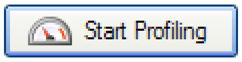
ASP.NET web application (path):

☑ Profile child processes
☑ Monitor disposal of objects

Server Settings

Web server virtual directory: /
Port to bind web server to: 8013
.NET version: .NET 2

3. In the **ASP.NET web application (path),** browse to your application.
4. Normally, you'll want to keep **Profile child processes** and **Monitor disposal of objects** selected, but this can affect your application's performance.
5. In the **Web server virtual directory** box, enter the virtual directory where your application will start.
6. In the **Port to bind web server to** box, select the port where your application will start.
   For example, if the virtual directory is *staging* and the port is *8013*, the URL profiled will be *http://localhost:8013/staging/*
7. Specify the .NET version which the web application is compiled against.
8. Click



.
9. Check whether there are any memory problems.

Profiling SharePoint

ANTS Memory Profiler can profile managed code that runs on a Microsoft SharePoint 2007 or 2010 server.

- ANTS Memory Profiler 7.3 and later
- ANTS Memory Profiler 7.2 and earlier
- Troubleshooting

ANTS Memory Profiler 7.3 and later

To profile SharePoint:

1. Start ANTS Memory Profiler. If it's already running, on the **File** menu, click **New Profiling Session**.
2. Under **Choose application type to profile**, select**SharePoint web application (IIS)**.
3. Click
   to load a list of SharePoint sites that are currently running into the dropdown.
4. Choose the site that you want to profile.
5. Normally, you'll want to keep **Profile child processes** and **Monitor disposal of objects** selected, but this can affect your application's performance.
6. Choose the port to profile your application on:
   - If your application doesn't bind to a specific port, select **Unused port** and choose a port that is not used by IIS.
   - If your application must use the port it currently runs on, select **Original port (IIS will restart)**.

   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 Memory Profiler Settings dialog box.
   If needed, select **Manually specify ASP.NET account details** and enter the User name, Password, and Domain.

   > ANTS Memory Profiler profiles your web application as the Windows Local System user. If the Windows Local System user doesn't have the permissions to use your SharePoint site collection, enter the credentials of a user with the required permissions. This user must also be an administrator on the computer ANTS Memory Profiler runs on.
   >
   > With IIS 5, your web application will always run under the ASPNET account. Make sure that the ASPNET account has permission to read from *%ProgramFiles%\Red Gate\ANTS Memory Profiler 7\RedGate.Memory.Core.dll*

   The **Profiling will use URL** notice confirms the URL that will be used for profiling, including any unused port you have set.
7. If needed, change the performance counters to record.
8. Click
   Start Profiling
   .
9. Check whether there are any memory problems.

ANTS Memory Profiler 7.2 and earlier

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

Though ANTS Memory Profiler was not designed to support SharePoint 2010 profiling, a workaround exists that may let you profile managed code running on a SharePoint 2010 server. For details, see Profiling SharePoint 2010.

To profile Sharepoint 2007, follow the procedure for profiling an ASP.NET web application on IIS.

If you encounter difficulties, the most likely cause is that ANTS Memory Profiler cannot read from the directory to which SharePoint is writing data. 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 Profiling SharePoint with ANTS Performance Profiler 5.2 (Simple Talk).

Troubleshooting

If you experience problems, see Troubleshooting SharePoint profiling.

SharePoint's API is still partly unmanaged; if you have been experiencing memory problems with SharePoint, you might find the following MSDN articles helpful:

- Common Coding Issues When Using the SharePoint Object Model
- Using disposable Windows SharePoint Services objects

Profiling SharePoint 2010

This page only applies to ANTS Memory Profiler 7.2 and earlier.

ANTS Memory Profiler 7.2 and earlier do not support profiling SharePoint 2010, but this workaround may let you profile managed code running on a SharePoint 2010 server with IIS 7.

To profile SharePoint 2010 with IIS 7:

1. Open the IIS applicationHost configuration file.
   By default, this is located in *C:\Windows\system32\inetsrv\config\applicationHost.config*
2. In the <globalModules> element, comment out the following line:
   ```
   <add name="SharePoint14Module" image="C:\Program Files\Common Files\Microsoft Shared\Web Server
   Extensions\14\isapi\owssvr.dll" preCondition="appPoolName=SharePoint Central Administration
      v4;SharePoint - 80" />
   ```
3. Immediately below this line, add the following line:
   ```
   <add name="SharePoint14Module" image="C:\Program Files\Common Files\Microsoft Shared\Web Server
   Extensions\14\isapi\owssvr.dll" preCondition="appPoolName=RGTestAppPool;SharePoint - 80" />
   ```

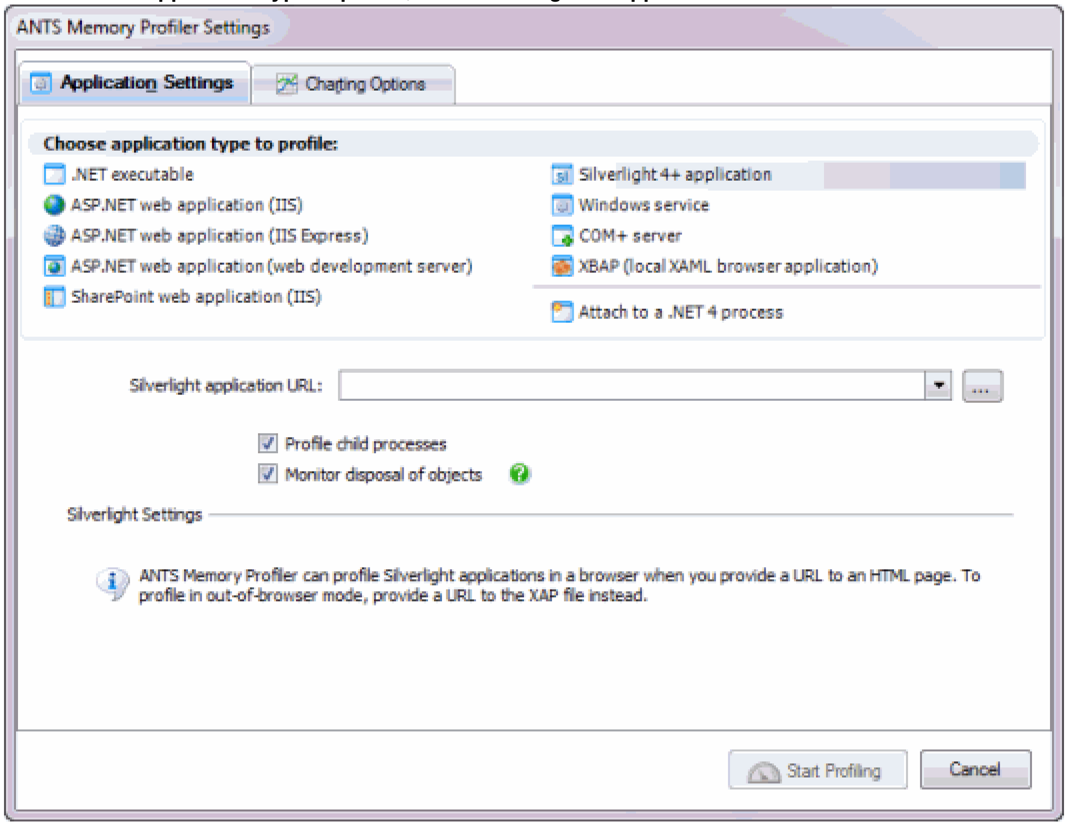
   This sets a precondition allowing ANTS Memory Profiler to load the SharePoint server DLL. ANTS Memory Profiler will now be able to profile code on the Sharepoint server.
4. Follow the instructions on Profiling SharePoint.

Profiling a Silverlight application

You can only profile Silverlight 4 or later applications. Profiling Silverlight applications requires the Silverlight 4 (or later) Internet Explorer plugin.

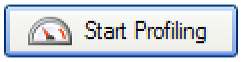
ANTS Memory Profiler 7 only supports Silverlight 4 applications.

To profile a Silverlight application:

1. If your application runs in a browser, use Task Manager to kill all running *iexplore.exe* processes.
   This makes sure that ANTS Memory Profiler attaches to the right *iexplore.exe* process.
2. Start ANTS Memory Profiler. If it's already running, on the **File** menu, click **New Profiling Session**.
3. Under **Choose application type to profile**, select **Silverlight 4+ application**.
   
   ANTS Memory Profiler Settings
Application Settings
Charting Options
Choose application type to profile:
.NET executable
ASP.NET web application (IIS)
ASP.NET web application (IIS Express)
ASP.NET web application (web development server)
SharePoint web application (IIS)
Silverlight 4+ application
Windows service
COM+ server
XBAP (local XAML browser application)
Attach to a .NET 4 process
Silverlight application URL:
Profile child processes
Monitor disposal of objects
Silverlight Settings
Start Profiling
Cancel
   ANTS Memory Profiler can profile Silverlight applications in a browser when you provide a URL to an HTML page. To profile in out-of-browser mode, provide a URL to the XAP file instead.
4. If your application runs in out-of-browser mode, next to **Silverlight application URL**, enter:
   - A local path to the XAP file
   - A URL (pointing to localhost) of the XAP file

   If your application runs in a browser, next to **Silverlight application URL**, enter:
   - A local path to the file that embeds the application
   - A URL (pointing to localhost) of the file which embeds the application

   ANTS Memory Profiler 7.4 only
   When profiling Silverlight browser applications, you cannot navigate to your source code using the Visual Studio add-in. See below for how to configure your application and profiling session to work with the add-in.

5. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may affect the performance of your application.
6. If required, change the performance counters to record.
7. Click
   

   .
8. Check whether there are any memory problems.

If you have problems, see Silverlight out-of-browser profiling stops with no results or Silverlight in-browser profiling stops with no results.

Navigating to Silverlight source code

This section only applies to ANTS Memory Profiler 7.4.

If you profile a Silverlight application that runs in a web browser, you cannot switch to view your source code using the ANTS Performance Profiler Visual Studio Add-in. To navigate to your source code, configure your Silverlight application to run in out-of-browser mode, and profile the Silverlight launcher application:

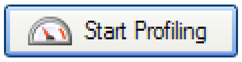
1. In Visual Studio, open the application's Project Settings and select **Enable running application out of browser**.
2. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**.
3. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **.NET executable**.
4. In **.NET executable**, set the path to *sllauncher.exe* - e.g. *C:\Program Files (x86)\Microsoft Silverlight\sllauncher.exe*
5. In **Startup Options**, set the **Working directory** to the directory containing your Silverlight application.
6. In **Arguments**, set the */emulate* and */origin*switches to the path of the Silverlight application - e.g.

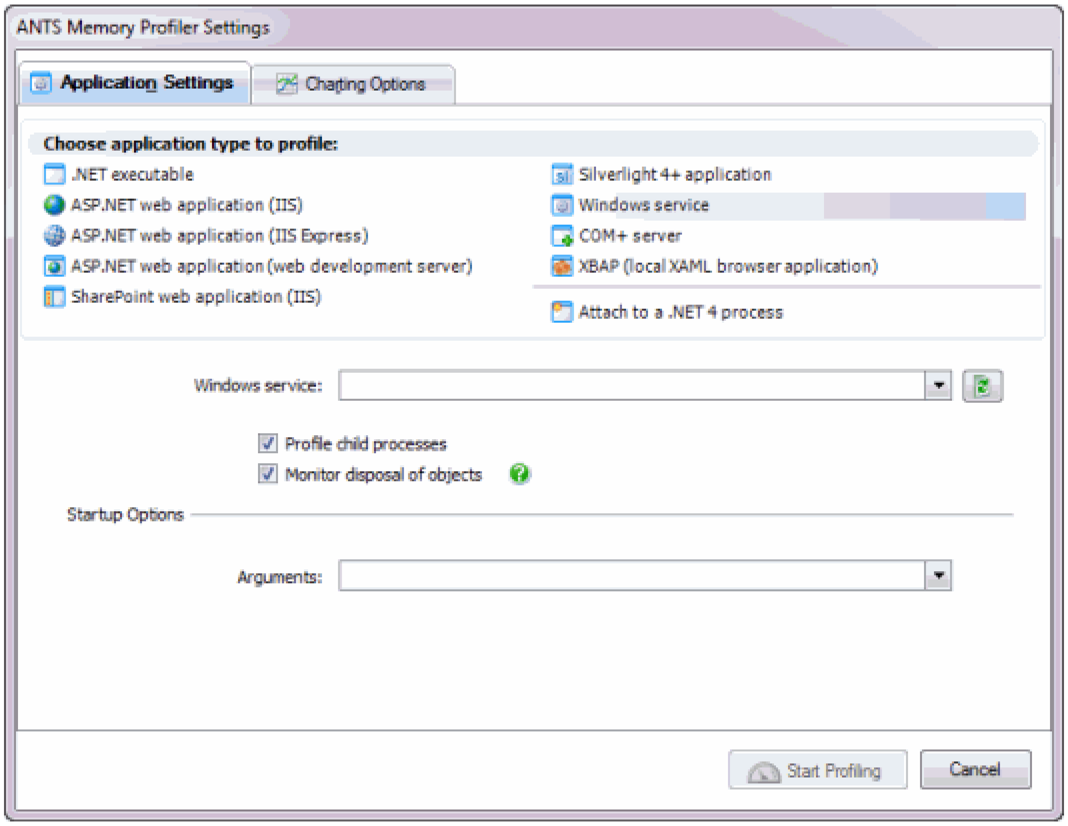
```
/emulate:"D:\Dev\SilverlightApplication1\BinRelease\SilverlightApplication1.xap"

/origin:"D:\Dev\SilverlightApplication1\Bin\Release\SilverlightApplication1.xap"
```

7. Click **Start Profiling**.

A new profiling session starts, and you can navigate to your source code from profiling results, using the Visual Studio add-in.

Profiling a Windows service

1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **Windows service**.
3. Next to **Windows service:** choose the windows service from the dropdown list. Click the icon to refresh the list, for example, if you have started the service since the ANTS Memory Profiler Settings dialog box was displayed.
4. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may impact the performance of your application.
5. If required, specify any **Arguments** required by the service.
6. If required, change the performance counters to record.
7. Click 
   .
8. Check whether there are any memory problems.


ANTS Memory Profiler Settings

Application Settings | Charting Options

Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (IIS Express)
- ASP.NET web application (web development server)
- SharePoint web application (IIS)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

Windows service:

☑ Profile child processes
☑ Monitor disposal of objects

Startup Options

Arguments:

Start Profiling | Cancel

If you experience problems, see Profiling a Windows service fails if the service uses a system account.

Profiling COMplus servers

Before you start

To profile a COM+ server application correctly, 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), or if you set it to `ActivationOption.Library`, 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
- will be treated by ANTS Memory Profiler as a DLL invoked by the client.

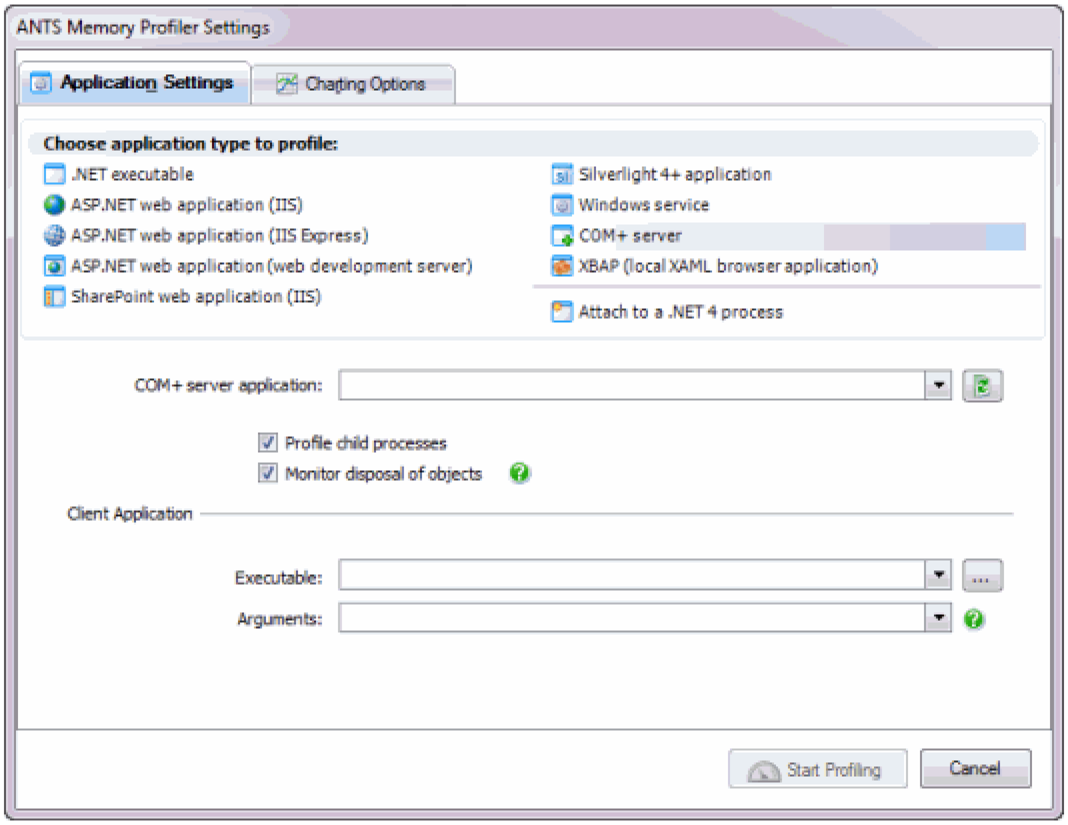
Also, you may need to make the application's code fully trusted by setting the `ApplicationAccessControl` attribute as follows:

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

You should not normally release the COM+ application in this trusted state.

Profiler settings

1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **COM+ Server**.
3. Next to **COM+ server application** choose the COM+ server application from the dropdown list. Click the icon to refresh the list, for example, if you have started the server since the ANTS Memory Profiler Settings dialog box was displayed.
4. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may impact the performance of your application.
5. Under **Client application**, next to **Executable**, choose the path to the client application.
6. If required, specify any **Arguments** required by the client application.
7. If required, change the performance counters to record.
8. Click Start Profiling .
9. Use the client application to test the COM+ server application during your profiling session.
10. Check whether there are any memory problems.


ANTS Memory Profiler Settings

Application Settings | Charting Options

Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (IIS Express)
- ASP.NET web application (web development server)
- SharePoint web application (IIS)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

COM+ server application:

[✓] Profile child processes

[✓] Monitor disposal of objects

Client Application

Executable:

Arguments:

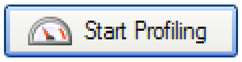
Start Profiling | Cancel

Profiling XBAPs

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

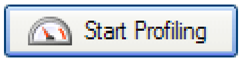
To profile an XBAP application, Internet Explorer must be set as your default browser and it must be closed before profiling.

To profile a locally-hosted XBAP application

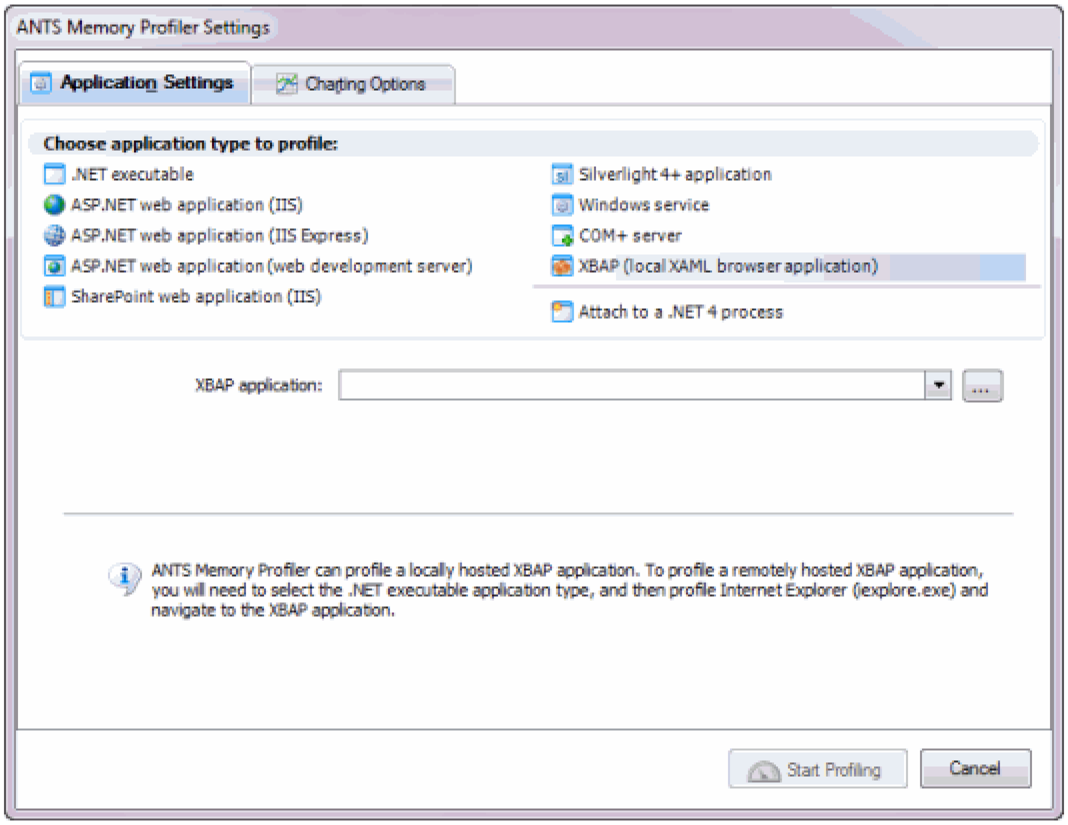
1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **XBAP (local XAML browser application)**.
3. Enter the path to the XBAP application that you want to profile.
4. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may impact the performance of your application.
5. If required, change the performance counters to record.
6. Click
   

   .
7. Check whether there are any memory problems.

To profile a remote XBAP application

1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **.NET executable**.
3. Enter the path to Microsoft Internet Explorer (*iexplore.exe*).
4. You should normally leave **Profile child processes** and **Monitor disposal of objects** selected, although monitoring the disposal of objects may impact the performance of your application.
5. If required by your application, specify its **Working directory**. By default, the directory containing the .NET executable is used.
6. If required by your application, specify any **Arguments** which should be supplied to the .NET executable at launch.
7. If required, change the performance counters to record.
8. Click
   

   .
9. When Internet Explorer starts, navigate to the XBAP application and interact with it in the normal way.
10. Check whether there are any memory problems.



Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (IIS Express)
- ASP.NET web application (web development server)
- SharePoint web application (IIS)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

XBAP application:

ANTS Memory Profiler can profile a locally hosted XBAP application. To profile a remotely hosted XBAP application, you will need to select the .NET executable application type, and then profile Internet Explorer (iexplore.exe) and navigate to the XBAP application.

Attaching to a process

If your application is not a .NET 4 application

Due to a limitation in previous versions of the CLR, you can only attach ANTS Memory Profiler to a .NET 4 process.

If your process runs on an older version of .NET, however, it is often possible to force it to run as .NET 4 without recompiling it. For more information, see Forcing your application to use .NET 4.

Before you start

Before you can attach ANTS Memory Profiler to a running .NET 4 process, you must first disable concurrent garbage collection in your application. This allows ANTS Memory Profiler to monitor garbage collections, which it cannot do if garbage collections take place in a different thread.

Disabling concurrent garbage collection means that garbage collection will take place in the same thread as the application's thread. Note: This may affect the performance of your application; see the notes below.

For more information on concurrent garbage collection, see Fundamentals of garbage collection: concurrent garbage collection (MSDN) .

To disable concurrent garbage collection, follow the steps below:

- While concurrent garbage collections are disabled, your application will run faster but may pause occasionally to allow garbage collections to take place.
- While concurrent garbage collections are disabled, the total memory footprint of your application is fractionally increased. This is because the concurrent garbage collector would normally build a graph of unreachable objects on the other thread. This difference should not be significant enough to alter the results shown by ANTS Memory Profiler.
- You cannot attach to a .NET 4 process which is running in server garbage collection mode. If your configuration file contains `<gcServer enabled="true"/>`, delete this node.

Disabling concurrent garbage collection (If you are not attaching to a webservice running on the web development server):

1. Create a new file, and copy the following XML into it:

```
<configuration>
   <runtime>
      <gcConcurrent enabled="false"/>
   </runtime>
</configuration>
```

2. Save this file in the same directory as the application's executable file with the same name as the executable file, but appending a .config extension. (For example, if your application is *sample.exe*, the configuration file must be named *sample.exe.config*).
   It is not necessary to rebuild your executable. The .NET CLR will automatically load the configuration file at runtime.
3. Run your executable and then attach ANTS Memory Profiler to it (see Profiler settings).

When you have finished profiling, delete the configuration file.

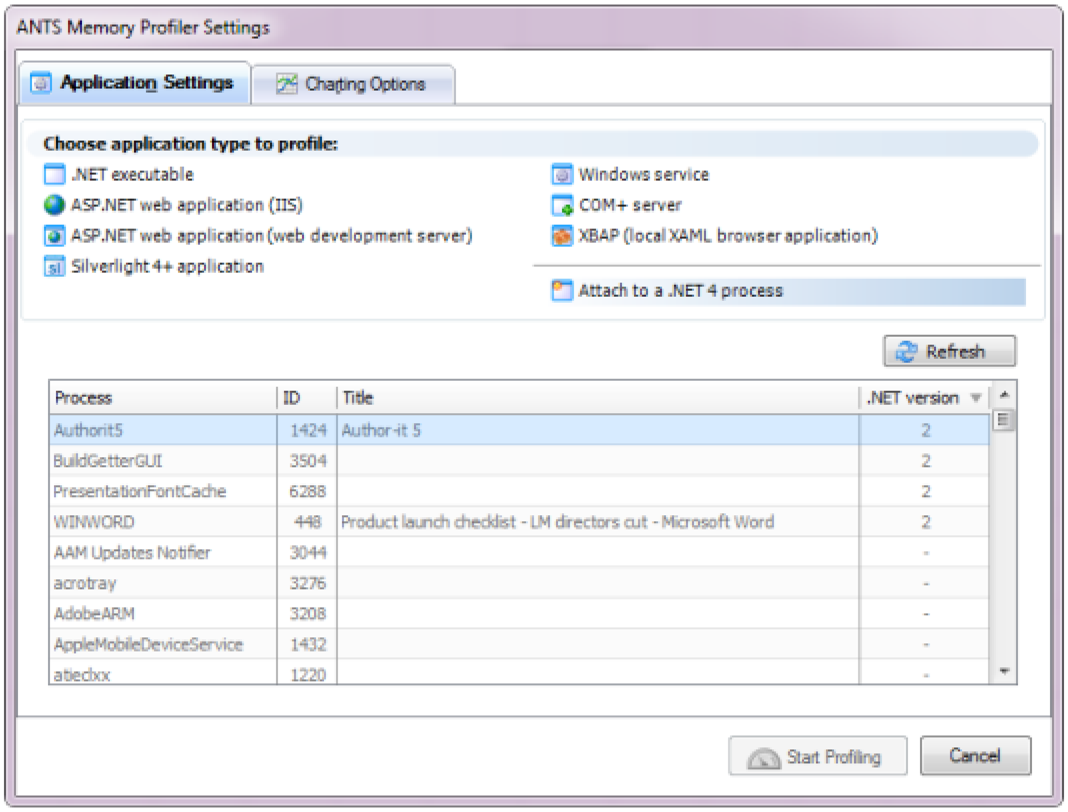
Disabling concurrent garbage collection (If you are attaching to a webservice running on the web development server):

1. Open the *WebDev.WebServer.exe.config* file, located at *%ProgramFiles%/Common Files/microsoft shared/Dev Server/*, in a text editor or XML editor.
2. Under the `<runtime>` node, insert the following text:
   `<gcConcurrent enabled="false"/>`
3. Save the WebDev config file.

When you have finished profiling, delete the `<gcConcurrent>` node.

Profiler settings

1. Start ANTS Memory Profiler. If it is already running, on the **File** menu, click **New Profiling Session**...
2. On the ANTS Memory Profiler Settings dialog box, on the **Application Settings** tab, select **Attach to a .NET 4 process**.
3. Processes running under the same user profile as ANTS Memory Profiler are shown in a list. Processes which are not .NET 4 processes are unavailable. To show processes from other users select **Show processes from all users**. To refresh the list, click
   Refresh
   .
4. Select the process that you want to profile.
5. If required, change the performance counters to record.
6. Click
   Start Profiling
   .
7. Check whether there are any memory problems.



Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS)
- ASP.NET web application (web development server)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

Process	ID	Title	.NET version
Authorit5	1424	Author-it 5	2
BuildGetterGUI	3504		2
PresentationFontCache	6288		2
WINWORD	448	Product launch checklist - LM directors cut - Microsoft Word	2
AAM Updates Notifier	3044		-
acrotray	3276		-
AdobeARM	3208		-
AppleMobileDeviceService	1432		-
atiedxx	1220		-

If you experience problems, see Attach to process unavailable with some anti-virus software.

Forcing your application to use .NET 4

To use the 'attach to process' feature in ANTS Memory Profiler 6 and above, your assembly must use the .NET 4 CLR.

You can change the CLR version that your application uses without rebuilding it. This allows you to use 'attach to process' on applications compiled against previous versions.

This will only work if you have not hard-coded a reference to a specific version of .NET in your code.

To force your application to use .NET 4:

1. Copy the following XML to a new file:

```
<configuration>
  <startup>
    <supportedRuntime version="v4.0"/>
  </startup>
</configuration>
```

2. Save the file in the same location as the application's *.exe* file, adding *.config* to the *.exe*'s file name.
   For example, if your application is saved at *C:/Program Files/Company/Program/myApplication.exe*, you must save the XML file as *C:/Program Files/Company/Program/myApplication.exe.config*.
3. Run your application as normal. .NET will automatically load the *.exe.config* file and the application will run against .NET 4.
4. While the application is running, you can attach the profilers to it, as for a normal .NET 4 application.

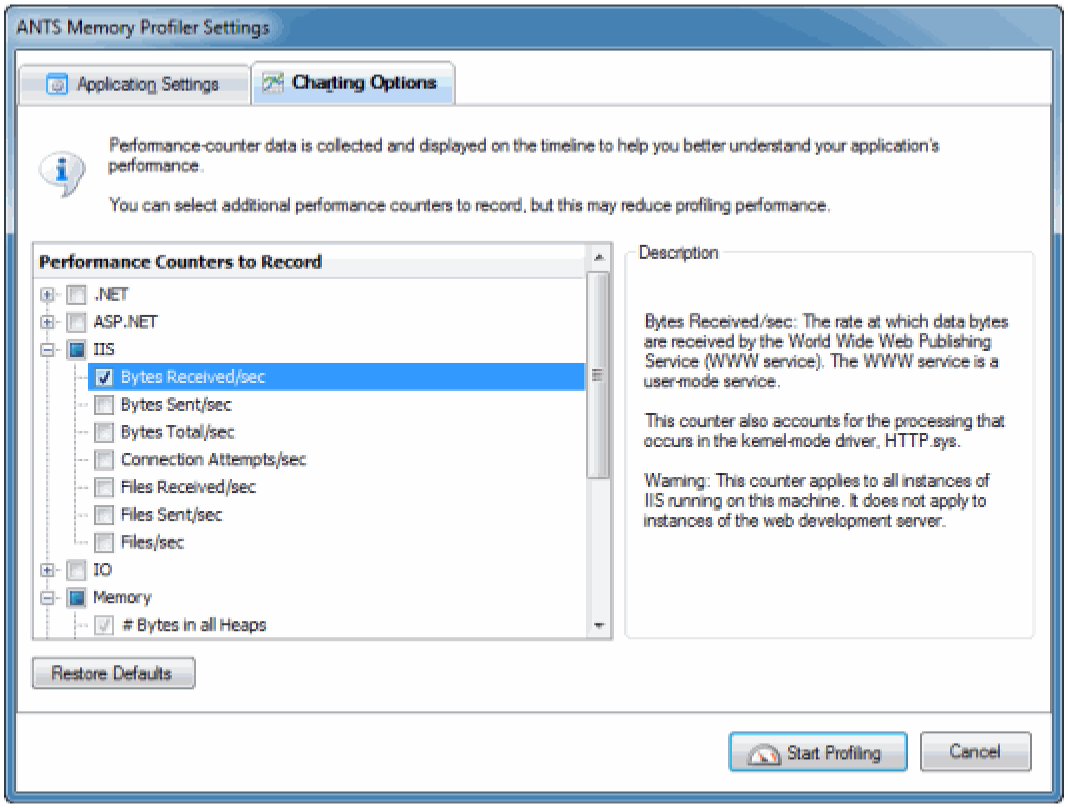
If the **.exe.config* file already exists, edit it with an XML editor to add the `<supportedRuntime>` node.

Setting up performance counters

ANTS Memory Profiler can monitor the values of a number of Windows performance counters while the application you are profiling is executing. The values of these counters are constantly updated on the timeline during profiling.

We recommend using only the default counters. Each additional counter you monitor adds to the overhead introduced by the profiler. Adding too many counters may cause your application to run substantially more slowly.

Choose the performance counters you want to monitor using the **Charting Options** tab on the **ANTS Memory Profiler Settings** dialog box.



Performance-counter data is collected and displayed on the timeline to help you better understand your application's performance.

You can select additional performance counters to record, but this may reduce profiling performance.

Performance Counters to Record

- .NET
- ASP.NET
- IIS
  - ✓ Bytes Received/sec
  - Bytes Sent/sec
  - Bytes Total/sec
  - Connection Attempts/sec
  - Files Received/sec
  - Files Sent/sec
  - Files/sec
- IO
- Memory
  - ✓ # Bytes in all Heaps

Description

Bytes Received/sec: The rate at which data bytes are received by the World Wide Web Publishing Service (WWW service). The WWW service is a user-mode service.

This counter also accounts for the processing that occurs in the kernel-mode driver, HTTP.sys.

Warning: This counter applies to all instances of IIS running on this machine. It does not apply to instances of the web development server.

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

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

Comparing memory counters

There are many ways to measure memory usage. No single counter reveals a complete picture of the memory requirements of your program, and the values of different counters are incomparable.

By default, ANTS Memory Profiler uses the Private bytes counter, because this measure is considered the most useful for most kinds of memory problem.

Windows Task Manager shows the Private working set counter by default. The Private working set is unavailable in ANTS Memory Profiler because it is only accurate immediately after a garbage collection.

The following table summarizes some of the main different ways of measuring memory:

Counter	**Description**	**Including shared processes? (DLLs in memory, .NET runtime)**	**Including memory paged to disk?**
Private bytes	Includes memory allocated (even if not in use)	No	Yes
Working set	Only includes memory in use	Yes	No

Virtual bytes	Only includes memory in use	Yes	Yes
Private working set (Unavailable in ANTS Memory Profiler)	Only includes memory in use	No	No

Performance Monitor (*perfmon.exe*) has a performance counter called # *Bytes in all heaps*, which is part of the .NET CLR Memory object.

The value of the Performance Monitor # *Bytes in all heaps* counter is often larger than the ANTS Memory Profiler # *Bytes in all heaps* counter. The discrepancy is caused because the Performance Monitor # *Bytes in all heaps* counter includes memory allocated to the .NET heaps but which is not being used, whilst the ANTS Memory Profiler # *Bytes in all heaps* only includes memory that is both allocated and in use.

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 Memory Profiler.
2. Expose your performance counter to the Windows Performance Counter API using the *PerformanceCounter* and *PerformanceCounterCategory* classes of the *System.Diagnostics* namespace.

   For an example of how to do this, see PerformanceCounter class (MSDN).

3. Create a new XML file containing the following:

   ```
   <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 Memory Profiler 7*.
5. Restart ANTS Memory Profiler.
6. The counters you defined are shown in the relevant category on the **Charting Options** tab.

Using the Visual Studio add-in

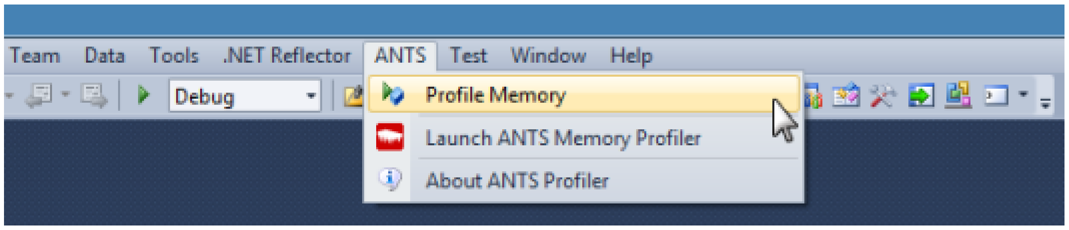
The ANTS Memory Profiler Visual Studio add-in allows you to:

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

Launching ANTS Memory Profiler from Visual Studio

Installing the add-in adds a new **ANTS** menu in Visual Studio.

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



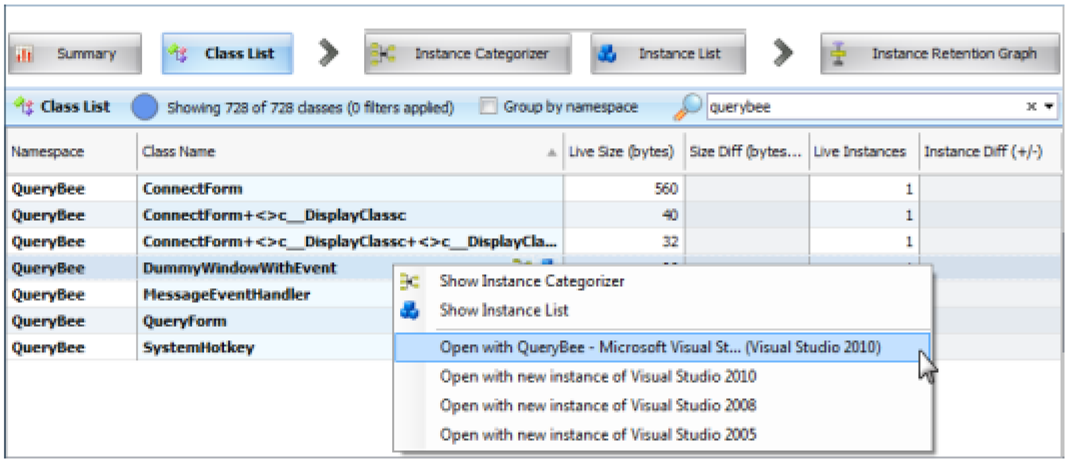
Switching to your source code from ANTS Memory Profiler

So that ANTS Memory Profiler can identify which classes have source code, you must ensure that the .pdb file is in the same directory as the application.

You can switch to source code from the Class List, the Instance Categorizer, the Instance List, and the Instance Retention Graph.

Classes with source code are shown in bold. You can use the find bar on the Class List to search for your class's namespace.

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


Summary
Class List
Instance Categorizer
Instance List
Instance Retention Graph
Class List
Group by namespace
querybee
Show Instance Categorizer
Show Instance List
Open with QueryBee - Microsoft Visual St... (Visual Studio 2010)
Open with new instance of Visual Studio 2010
Open with new instance of Visual Studio 2008
Open with new instance of Visual Studio 2005

Showing 728 of 728 classes (0 filters applied)

Namespace	Class Name	Live Size (bytes)	Size Diff (bytes...	Live Instances	Instance Diff (+/-)
QueryBee	**ConnectForm**	560		1	
QueryBee	**ConnectForm+<>c__DisplayClassc**	40		1	
QueryBee	**ConnectForm+<>c__DisplayClassc+<>c__DisplayCla...**	32		1	
QueryBee	**DummyWindowWithEvent**				
QueryBee	**MessageEventHandler**				
QueryBee	**QueryForm**				
QueryBee	**SystemHotkey**				

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, you need to already have opened the solution in Visual Studio, before opening the context menu. On the context menu, select **Open with (Solution Name) - Microsoft Visual Studio (Visual Studio 20xx)**.

If you experience problems, see Troubleshooting the Visual Studio add-in.

Strategies for memory profiling

This page contains guidelines and recommendations for common memory profiling scenarios, including some heuristics about memory usage patterns that might indicate a memory problem. Of course, every application is different, so these guidelines only outline strategies for memory profiling; to apply these strategies you will need a good understanding of your application.

To get the most useful results, you will need to take one or more snapshots at appropriate times. You can either do this manually, while you interact with your application, or you can take snapshots at precisely the right moment by modifying your code.

Note that Windows Task Manager and Performance Monitor can give misleading results because of the particular memory counters they use. Because of this, you should not try to compare results shown in ANTS Memory Profiler with these other products. For more information about different types of memory counter, see Setting up performance counters.

Identifying types of memory problems

There are many different types of memory issue, so the first step to solving your problem is to work out what type of problem you are facing.

We recommend the following strategy:

1. Check for large object heap fragmentation.
2. Check for a leak in unmanaged memory.
3. Check for a leak in, or excessively high usage of, managed memory.

Checking for large object heap fragmentation

We recommend that you check for large object heap fragmentation before testing for other memory problems.

If you are not familiar with memory profiling, we recommend that before starting you read the .NET Memory Primer.

The main indicator of large object heap fragmentation is that your program throws OutOfMemory exceptions after you have been using the application for a while, and that the timing of the exception's occurrence is not predictable.

Information about large object heap fragmentation is not available when you profile a .NET 1.1 application.

To check for large object heap fragmentation

On the **Summary**, look at the **Memory fragmentation** section. This shows the amount of fragmentation on the large object heap. In addition, ANTS Memory Profiler heuristically checks whether fragmentation could be causing problems for your application.

If ANTS Memory Profiler detects fragmentation, apply the filter **Objects on the large object heap**. This can be useful, for example, to find out which objects are currently on the large object heap, and why they are in memory.

If the large object heap is not fragmented, we recommend that you check for unmanaged memory usage next.

How ANTS Memory Profiler identifies fragmentation problems

ANTS Memory Profiler compares the size of the largest fragment to the total available free space, and also checks how much allocated memory is unused. For a description of the different notices displayed by ANTS Memory Profiler if fragmentation is detected, see Fragmentation notices in ANTS Memory Profiler 7.

Solving large object heap fragmentation

Large object heap fragmentation can be one of the most difficult types of memory problem to solve, because it often involves changes to the architecture of the application. The best approach to use will depend on the exact nature of your program:

- Split arrays into smaller units so that they remain below 85kB (and so are never allocated on the large object heap).
- Alternatively, you can allocate the largest and longest-living objects first (if your objects are files which are queued for processing, for example).
- In some cases, it may be that periodically stopping and restarting the program is the only option.

Fragmentation notices in ANTS Memory Profiler 7

When you profile an assembly, ANTS Memory Profiler analyzes the amount of fragmentation on the Large Object Heap (LOH).

It performs the following calculations to assess whether fragmentation is causing a problem:

Fragmentation ratio = size of the largest empty block on the LOH : total size of all empty blocks on the LOH

Free space ratio = total size of all empty blocks on the LOH : total memory allocated to objects on the LOH

Depending on the result of this calculation, an appropriate message is displayed:

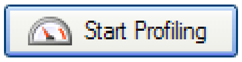
- Memory fragmentation is restricting the size of objects that can be allocated.
  (Displayed in red.)
  The fragmentation ratio is less than 0.3:1.
- Memory fragmentation is affecting the size of the largest object that can be allocated.
  (Displayed in orange.)
  The fragmentation ratio is less than 0.6:1, and greater than 0.3:1.
- Memory fragmentation may be causing .NET to reserve too much free memory.
  (Displayed in red.)
  The free space ratio is greater than 0.7:1, and there are more than three empty blocks over 64k.
- Memory fragmentation may be causing .NET to reserve too much free memory.
  (Displayed in orange.)
  The free space ratio is less than 0.7:1, and greater than 0.4:1, and there are more than three empty blocks over 64k.
- There are many large fragments; fragmentation may become an issue over time.
  (Displayed in black.)
  There are more than ten empty blocks over 64k.

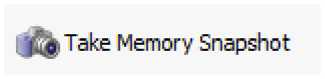
Checking unmanaged memory usage

Before you check your application's usage of unmanaged memory, we recommend that you check for fragmentation of the large object heap.

If you are not familiar with memory profiling, you might find it helpful to read about the .NET heaps before you start.

To check for unmanaged memory usage

1. Set up ANTS Memory Profiler.
2. Click


.
3. Do the action that you think causes the problem (for example, open the dialog box).
4. Click


.
5. On the summary screen, look at the .NET and unmanaged memory pie chart. If there is a large amount of unmanaged usage, you should investigate it; see Identifying the cause of large amounts of unmanaged memory usage.

If you do not have a large amount of unmanaged memory use, continue checking managed memory use.

Identifying the cause of large amounts of unmanaged memory usage

Possible causes of large amount of unmanaged memory usage include:

- Graphics buffers - see below.
- The .NET Common Language Runtime (CLR) itself.
  The CLR itself uses unmanaged memory. If your application uses very little memory in total, the amount of memory used by the CLR may appear to take up a large percentage of the memory in use by the application.
  If the amount of unmanaged memory used in your application is less than about 40MB, the CLR is likely to be the cause. Continue checking for managed memory usage.
- Accessing unmanaged data through P/Invoke or COM+.
  In these cases, ANTS Memory Profiler is unlikely to reveal any useful information, because it cannot profile unmanaged code.
- Problems with object disposal in unmanaged code.
  This is rarely the cause of a problem, but to check for this, switch to the **Class List**. On the **Filters** panel, switch to **Filter By Reference**. Select **Kept in memory exclusively by...**, click **Add new item** and then select **By disposed objects**. For each class that remains in the Class List, click the instance categorizer icon (
  ), to check why instances of that class are being kept in memory.

Checking unmanaged memory usage due to graphics buffers

1. Switch to the **Class List**.
2. On the Filters panel, switch to the **Filter by Object Type** tab and select **Objects that implement**.
3. Click **Add class / interface**, then choose the appropriate graphics class for your application type:
   - For WinForms applications, choose *System.Image.*
   - For WPF applications, choose *System.Windows.Media.Imaging.BitmapImage*.
4. Switch to the **Instance List** and sort it by **Distance from GC Root**.
5. Look at the instances which are furthest from the GC Root. This is because leaked images are likely to be further from the root.
6. Use the instance retention graph to work out why each image is still in memory; see The Instance Retention Graph.
7. If you do not see the cause of the memory problem, and if your application is a WinForms application, repeat the steps above using **Objects that implement** *System.Windows.Media.Imaging.CachedBitmap.*
8. If the objects that implement *System.Windows.Media.Imaging.CachedBitmap filter* does not show the cause of the memory problem, repeat the steps again using **Objects that implement** *System.Windows.Media.ImageSource*.

We recommend that you do not use the instance categorizer for this investigation, because the categorization will tend to emphasize large numbers of small images. It may not show small numbers of large images.

In the Instance List all images will appear to be the same very small size (24 bytes in the case of System.Image objects). This is actually only the amount of managed memory used to create the pointer to the image. The image itself is in unmanaged memory, and so ANTS Memory Profiler cannot show its size.

Checking managed memory usage

Before checking for managed memory usage, we recommend that you first check for large object heap fragmentation and use of unmanaged memory.

If you are not familiar with memory profiling, you might find it helpful to read about the .NET heaps before you start.

There are a number of different techniques that can be used to check for managed memory usage, and, as you become more experienced at using ANTS Memory Profiler, you will discover the best ways to profile your code. Generally, the two main investigations are:

- Finding out what is using most memory.
  Do this if you think that the application is using too much memory.
- Finding and fixing a memory leak.
  Although the .NET Garbage Collector is designed to dispose of objects so that you do not have to, mistakes in your code (or third-party code) can lead to leaking objects. Check for a leak if you see that memory usage increases whenever you carry out a specific action repeatedly.

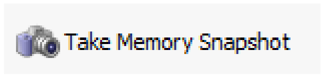
Finding out what is using most memory

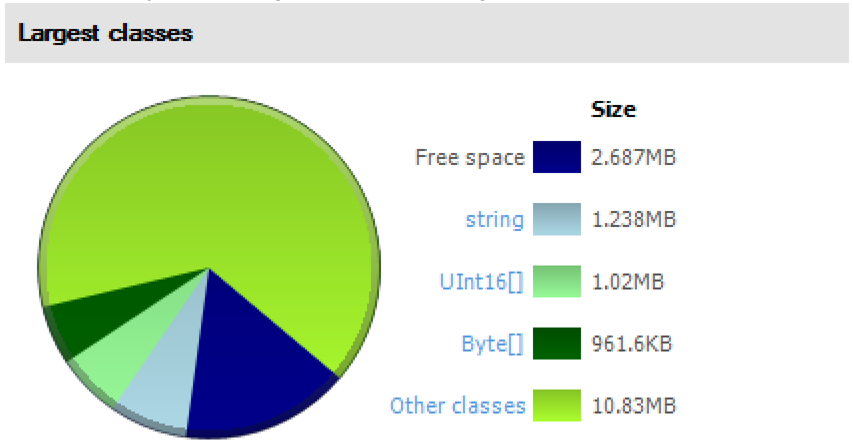
Before finding out what is using most memory, we recommend that you first check for large object heap fragmentation and use of unmanaged memory.

If you are not familiar with memory profiling, you might find it helpful to read the .NET Memory Primer before you start.

> You should not normally filter classes that you do not recognize. For example, if your code contains an array which itself contains a large number of byte arrays, you should start by looking at those byte arrays.

To find out what is using most memory:

1. Start ANTS Memory Profiler and start profiling your application.
2. Use your application until the point where you are interested in its memory. For example, if you believe that a lot of memory is used when a certain screen is displayed, use the application normally until the screen is shown.
3. Click
   

   .
4. On the Summary, look at **Largest classes**. The largest classes are shown in order of the amount of memory they use.
   

5. Starting with the largest class, click each class in turn to see the instance categorizer for that class. For this analysis, you are exploring memory usage by following references to a class, so it is not important if you do not recognize these classes.
6. Instances of the chosen class are grouped by the chain of references which holds them in memory, with the largest group of instances displayed at the top of the screen. Starting with the largest category, use the explorer to look at what is keeping your selected class in memory, reading the graph from right-to-left.
   For example, *System.String* is often the largest class; using the instance categorizer you can find out what is keeping strings in memory, such as an event handler.
7. If the largest class does not reveal interesting results, click **Summary**, and try again with the next-largest class, and so on.
8. If you still cannot see why large amounts of memory are being used after you have examined all of the largest classes by size, click Other classes. The Class List is displayed. Sort the Class List by Live Instances, and look at the instance categorizer for the classes with the most instances.

After using the instance categorizer to find out why instances of a class are still in memory, you might find that some instances are being kept in memory by a type (most commonly an event handler) that should not be referencing those objects. If this happens, when viewing the instance categorizer:

1. Click **Show instances on this path**.
2. The Instance List is displayed, showing the instances of the object which are retained on the path previously shown in the instance categorizer.
3. Click on any instance to select it (it doesn't matter which one, because they are all on the same shortest path) then click
   to display the instance retention graph.
4. Look for the reference that should not be there on the instance retention graph, which shows all paths holding the object in memory.
5. Change your code to remove the incorrect reference, and then check that the problem is fixed.

For more information, see the Worked Example.

Finding and fixing a memory leak

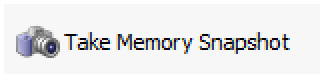
Before you look for a memory leak, we recommend that you first check for large object heap fragmentation and use of unmanaged memory.

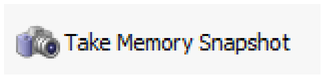
If you are not familiar with memory profiling, you might find it helpful to read about the .NET heaps before you start.

Your approach to finding the source of the leak will depend on whether you have a hypothesis as to what is causing the problem.

If you think you know what is causing the problem

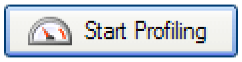
You may already have an idea of the action that might be causing the problem. In this case, use ANTS Memory Profiler to avoid time-consuming trial-and-error investigations in the code.

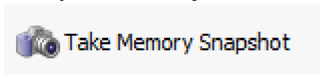
1. Start ANTS Memory Profiler and start profiling your application.
2. Use your application until the point where you are interested in its memory.
3. Click
   

   .
4. Cause the action or method to take place (for example, open the dialog box).
5. Click
   

   .
6. Switch to the **Class List** and try each of the following approaches to check whether the type that you suspect is the source of the problem:
   a. Type the name of the type which you suspect is causing the problem into the search box.
   b. On the filter panel, click **Filter by Reference**. Select **Kept in memory exclusively by**, click **Add class/interface**, and enter the name of the type there.
   c. If you know that the class should only be in memory because it is referenced only by one other class, check using the **Never referenced by** filter.
7. If this procedure does not reveal any problems, continue reading 'Analyzing heap usage', below.

If you do not know what could be causing the problem

1. Set up ANTS Memory Profiler
2. Click
   

   .
3. When you see the **Bytes in all heaps** graph on the timeline increase substantially, click
   

   twice, a few seconds apart.
4. Continue reading at 'Analyzing heap usage' below.

Analyzing heap usage

Our best advice for analyzing heap usage is to ensure that you perform the analysis methodically, noting the results so that you can check that any changes made to your code have fixed the problem.

For each suspicious class (see the list below), in turn:

1. Show the **Instance Categorizer** graph.
2. If the class is one of yours, switch to the **All references** mode, so you can see the objects this class references.
3. Check how the class is being kept in memory by looking at classes displayed to the left of your selected class. Check especially for any event handlers referencing your selected class.
4. If a path looks incorrect, switch to the **Instance Retention Graph** to show how an instance is referenced along that path.

Suspicious classes are:

- The largest classes (see the Summary or Class List)
- Classes displayed when the **Kept in memory exclusively by disposed objects** / **by event handlers** filters are applied. Having objects in the latter is an especially good indicator of a leak (or poor coding practice).
- Classes which ought not to be displayed when the **Survivors in growing classes** and/or **New objects** filters are applied.

> For the largest classes, you can jump straight to the categorized references graph from the summary. You do not need to generate the Class List every time.

Solving the memory leak

Once you have identified a path which is incorrect:

1. Break the incorrect references in your code
2. Retest the leak.

Checking that a memory leak is fixed

This page outlines how to check that a memory leak you previously identified is now fixed:

1. Repeat the steps you used to find the memory leak. If you are looking for a particular class or object, switch to the Class List, and use the 🔍 find box to locate the class or object in which you are interested.
2. If you have fixed the leak, the objects should not be in memory.
   If an object is unexpectedly still found in memory, there are two main likely causes:

   - The object might be on the finalizer queue. To check, display the object on the instance retention graph, and view objects on the finalizer queue by clearing the **Hide finalizer queue GC roots** option on the bar above the graph. If your object is on the finalizer queue, take another snapshot and check again: the object may be removed when the garbage collector runs.
   - The object may still be held in memory on a different path. The instance categorizer only shows retention along the shortest path. If your object was being kept in memory on more than one path, you may have to break references on these other paths, too.

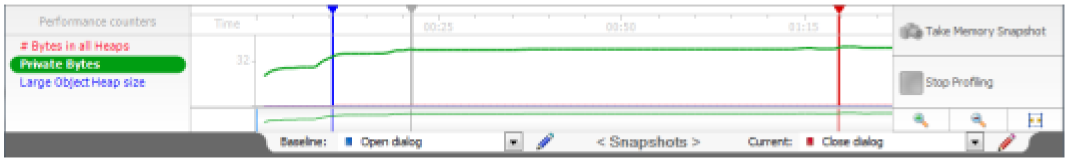
Working with ANTS Memory Profiler

For information on how to work with specific features in ANTS Memory Profiler, see:

- Working with the timeline
- Working with the Summary tab
- Working with the Class List
- Working with the Instance Categorizer
- Working with the Instance List
- Working with the Instance Retention Graph
- Working with object filters
    - Working with basic filters
    - Suggestions for using filters
    - Filtering by object type
    - Filtering by reference
- Using the snapshot API

Working with the timeline

The timeline shows performance counter values and instances of events related to the application you are profiling. You can use the timeline to investigate memory profiling results for specific time periods.



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. For more information, see Setting up performance counters.

Snapshots

While you are profiling your application, click the **Take Memory Snapshot** button to record memory usage at the present moment.

Taking a snapshot calls the `GC.Collect` method, which runs the .NET garbage collector. ANTS Memory Profiler then displays the objects that survive the garbage collection. For more information, see the .NET memory management primer.

Select the snapshot that you are interested in as the **Current** snapshot and select a snapshot to compare it with as the **Baseline** snapshot.

You can name a snapshot to make it easier to identify. To name a snapshot, click

next to its name, and type the new name, followed by the Enter key. The name is shown on the snapshot's tooltip and in the **Baseline:** / **Current:** combo boxes beneath the timeline.

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

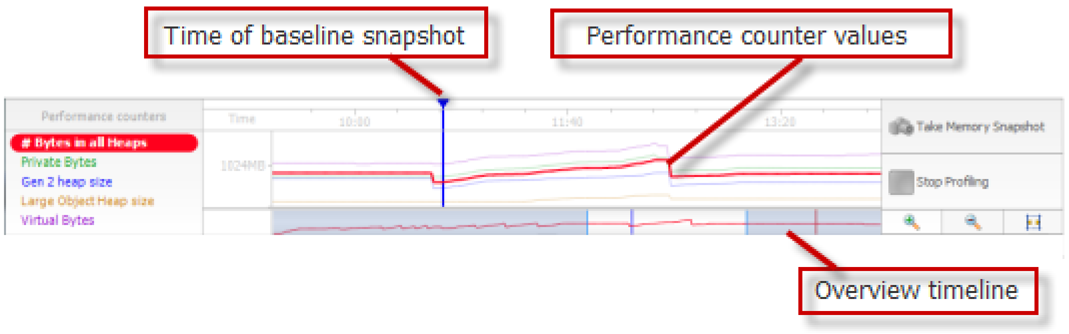
, zoom out

, 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.



Troubleshooting the timeline

If you are profiling a Silverlight application, the heap size counters are not displayed in the timeline. This is due to a limitation in Silverlight. Click the **Session Overview** tab to show these values.

If you are not profiling a Silverlight application, and you cannot see the performance counters in the timeline, you may need to rebuild your performance counters.

Counter values may be different from those shown in Performance Monitor or Windows Task Manager, because there are different ways of counting memory usage. For more details, see Setting up performance counters.

Working with the Summary tab

The Summary page shows a summary of how your application was using memory when you took the current snapshot.

The Summary page is divided into four areas:

- Total size of large objects
- Memory fragmentation
- .NET and unmanaged memory
- Largest classes

Total size of live objects

The following information is shown:

1. **Total size of objects in baseline snapshot (Snapshot Name)** - The amount of memory being used by the .NET heaps at the time of the baseline snapshot.
2. **Total size of objects in (Snapshot Name)** - The amount of memory being used by the .NET heaps in the current snapshot.
3. **Difference between (Current Snapshot name) and (Baseline snapshot name)** - Difference between the two.

If you have not selected a baseline snapshot, or if you have only taken one snapshot, the **Total size of objects in baseline snapshot** and **Differ ence between** snapshots are unavailable.

Use this information to check how memory usage has changed between the snapshots. To change the snapshots being compared, under the timeline, use the **Snapshot:** and **Current:** dropdown lists. You can name the snapshots; see Working with the timeline.

Memory fragmentation

The following information is shown:

1. **Total number of fragments** - The amount of fragmentation.
2. **Number of large fragments** - Having many large fragments increases the likelihood of memory problems. ANTS Memory Profiler also shows the percentage of total free memory accounted for by large fragments. If the number of large fragments is high, and the percentage of free memory accounted for by those fragments is also high, problems are likely to occur sooner.
3. **Wastage due to small fragments** - Small fragments are not bad. They are just an indication that the garbage collection is not compacting the heaps appropriately. If the percentage of free memory accounted for by small fragments is high, you probably do not have a problem with large object heap fragmentation.
4. **Largest fragment** - The size of the largest block of unused memory that is currently reserved for .NET.
5. **The profiler's interpretation of this data.**

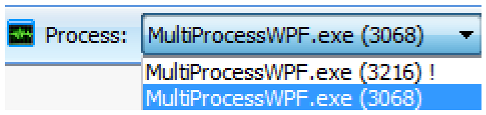
'Fragment' here means 'block of unused memory'.

If you are not used to memory profiling, we recommend that you start by checking for large object heap fragmentation.

If you are profiling an assembly with more than one process

The .NET heaps and garbage collector exist per process, and so it would be meaningless to analyze fragmentation across multiple processes. Use the **Process:** dropdown to choose the process to display fragmentation for. The **Process:** option is not displayed if your application only has one process.

A process followed by an exclamation mark (!) after its name was not running when the current snapshot was taken (either because it had already finished, or because it had not yet started).


Process:
MultiProcessWPF.exe (3068)
MultiProcessWPF.exe (3216) !
MultiProcessWPF.exe (3068)

.NET and unmanaged memory

This pie chart shows the memory used by your program when the current snapshot was taken. If you do not have large object heap fragmentation, use this pie chart to identify whether your problem is caused by an unmanaged or a managed memory problem.

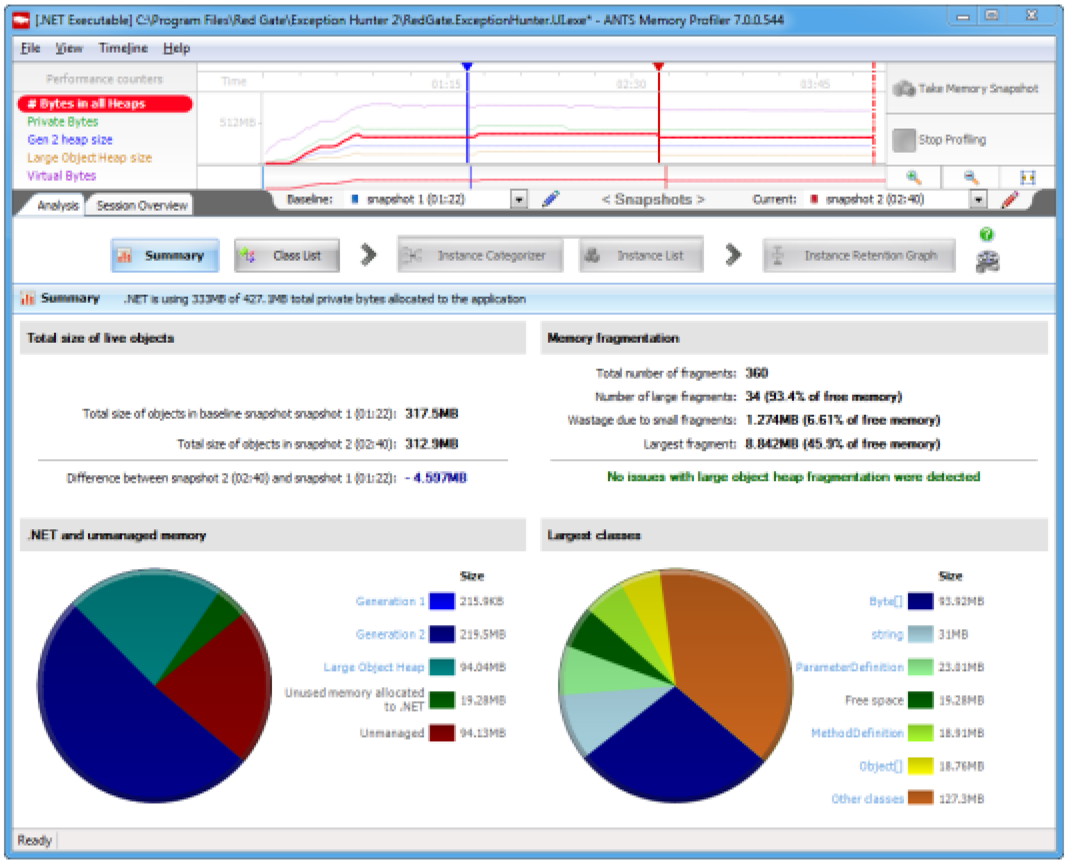
1. **Generation 1** - The size of objects stored on the generation 1 heap (objects that have survived at least one garbage collection). Click the pie chart segment, or the Generation 1 label, to show the classes on the generation 1 heap in the Class List.

2. **Generation 2** - The size of objects stored on the generation 2 heap (long-lived objects that have been promoted from the generation 1 heap). Click the pie chart segment, or the Generation 2 label, to show the classes on the generation 2 heap in the Class List.
3. **Large Object Heap** - The size of objects stored on the large object heap (mainly arrays larger than 85kB). Click the pie chart segment, or the Large Object Heap label, to show the classes on the large object heap in the Class List.
4. **Unused memory allocated to .NET** - The amount of memory which is currently available for .NET.
5. **Unmanaged** - The amount of unmanaged memory used by your program.

The generation 0 heap is not shown because taking a snapshot forces a garbage collection.

Largest classes

After you have determined that you have a problem in the managed memory, and if you are not sure which class is causing the problem, we recommend that you investigate the largest classes. For more information, see Finding out what is using most memory.



Baseline: snapshot 1 (01:22)   Current: snapshot 2 (02:40)

Summary .NET is using 333MB of 427.1MB total private bytes allocated to the application

Total size of live objects

Total size of objects in baseline snapshot snapshot 1 (01:22):	317.5MB
Total size of objects in snapshot 2 (02:40):	312.9MB
Difference between snapshot 2 (02:40) and snapshot 1 (01:22):	- 4.597MB

Memory fragmentation

Total number of fragments:	360
Number of large fragments:	34 (93.4% of free memory)
Wastage due to small fragments:	1.274MB (6.61% of free memory)
Largest fragment:	8.842MB (45.9% of free memory)

No issues with large object heap fragmentation were detected

.NET and unmanaged memory

		Size
Generation 1	🟦	215.9KB
Generation 2	🟦	215.5MB
Large Object Heap	🟦	94.04MB
Unused memory allocated to .NET	🟩	19.28MB
Unmanaged	🟥	94.13MB

Largest classes

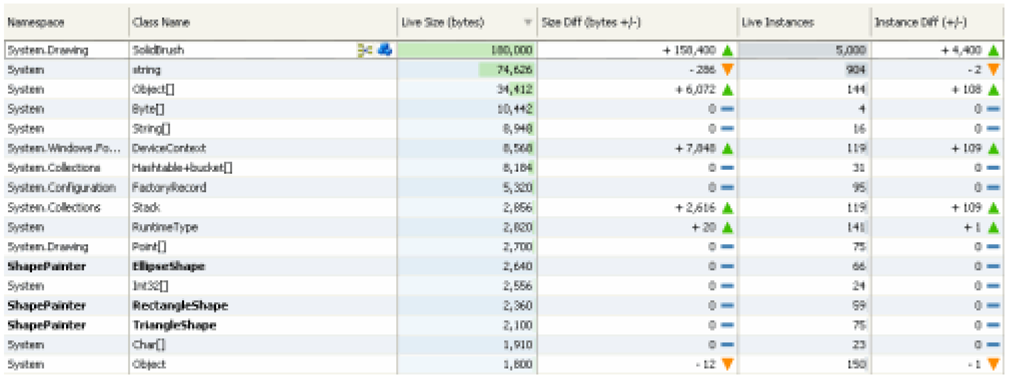
		Size
Byte[]	🟦	93.92MB
string	🟦	31MB
ParameterDefinition	🟩	23.01MB
Free space	🟩	19.28MB
MethodDefinition	🟩	18.91MB
Object[]	🟨	18.76MB
Other classes	🟧	127.3MB

Working with the Class List

Use the Class List to identify classes which you do not expect to be in memory, or classes with unexpectedly high memory usage. Use filters to restrict your investigation to objects with specific characteristics or specific parts of the application.

The class list shows detail of memory usage per class. On the summary or class list, look for classes with unexpectedly high memory usage, or a large increase in size between snapshots.

When you enable filters or use the find box, the values in the list only include objects that match the selected criteria.



Namespace	Class Name	Live Size (bytes)	Size Diff (bytes +/-)	Live Instances	Instance Diff (+/-)
System.Drawing	SolidBrush	180,000	+ 158,400 ▲	5,000	+ 4,400 ▲
System	string	74,626	- 286 ▼	904	- 2 ▼
System	Object[]	34,412	+ 6,072 ▲	144	+ 108 ▲
System	Byte[]	10,442	0 —	4	0 —
System	String[]	8,948	0 —	16	0 —
System.Windows.Fo...	DeviceContext	8,568	+ 7,848 ▲	119	+ 109 ▲
System.Collections	Hashtable+bucket[]	8,184	0 —	31	0 —
System.Configuration	FactoryRecord	5,320	0 —	95	0 —
System.Collections	Stack	2,856	+ 2,616 ▲	119	+ 109 ▲
System	RuntimeType	2,820	+ 20 ▲	141	+ 1 ▲
System.Drawing	Point[]	2,700	0 —	75	0 —
ShapePainter	**EllipseShape**	2,640	0 —	66	0 —
System	Int32[]	2,556	0 —	24	0 —
ShapePainter	**RectangleShape**	2,360	0 —	59	0 —
ShapePainter	**TriangleShape**	2,100	0 —	75	0 —
System	Char[]	1,910	0 —	23	0 —
System	Object	1,800	- 12 ▼	150	- 1 ▼

If you are checking where most memory is used, it can be useful to start by looking at **Live Size** or **Live Instances**. Click on the column heading to sort the column, and look for classes with an unexpectedly large size or number of instances.

Right-click on a class and select **Show Instance Categorizer** to see where instances of the class are being referenced.

If you are looking for a memory leak, it can be useful to begin by looking for unexpected differences between two snapshots in the **Size Diff** or **Instance Diff** columns. Click on the column heading to sort the column, and then look for classes where the memory usage or instance count has increased significantly.

Use the **Comparing snapshots** filters to focus your analysis, depending on the snapshots you are comparing:

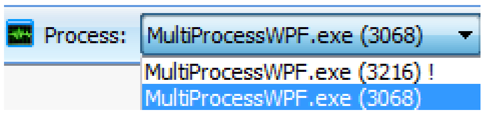
- If you are looking for objects which exist in both the baseline and the current snapshot, select **Only surviving objects**;
- If you are looking for objects which were created between the two snapshots, select **Only new objects**.

After selecting an appropriate filter:

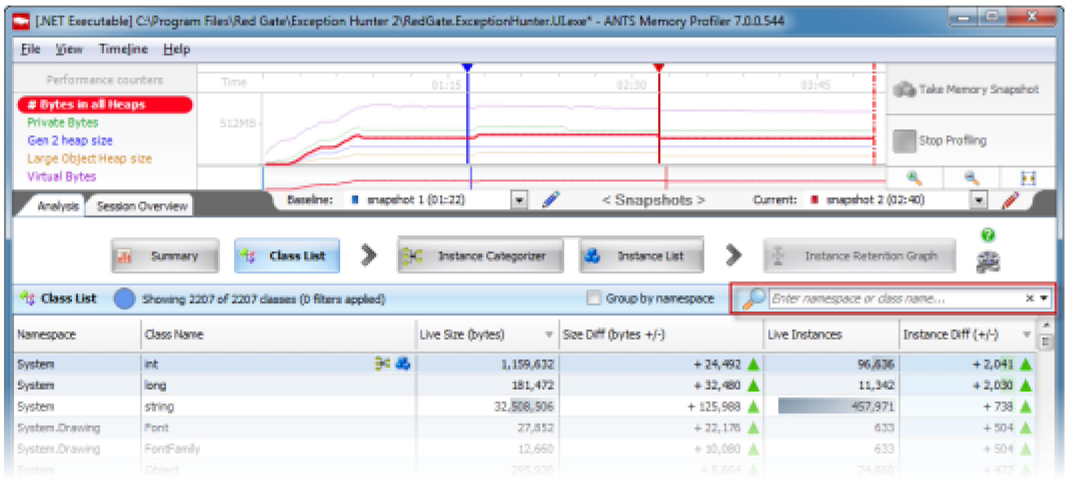
- If the class that looks interesting is one you recognize, right-click and select **Show Instance List** to look for instances of the class.
- If the class that looks interesting is not one you recognize, right-click and select **Show Instance Categorizer** to find out where instances of the class are being referenced.

If your application has more than one process, select the process from which you want to view the classes by using the **Process:** dropdown box. The **Process:** option is not displayed if your application only has one process.

A process followed by an exclamation mark (!) after its name was not running when the current snapshot was taken (either because it had already finished, or because it had not yet started).


Process:
MultiProcessWPF.exe (3068)
MultiProcessWPF.exe (3216) !
MultiProcessWPF.exe (3068)

Finding a specific namespace and class name



Namespace	Class Name	Live Size (bytes)	Size Diff (bytes +/-)	Live Instances	Instance Diff (+/-)
System	int	1,159,632	+ 24,492	96,636	+ 2,041
System	long	181,472	+ 32,480	11,342	+ 2,030
System	string	32,508,506	+ 125,988	457,971	+ 738
System.Drawing	Font	27,852	+ 22,176	633	+ 504
System.Drawing	FontFamily	12,660	+ 10,080	633	+ 504

To find a specific namespace or class, type part of the name in the find box. This can be useful, for example, if you are checking back on a memory leak you have fixed. In many cases, we do not recommend starting your investigation by looking for specific classes; instead, start by looking at the size or instances columns.

We recommend this approach because a lot of the code being executed by your application is likely to be part of the .NET framework libraries or other third-party libraries, so you are likely to see leaks in classes which are not your own, even where your code is the cause of the leak.

Live Size and Live Instances

The **Live Size** column shows the total size of instances of the class in the current snapshot.

The **Live Instances** column shows the total number of instances of the selected class in the current snapshot.

The values do not include instances of classes referenced by the selected class.

These values can be good starting points for finding out where most memory is being used by your application.

To investigate why a class has a large size or high number of instances, do one of the following:

- look at instances of the class on the instance list
- use the instance categorizer to investigate whether another class is keeping instances of your class in memory unexpectedly.

Size Diff and Instance Diff

When you compare two snapshots, the **Size Diff** and **Instance Diff** column show the differences between the baseline snapshot and the current snapshot.

An unexpected increase in the size of a class or number of instances may indicate a memory leak. For example, if you perform an action where you expect new objects to be cleaned up between the snapshots (such as opening and closing a dialog box), you would not expect the class to increase in size or the number of instances to increase.

In some cases, an increase in size or number of instances may not indicate a leak:

- For an application that includes a text editor, the size of the text buffer would be expected to increase as the user adds more text to a document. In this case, the **Size Diff** column for the text buffer class shows an increase in size, but this is not an indication of a memory leak.
- For an application with a text editor backed by a DOM, the number of nodes of a DOM would be expected to increase. In this case, the **In stance Diff** column for the DOM classes shows an increase, but this is not an indication of a memory leak.

The **Instance Diff** column has a specific purpose when either of the following filters is enabled:

- From the current snapshot, show only survivors in growing classes.
- From the current snapshot, show only zombie objects.

See Basic filters for more details.

Working with the Instance Categorizer

Use the Instance Categorizer to investigate the paths that are holding classes in memory.

The instance categorizer shows reference relationships between classes, which can help identify instances that are unexpectedly kept in memory. This can be particularly useful if your snapshot analysis identifies a class you do not recognize: Use the explorer to follow chains of references to the class, until you reach a class you recognize which may be responsible for the memory usage.

When you apply filters, the instance categorizer only includes the filtered objects.

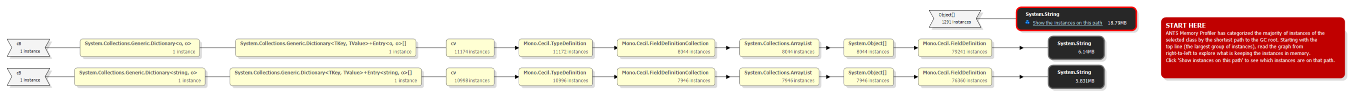
The instance categorizer has two different views:

- Categorized references view
- All references view

Categorized references view

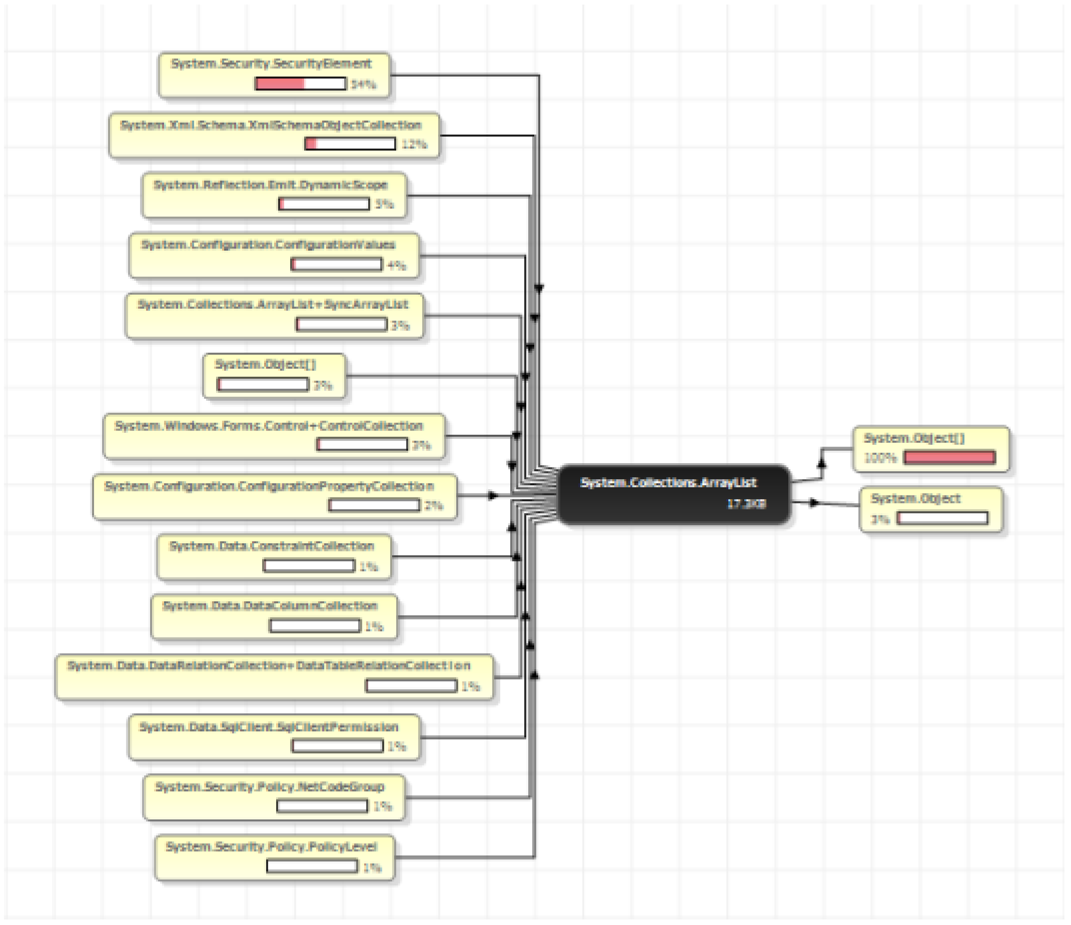
This is the default view. In this view, instances of the selected object are categorized by the shortest path to a GC Root. The path responsible for holding most instances of the selected class is shown at the top.

To see the instances that each category contains, click your selected class (in black, on the right), and then click **Show instance list**.


Object[]
1291 instances
System.String
Show the instances on this path 18.79MB
cB
1 instance
System.Collections.Generic.Dictionary<o, o>
1 instance
System.Collections.Generic.Dictionary<TKey, TValue>+Entry<o, o>[]
1 instance
cv
11174 instances
Mono.Cecil.TypeDefinition
11172 instances
Mono.Cecil.FieldDefinitionCollection
8044 instances
System.Collections.ArrayList
8044 instances
System.Object[]
8044 instances
Mono.Cecil.FieldDefinition
79241 instances
System.String
6.14MB
cB
1 instance
System.Collections.Generic.Dictionary<string, o>
1 instance
System.Collections.Generic.Dictionary<TKey, TValue>+Entry<string, o>[]
1 instance
cv
10998 instances
Mono.Cecil.TypeDefinition
10996 instances
Mono.Cecil.FieldDefinitionCollection
7946 instances
System.Collections.ArrayList
7946 instances
System.Object[]
7946 instances
Mono.Cecil.FieldDefinition
76360 instances
System.String
5.831MB
START HERE
ANTS Memory Profiler has categorized the majority of instances of the selected class by the shortest path to the GC root. Starting with the top line (the largest group of instances), read the graph from right-to-left to explore what is keeping the instances in memory.
Click 'Show instances on this path' to see which instances are on that path.

All references view

In the all references view, your selected class is shown at the center of the graph (in black). Classes to the left have instances that reference any instance of the selected class; classes to the right have instances which are referenced by at least one instance of the selected class.


System.Security.SecurityElement
54%
System.Xml.Schema.XmlSchemaObjectCollection
12%
System.Reflection.Emit.DynamicScope
5%
System.Configuration.ConfigurationValues
4%
System.Collections.ArrayList+SyncArrayList
3%
System.Object[]
3%
System.Windows.Forms.Control+ControlCollection
3%
System.Configuration.ConfigurationPropertyCollection
2%
System.Data.ConstraintCollection
1%
System.Data.DataColumnCollection
1%
System.Data.DataRelationCollection+DataTableRelationCollection
1%
System.Data.SqlClient.SqlClientPermission
1%
System.Security.Policy.NetCodeGroup
1%
System.Security.Policy.PolicyLevel
1%
System.Collections.ArrayList
17.1KB
System.Object[]
100%
System.Object
3%

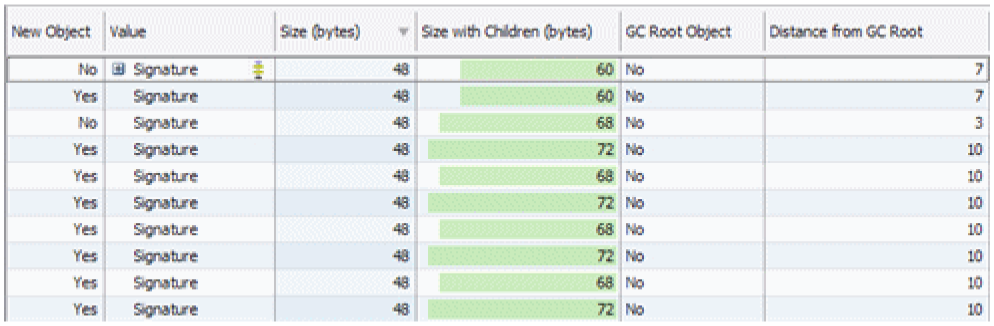
Tips

- The percentage shown on a class indicates the proportion of instances of the selected class (in the center) that are connected to it by references along that path. Look at the percentages on classes to the left of the selected class to assess which classes are responsible for instances of the selected class being in memory.
- To find out why your selected class is unexpectedly large, look at the class that is responsible for most references to the selected class.
    - In the categorized references view, this is on the top row, to the left of the selected class. Click the class to expand the graph, showing classes that reference this class.
    - In all references view, this is the class at the top-left of the graph. Follow the path to the left to see classes that reference this class.
- The graph shows all references between classes, so you may find that as you expand classes and follow references along a particular path you start to see the same classes repeatedly in the path. This is a circular reference chain, and you are unlikely to find useful information by continuing to follow it. Instead, click **Show instances of this class on this path** to display the instance list for one of the classes on this path, and then display the instance retention graph for an instance, to investigate why the instance is in memory.
- If you are looking for the Class Reference Explorer from ANTS Memory Profiler 6 and earlier, select All references view.

Working with the Instance List

The **instance list** shows information about all instances of a class. This can help to identify instances that may be the cause of a memory problem.

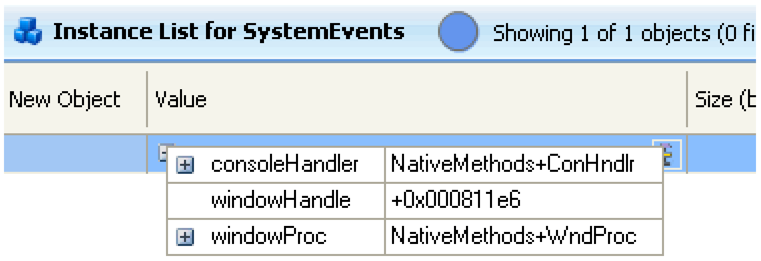
When you apply filters, the instance list only includes the filtered objects.



New Object	Value	Size (bytes)	Size with Children (bytes)	GC Root Object	Distance from GC Root
No	⊞ Signature	48	60	No	7
Yes	Signature	48	60	No	7
No	Signature	48	68	No	3
Yes	Signature	48	72	No	10
Yes	Signature	48	68	No	10
Yes	Signature	48	72	No	10
Yes	Signature	48	68	No	10
Yes	Signature	48	72	No	10
Yes	Signature	48	68	No	10
Yes	Signature	48	72	No	10

The instance list is useful for understanding what instances of a class are in memory, and for identifying objects which are likely to be involved in a memory leak.

Select an instance and then click
⊞
in the **Value** column to find out more about the properties of the specific instance.



Instance List for SystemEvents Showing 1 of 1 objects (0 fi

New Object	Value	Size (b

⊞ consoleHandler	NativeMethods+ConHndlr
windowHandle	+0x000811e6
⊞ windowProc	NativeMethods+WndProc

New Object

The **New Object** column indicates whether the object was created between two snapshots you are comparing, or whether it existed already in the earlier snapshot. (This column is only populated when you are comparing two snapshots.)

When you are comparing two snapshots, either *Yes* or *No* in the **New Object** column may be an indication of leaked objects, depending on when you took your snapshots.

- *Yes*may indicate a memory leak when you are comparing snapshots and you expect instances of the selected class to be cleaned up by the garbage collector between snapshots. For example, you take a snapshot before and after opening and closing a dialog box. Objects created by the action should be cleaned up before you take the second snapshot, so new objects in the second snapshot are likely to indicate a memory leak.

  When you investigate these new objects further, apply the **New objects** filter; this ensures you are only looking at objects which are new in the second snapshot.
- *No* may indicate a memory leak when you are comparing snapshots and you expect instances that exist before the first snapshot to be cleaned up by an action you take between snapshots. For example, you populate a list with data, and then take a snapshot before and after clearing the list. Objects created before the first snapshot should be cleaned up before you take the second snapshot, so old objects in the second snapshot are likely to indicate a memory leak.

  When you investigate these objects further, apply the **Surviving objects** filter; this ensures you are only looking at objects which exist in both snapshots.

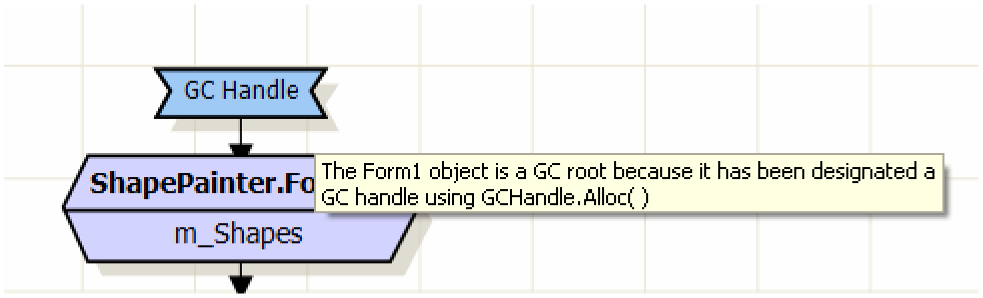
GC Root Object

The **GC Root Object** column indicates whether the object is a GC root object. A GC root can be any storage slot to which the running program has access, such as a local variable, static variables, or a CPU register. (Note that the object itself is not the GC root; the storage slot that holds the reference to the object is the GC root.)

When the garbage collector runs, it determines which objects are not garbage by 'walking the heap', starting at the GC roots. Objects which can be reached by following a chain of references from a GC root are designated as not garbage, and are not collected.

To find out why an object in the instance list is a GC root, select it and then click

. Information on the graph shows why the object is a GC root.



The Form1 object is a GC root because it has been designated a GC handle using GCHandle.Alloc( )

GC root objects are not usually the source of memory leaks. However, they can be useful in *finding* memory leaks because there is always a chain of references between the leaked object and one or more GC roots. To enable the garbage collector to clean up the object, you need to break this chain of references by changing your code to remove one of the "links" in the chain.

The GC Root Object column shows 'Yes - Weakly Referenced' if the object is weakly referenced. Objects with weak references are often used for cacheing because these objects can be destroyed by the garbage collector if memory becomes low. For this reason, weakly referenced objects are not usually the source of memory leaks.

Distance from GC Root

The **Distance from GC Root** column shows the number of references in the chain between the object and its nearest GC root.

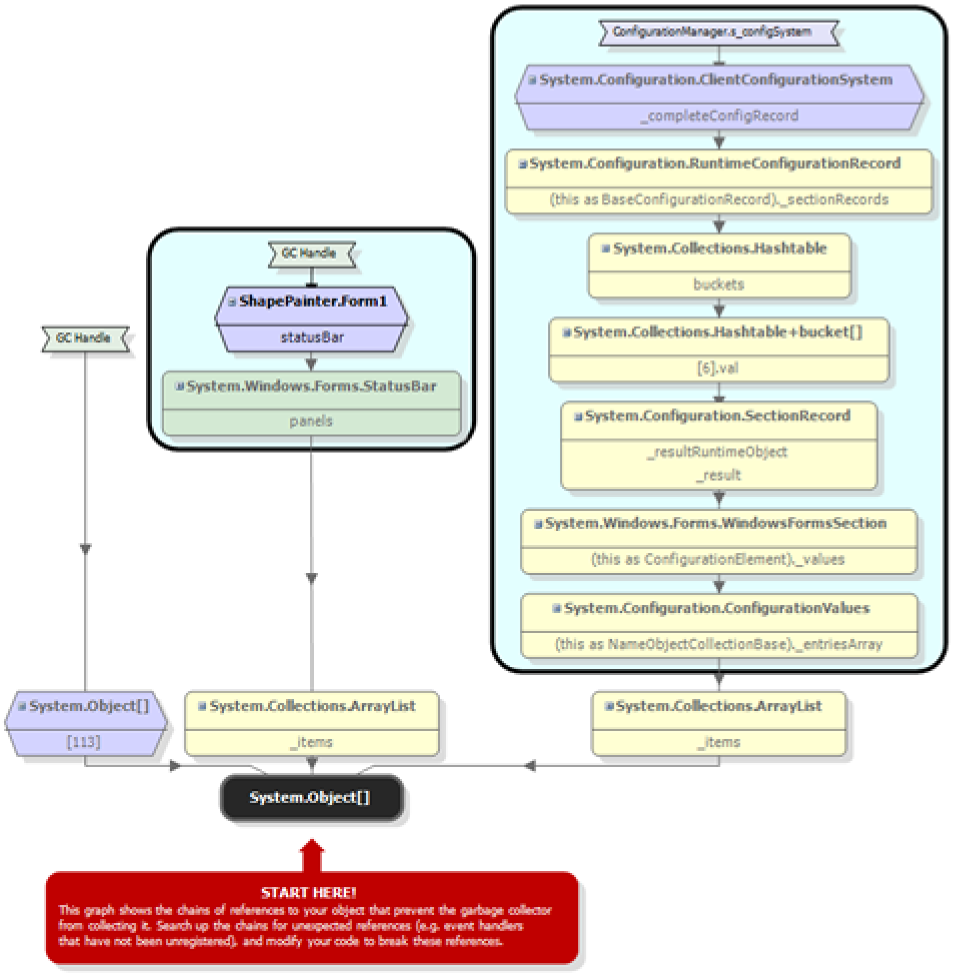
It is likely that shorter, more obvious chains of references between objects and their GC roots have been broken already. Often objects which are at a greater distance from a GC root may be involved in a memory leak, because the chain of references from the GC root to the object is more complex.

Size with children

The **Size with children** column shows the size of the object and any object that it references that is further away from a GC root. This means that the value is a realistic estimate of the amount of memory that would be saved by removing a particular object from memory.

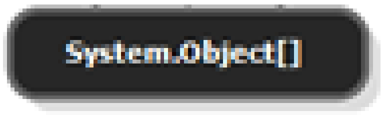
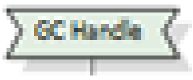
Working with the Instance Retention Graph

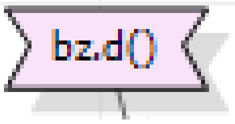
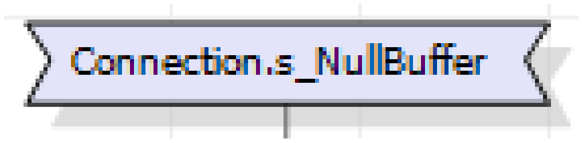
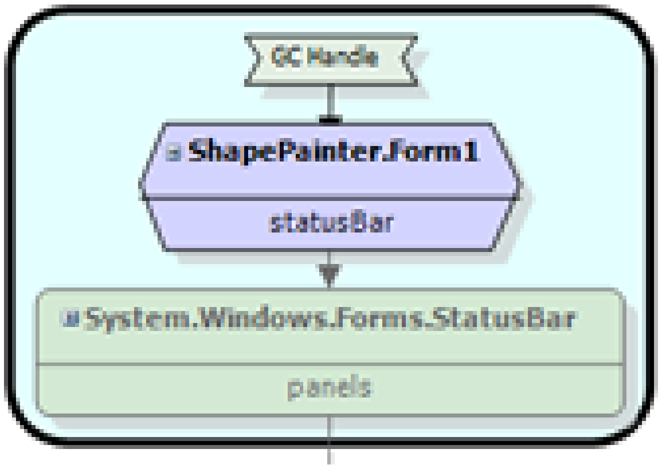
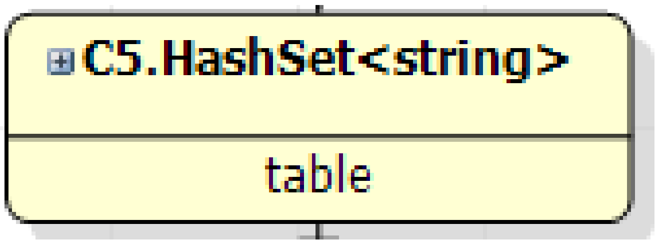
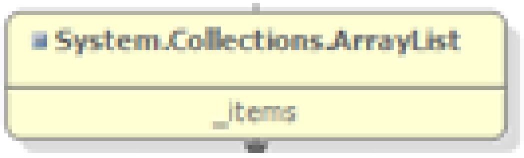
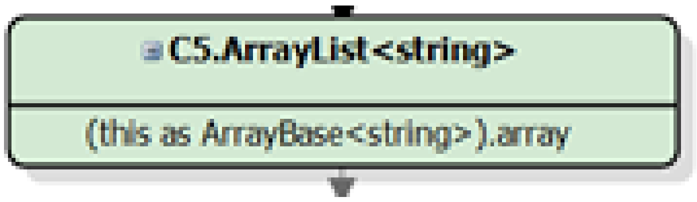
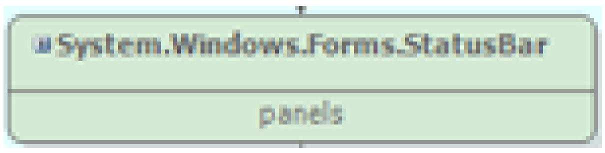
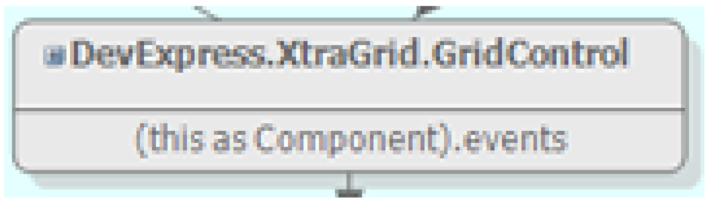
The **instance retention graph** shows chains of references between GC roots and your selected object. Start at your selected object and follow the chains of references up towards the GC roots, to identify references that are preventing the garbage collector from collecting your object. When you find an unexpected reference, modify your code to break the reference, and then profile the application again to check the problem is fixed.

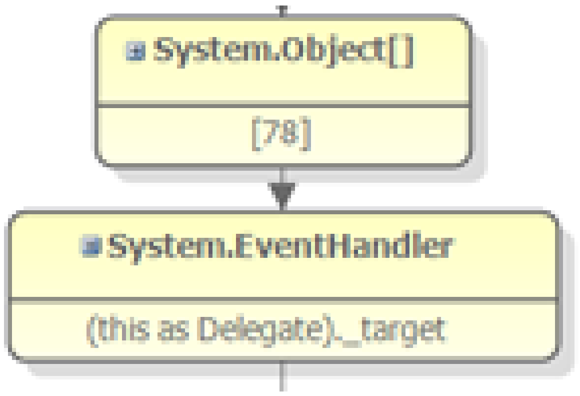

ConfigurationManager.s_configSystem
System.Configuration.ClientConfigurationSystem
_completeConfigRecord
System.Configuration.RuntimeConfigurationRecord
(this as BaseConfigurationRecord)._sectionRecords
System.Collections.Hashtable
buckets
System.Collections.Hashtable+bucket[]
[6].val
System.Configuration.SectionRecord
_resultRuntimeObject
_result
System.Windows.Forms.WindowsFormsSection
(this as ConfigurationElement)._values
System.Configuration.ConfigurationValues
(this as NameObjectCollectionBase)._entriesArray
GC Handle
ShapePainter.Form1
statusBar
System.Windows.Forms.StatusBar
panels
GC Handle
System.Object[]
[113]
System.Collections.ArrayList
_items
System.Collections.ArrayList
_items
System.Object[]

START HERE!
This graph shows the chains of references to your object that prevent the garbage collector from collecting it. Search up the chains for unexpected references (e.g. event handlers that have not been unregistered), and modify your code to break these references.

Key

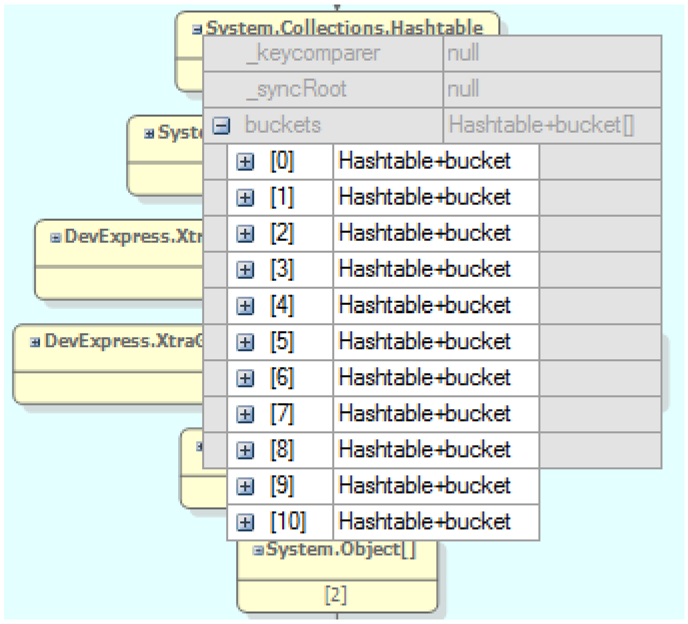
	The instance you selected.
System.Object[] [113]	GC root object.
	The object under this one in the graph is a GC root because it is a GC Handle.

	The object under this one in the graph is a GC root because it is on the stack, having been put on the stack by `bz.d()`.
	The object under this one in the graph is a GC root because it is a member of the static variable `Connection.s_NullBuffer`.
 GC Handle ShapePainter.Form1 statusBar System.Windows.Forms.StatusBar panels	Group of strongly-connected objects (see tips below).
	Non-disposable instance for which you have source code.
	Non-disposable instance for which you do not have source code.
	Disposable instance for which you have source code.
	Disposable instance for which you do not have source code.
 DevExpress.XtraGrid.GridControl (this as Component).events	Disposed instance.

 System.Object[] [78] System.EventHandler (this as Delegate)._target	The simplest path between these two instances. (Note that if you break this link, the objects may still be connected by a more complex link.)

Move your mouse pointer over an object to view more information in a tooltip.

Click
⊞
on a node in the graph to see the properties of a specific instance.



_keycomparer	null
_syncRoot	null
⊟ buckets	Hashtable+bucket[]
⊞ [0]	Hashtable+bucket
⊞ [1]	Hashtable+bucket
⊞ [2]	Hashtable+bucket
⊞ [3]	Hashtable+bucket
⊞ [4]	Hashtable+bucket
⊞ [5]	Hashtable+bucket
⊞ [6]	Hashtable+bucket
⊞ [7]	Hashtable+bucket
⊞ [8]	Hashtable+bucket
⊞ [9]	Hashtable+bucket
⊞ [10]	Hashtable+bucket

Tips

- The instance retention graph only shows the shortest chain of references from each GC root to your selected object. When you break this chain of references, the object may still be kept in memory by a longer chain of references.
  After you have modified your code to break the first chain of references, profile your application again; the instance retention graph updates to show the chain of references which is now the shortest chain. You will need to modify your code again, to break this chain of references, and repeat until all the chains of references are broken and the object is no longer in memory.
- Objects grouped in a box are strongly connected; every object references every other object in the group (the reference may be direct or indirect). To remove an object from memory, you do not normally need to break all the references between a GC root and your object: only *one*of the references needs to be removed to prevent your object from being kept in memory.

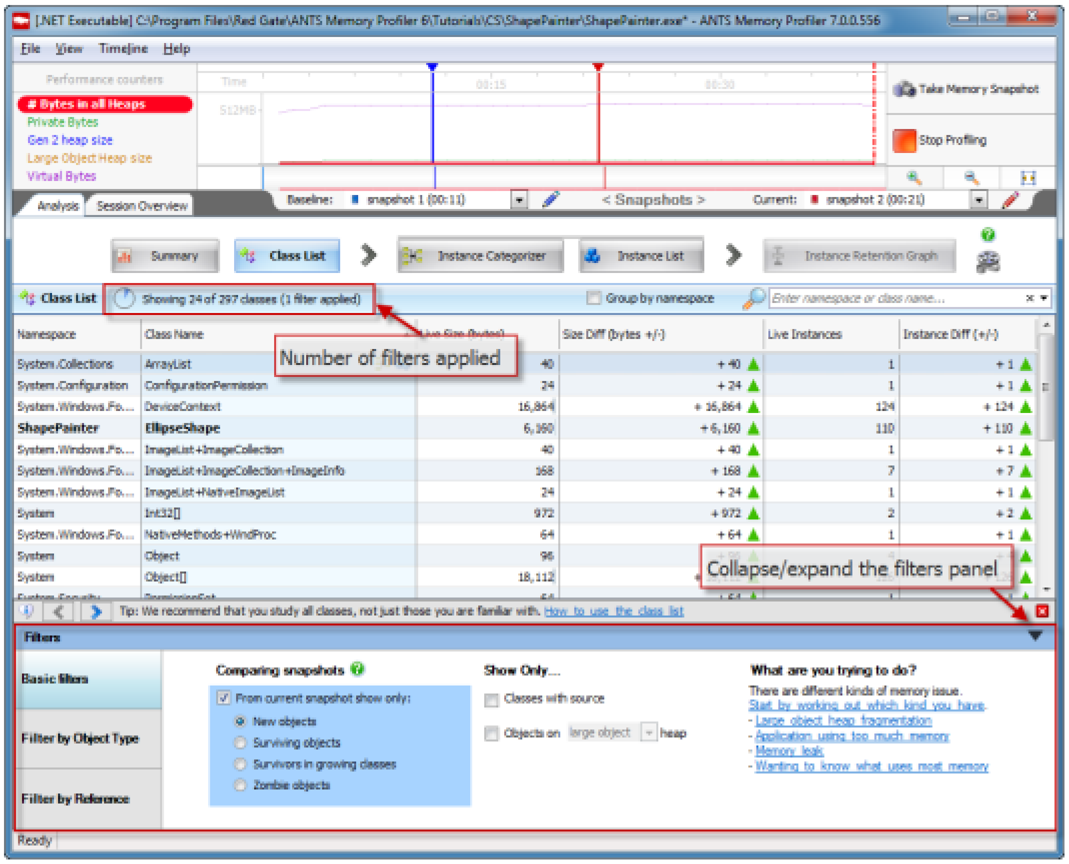
  However, the relationship between strongly connected objects is complex, so in this case you may need to break more than one reference to prevent your object from being held in memory. Break the references one at a time, and take new snapshots each time to check whether your object is still in memory.
- If there is an event handler in the chain of references from a GC root to your object, look at the objects that directly reference the event handler; these references are often a good point to break the chain of references to your object.
- If your graph shows an object which does not appear to be referenced by anything, it may be because this object is on the finalizer queue. The graph hides finalizer queue GC roots by default, because they do not normally indicate a memory problem. To show these objects on the graph, clear the **Hide finalizer queue GC roots** option (in the bar above the graph).
- If the graph shows your object is being kept in memory by an object on the finalizer queue, take another memory snapshot. Taking a snapshot forces the garbage collector to run, so your object should now be removed from memory.

Working with object filters

Use the filters panel to focus your memory usage investigation on objects that are more likely to be of interest.

Select one or more of the filters to show only objects that match all of the selected conditions. Other objects are hidden.

The bar above the results area indicates the number of objects affected by the current filters:



Class List — Showing 24 of 297 classes (1 filter applied) — Group by namespace — Enter namespace or class name...

Namespace	Class Name	Live Size (bytes)	Size Diff (bytes +/-)	Live Instances	Instance Diff (+/-)
System.Collections	ArrayList	40	+ 40	1	+ 1
System.Configuration	ConfigurationPermission	24	+ 24	1	+ 1
System.Windows.Fo...	DeviceContext	16,864	+ 16,864	124	+ 124
ShapePainter	**EllipseShape**	6,160	+ 6,160	110	+ 110
System.Windows.Fo...	ImageList+ImageCollection	40	+ 40	1	+ 1
System.Windows.Fo...	ImageList+ImageCollection+ImageInfo	168	+ 168	7	+ 7
System.Windows.Fo...	ImageList+NativeImageList	24	+ 24	1	+ 1
System	Int32[]	972	+ 972	2	+ 2
System.Windows.Fo...	NativeMethods+WndProc	64	+ 64	1	+ 1
System	Object	96	[illegible]	[illegible]	[illegible]
System	Object[]	18,112	[illegible]	[illegible]	[illegible]

Tip: We recommend that you study all classes, not just those you are familiar with. How to use the class list

Filters

Basic filters / Filter by Object Type / Filter by Reference

Comparing snapshots
[✓] From current snapshot show only:
[✓ New objects]
Surviving objects
Survivors in growing classes
Zombie objects

Show Only...
[ ] Classes with source
[ ] Objects on large object heap

What are you trying to do?
There are different kinds of memory issue.
Start by working out which kind you have.
- Large object heap fragmentation
- Application using too much memory
- Memory leak
- Wanting to know what uses most memory

Ready

When using filters, you may see some objects that do not match the filter criteria. If this happens, take another snapshot. This forces a garbage collection so that any objects on the finalizer queue are removed.

For more information about the different types of filters, see:

- Basic filters
- Filter by object type
- Filter by reference

Filters cannot identify memory problems themselves, but they can help by focusing your investigation. For notes about filters that can be useful in some conditions, see Suggestions for using filters.

When you apply multiple filters, only objects that match **all** the filters are shown (i.e. there is an AND relationship between filters).

For example, select both the following filters:

- **Disposed objects which are still in memory**
- **Objects on *Gen 1* heap**

Objects are shown only if Dispose() has been called on them *and* they are on the Gen 1 heap.

Working with basic filters

This page contains a description of the basic filters in ANTS Memory Profiler.

For a summary of how to use them, see Using filters to find objects. For a description of which filters are most useful, see Suggestions for using filters.

Comparing snapshots

Show objects in your current snapshot, based on the comparison with your baseline snapshot.

From the current snapshot show only new objects

Show only objects created between the baseline snapshot and the current snapshot.

Use this filter to find objects that did not exist in the baseline snapshot. This is useful if you want to understand why memory usage has increased between the snapshots.

From the current snapshot show only surviving objects

Show only objects that remain in memory in both the baseline snapshot and the current snapshot.

Use this filter to find objects that exist in both snapshots. This is useful if you want to look for objects which might be leaking in the current snapshot, because they should have been disposed between the two snapshots.

From the current snapshot show only survivors in growing classes

Show only objects that match all of the following conditions:

- were present in the baseline snapshot
- are present in the current snapshot
- are instances of a class which has more instances in the current snapshot than in the baseline snapshot.

When this filter is applied, in the **Class List**, the Instance Diff column assumes that all instances in the baseline snapshot are potential survivors, and then calculates the difference between the number of survivors in the current snapshot and the number of potential survivors from the baseline snapshot. An instance diff of 0 means that all objects survived; an instance diff of -10 means that 10 objects that were in the baseline snapshot are not in the current snapshot.

Because only growing classes are included in this filter, the instance diff will never be a positive number.

This filter is particularly useful, because classes with an instance diff of 0 are good candidates for finding memory leaks (no instances of the class were disposed between the two snapshots).

From the current snapshot show only zombie objects

Show only objects that are still in memory in the current snapshot, even though there were indications in the baseline snapshot that they would not survive.

This includes objects with the following characteristics:

- Objects on which `Dispose()` has been called in the baseline snapshot, but which are still in memory in the current snapshot because garbage collection was prevented for some reason
- Objects on the finalizer queue in the baseline snapshot (this includes objects still on the finalizer queue in the current snapshot and objects which are no longer on the finalizer queue in the current snapshot)

When this filter is applied, in the **Class List**, the Instance Diff column compares the number of objects on the finalizer queue in the baseline snapshot to the number of objects on the finalizer queue in the current snapshot. An instance diff of -1 means that one object that was on the finalizer queue in the baseline snapshot is no longer on the finalizer queue.

This filter is useful because these objects should not normally still be in memory in the current snapshot.

Show Only...

Classes with source

Show only objects with source code.

Note that you should not normally use this filter when you start looking for the cause of a memory problem (see Understanding ANTS Memory Profiler).

Objects on the large object / Gen 0 / Gen 1 / Gen 2 heap

Show only objects in the selected area of memory.

Use this filter to understand how objects are allocated in memory. Looking at objects on the Gen 2 heap may be especially helpful, because these are the longest-surviving small objects.

You can also use this filter to investigate objects on the large object heap, which is useful if you have large object heap fragmentation.

Note that you will not normally see any objects on the Gen 0 heap during profiling, because taking a snapshot forces a garbage collection. For more information, see 'Understanding the small object heaps' in the .NET Memory Primer.

> This filter is not available when you are profiling a .NET 1.1 application.

Suggestions for using filters

Filters cannot identify memory problems themselves, but they can help by focusing your investigation. The following notes suggest filters that can be useful in some conditions.

Filters that can help find a memory leak

The following filters can be useful in finding a memory leak:

- **From the current snapshot, show only survivors in growing classes**
  This filter is particularly useful, because classes with an instance diff of 0 on the Class List are good candidates for finding memory leaks (no instances of the class were disposed between the two snapshots).
- **Disposed objects which are still in memory**
  This filter is useful because disposed objects should not normally be kept in memory.
- **Kept in memory exclusively by event handlers**
  Event handlers should not normally be the only reason an object is kept in memory, so this filter can help identify a memory leak.
- **Kept in memory exclusively by disposed objects**
  Disposed objects should not normally be kept in memory, so this filter can help identify a memory leak.
- **From the current snapshot, show only zombie objects**
  This filter is useful because these objects should not normally still be in memory in the current snapshot.

The 'Disposed objects which are still in memory' and 'Kept in memory only by event handlers' filters are not available when profiling a .NET 1.1 application.

Filters that can help focus on a specific part of your application

The following filters can be particularly useful when you want to focus on a specific part of an application, or exclude a specific part of an application:

- **Objects on large object / Gen 0 / Gen 1 / Gen 2 heap**
  Use this filter to understand how objects are allocated in memory. Looking at objects on the Gen 2 heap may be especially helpful, because these are the longest-surviving small objects.
  You can also use this filter to investigate objects on the large object heap, which is useful if you have large object heap fragmentation.
- **Kept in memory exclusively by objects from namespace**
  Use this filter to look for your classes, excluding third-party and framework code. You should not normally do this when first looking for the source of a memory leak, because this filter may hide the symptoms of a leak originating in your code.
- **Never referenced by an instance of class/interface**
  Use this filter if you think that all classes should be referenced by a particular class or interface. If a class is still shown, it is being kept in memory for a different reason.

The 'Objects on large object / Gen 0 / Gen 1 / Gen 2 heap' filters are not available when profiling a .NET 1.1 application.

See the worked example, which demonstrates using filters in your memory profiling workflow.

Filtering by object type

This page describes the filters that allow you to filter by the object type.

For a summary of how to use them, see Using filters to find objects. For a description of which filters are most useful, see Suggestions for using filters.

Objects that are disposable

Show objects that implement the IDisposable interface. This filter is identical to the **Referenced by: System.IDisposable** filter.

Use this filter to check whether objects which should be disposable (whether or not they have actually been disposed) are still in memory.

Objects that are / are not disposed

When set to *are,* show only objects on which Dispose() has been called, but which cannot be garbage-collected because a reference to the object still exists in memory.

This filter is useful because disposed objects should not normally be kept in memory. When you have identified a disposed object with this filter, display it on the object retention graph, then follow the chains of references to identify the objects keeping the disposed object in memory.

This filter is not available when you are profiling a .NET 1.1 application.

Objects that are / are not GC roots

Show only GC root objects, or hide GC root objects

If you are investigating a memory leak, it may be useful to hide GC root objects, because they are not generally the source of leaks.

If you are performing a general check on memory usage, it may be useful to show only GC root objects. Show the class list to see details of classes with instances which are GC roots, and then use the class reference explorer to investigate the relationships between these classes and other classes with instances in memory.

To look for specific types of GC roots, combine this filter with the **Kept in memory exclusively by GC roots of type** filters on the **Filter by Reference** tab.

Objects that implement class/interface

Show objects that:

- inherit from the selected class
- implement the selected interface
- are objects of the selected class/interface.

If more than one class/interface is selected, objects are shown only if they implement all of the selected classes/interfaces.

This filter is useful if you believe that the selected class/interface may be being kept in memory by other type which implements it.

Filtering by reference

This page describes the filters that allow you to filter by the references to objects.

For a summary of how to use them, see Using filters to find objects. For a description of which filters are most useful, see Suggestions for using filters.

Kept in memory exclusively by

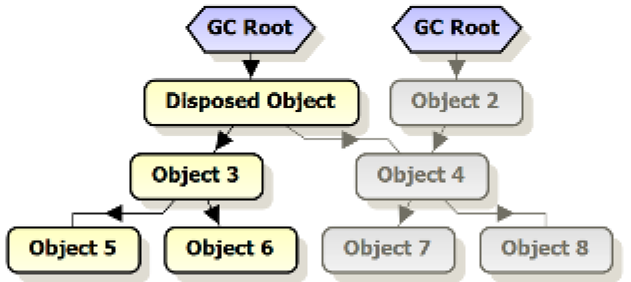
Disposed objects

This filter shows disposed objects and objects where all chains of references between the object and a GC root go through a disposed object.

The disposed object itself is included in the objects shown. Objects which are disposed but are on the finalizer queue are not shown.

To enable this filter, select **Kept in memory exclusively by**, then click **Add new item**, then select **Disposed objects**.

Example: The yellow objects are the objects shown by this filter.



Disposed objects should not normally be kept in memory, so this filter can help identify a memory leak.

When you have identified an object that is kept in memory only by a disposed object, display it on the object retention graph, and follow the chains of references to identify the objects keeping the selected object in memory.

This filter is not available when you are profiling a .NET 1.1 application.

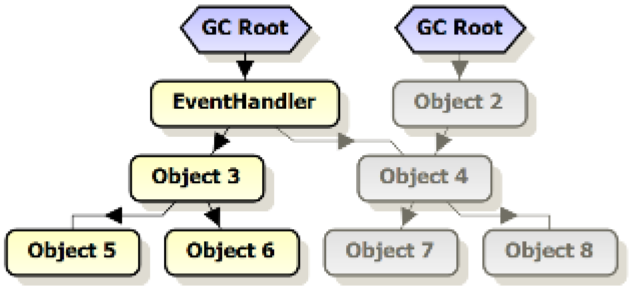
Event handlers

Show only objects where all chains of references between the object and a GC root go through an event handler.

The event handler is included in the objects shown. Objects which are held in memory only by event handlers but are on the finalizer queue are not shown.

To enable this filter, select **Kept in memory exclusively by**, then click **Add new item**, then select **Event handlers**.

Example: The yellow objects are the objects shown by this filter.



Event handlers should not normally be the only reason an object is kept in memory, so this filter can help identify a memory leak.

When you have identified an object that is kept in memory only by an event handler, display it on the object retention graph, and follow the chain of references from the object to the event handler; it is likely that the cause of the problem is an object close to the event handler.

GC root of type GC root type

Show objects with GC roots of the specified types.

> The GC root object is included in the objects shown.

To enable this filter, select **Kept in memory exclusively by**, then click **Add new item**, then select **GC root of type...**.

When you select more than one type of GC root, objects are shown if they have a GC root which is any of the selected types.

This filter is especially useful because, if you filter objects that are kept in memory exclusively by a GC root of type finalizer queue, this will show objects on which Dispose() should be called.

To look for specific types of GC roots, combine this filter with the **Objects which are GC roots** filter.

Undisposed objects

Show undisposed objects where no chain of references between the object and its GC roots go through a disposed object.

Use this if you want an estimate of how much memory could be freed by disposing the objects.

To enable this filter, select **Kept in memory exclusively by**, then click **Add new item**, then select **Undisposed objects**.

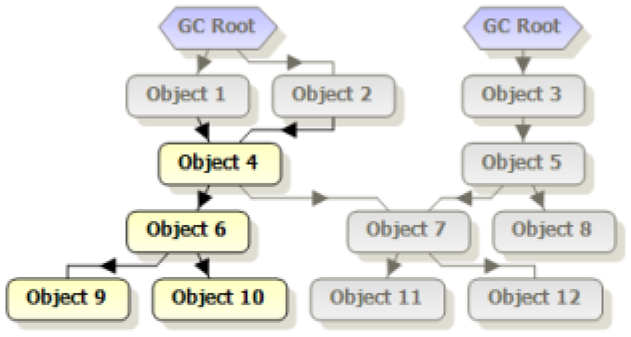
Add class / interface...

Show instances of the specified class or interface (including derived types) and objects where instances of the specified class or interface exist in all chains of references between the object and a GC root.

> Instances of the specified class are included in the objects shown.

To enable this filter, select **Kept in memory exclusively by**, then click **Add class / interface**, then select the required class or interface.

Example: The yellow objects are the objects shown by the filter (where *Object 4* is the only object that is an instance of the selected class or that implements the selected interface).



This filter shows objects where instances of the specified class or interface (including derived types) exist in *all* chains of references between the object and a GC root. To show objects where instances of the specified class exist in *at least one* chain of references (but not necessarily in *all* ch ains of references) between the object and a GC root, use the **Referenced by instances of class/interface** filter. Objects which match your criteria but are on the finalizer queue are *not* shown.

If you select more than one class / interface, objects which are kept in memory by all of the listed classes / interfaces are shown.

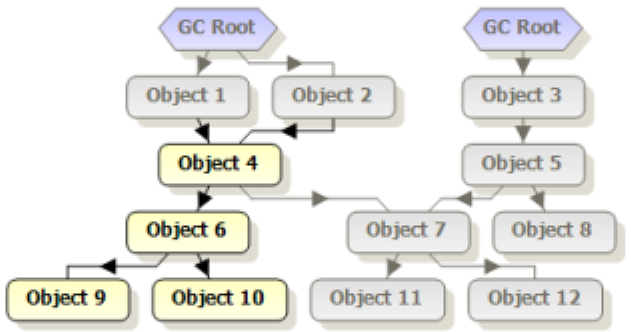
Add namespace...

Show objects from the specified namespace and objects where instances of classes from the specified namespace exist in all chains of references between the object and a GC root.

To enable this filter, select **Kept in memory exclusively by**, then click **Add namespace**, then select the required namespace.

Example: objects kept in memory exclusively by a class from the namespace that *Object 4* is an instance of:

(Note that *Object 4* - which is an instance of the a class in the specified namespace - is included in the objects shown)



If you select more than one namespace, objects which are kept in memory by all of the listed namespaces are shown.

> Objects which match your criteria but are on the finalizer queue are *not* shown.

When you select multiple criteria for this filter, objects are only shown if they are satisfy all of the selected criteria.

Referenced by

The **Referenced by** filters behave identically to the **Kept in memory exclusively by** filters, except that objects are shown if at least one of the paths of reference to the GC route goes through the specified type or GC Root. The reference may be direct or indirect.

(With the Kept in memory exclusively by filters, objects are shown if all of the paths of reference to the GC route go through the specified type or GC Root.)

When you select multiple criteria for this filter, objects are only shown if they are satisfy all of the selected criteria.

Never referenced by

The Never referenced by filters have the opposite behavior to the Objects referenced by filters. Objects are shown if none of the paths of reference (including indirect references) to the GC root goes through the specified type of GC Root.

> When viewing the Class List, the number of classes shown when the Never referenced by filter is enabled, plus the number of classes shown when the Referenced by filter is enabled, might add up to more than the total number of classes in the application. This is because some classes might have instances that are referenced by the selected type, while other instances of the same class are never referenced by the selected type.

The **Add class/interface** filter can be especially useful. Some examples of when you might want to use this filter:

- You know that all the data in your application should be referenced by a single main class. Apply the **Never referenced by an instance of class/interface** filter to remove objects referenced by the main class from the results.
- Your application performs caching (for example, web applications that cache the query results). Cached objects are deliberately kept in memory for a period of time. Apply the **Never referenced by an instance of class/interface** filter to exclude objects referenced by a cache class (for example, *System.Web.Caching.Cache*, for ASP.NET applications).

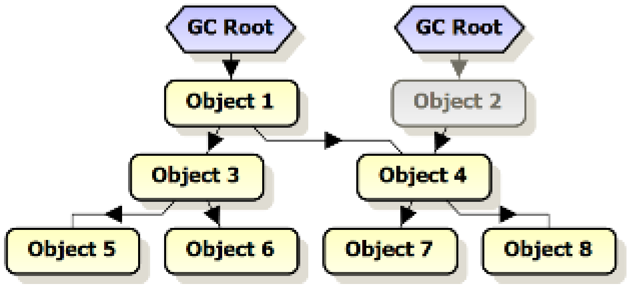
Relationships between objects

Filters on the **Filter by Reference** tab enable you to narrow down your search for memory problems by concentrating on certain types of relationships between objects.

Referenced by

Objects may be in memory because another object references them; the object is on at least one of the chains of references between the selected object and a GC root.

Example: objects referenced by *Object 1*

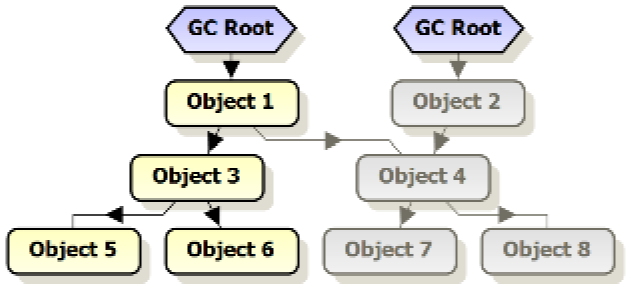


In this example all objects except *Object 2* are referenced by *Object 1*, either directly or indirectly. Note that the filter selection includes the specified object, *Object 1*.

Kept in memory exclusively by

Some filters show objects where the selected object is in all chains of references between the objects and a GC root - that is, objects are kept in memory *only* by the selected object.

Example: objects kept in memory by *Object 1* (note that the filter includes the selected object, *Object* 1)



In this example, only four objects are kept in memory only by *Object 1*. *Object 1* is not in all the chains of references between the remaining objects and their GC roots; for example, *Object 4* has another GC root which references it via *Object 2*. Note that the filter selection includes the specified object, *Object 1*.

Using the snapshot API

You can use the API exposed by ANTS Memory Profiler to take snapshots from within your code. First, you need to edit your application's code to trigger the snapshot, then run the built application alongside ANTS Memory Profiler.

Take snapshots from your code to ensure that snapshots are taken precisely when you believe that a leak occurs, or if your application does not have an interface through which you can control it. The snapshot API lets you essentially automate the process of taking a snapshot.

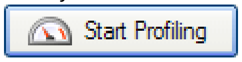
> If your application is a Silverlight browser application, you cannot use the snapshot API. (This is due to security restrictions in Silverlight.)

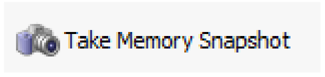
Editing your code to invoke the snapshot:

1. Create a reference to *RedGate.MemoryProfiler.Snapshot.dll* (You can find a copy in *%ProgramFiles%\Red Gate\ANTS Memory Profiler 7*)
2. Call `RedGate.MemoryProfiler.Snapshot.TakeSnapshot()` whenever you want to take a snapshot. This method returns `true` on success and `false` on failure. To take a snapshot and give it a memorable name at the same time, provide a string argument to the `TakeSnapshot` method, for example: `RedGate.MemoryProfiler.Snapshot.TakeSnapshot("Clicked button")`.

> We recommend that you contain the `TakeSnapshot()` method within a `try-catch` statement. This is because `TakeSnapshot()` can raise exceptions, for example if it is called more than 5 times in 30 seconds.

Running your application

1. Start ANTS Memory Profiler.
2. Set up ANTS Memory Profiler to profile your application in the normal way.
3. When you click
   

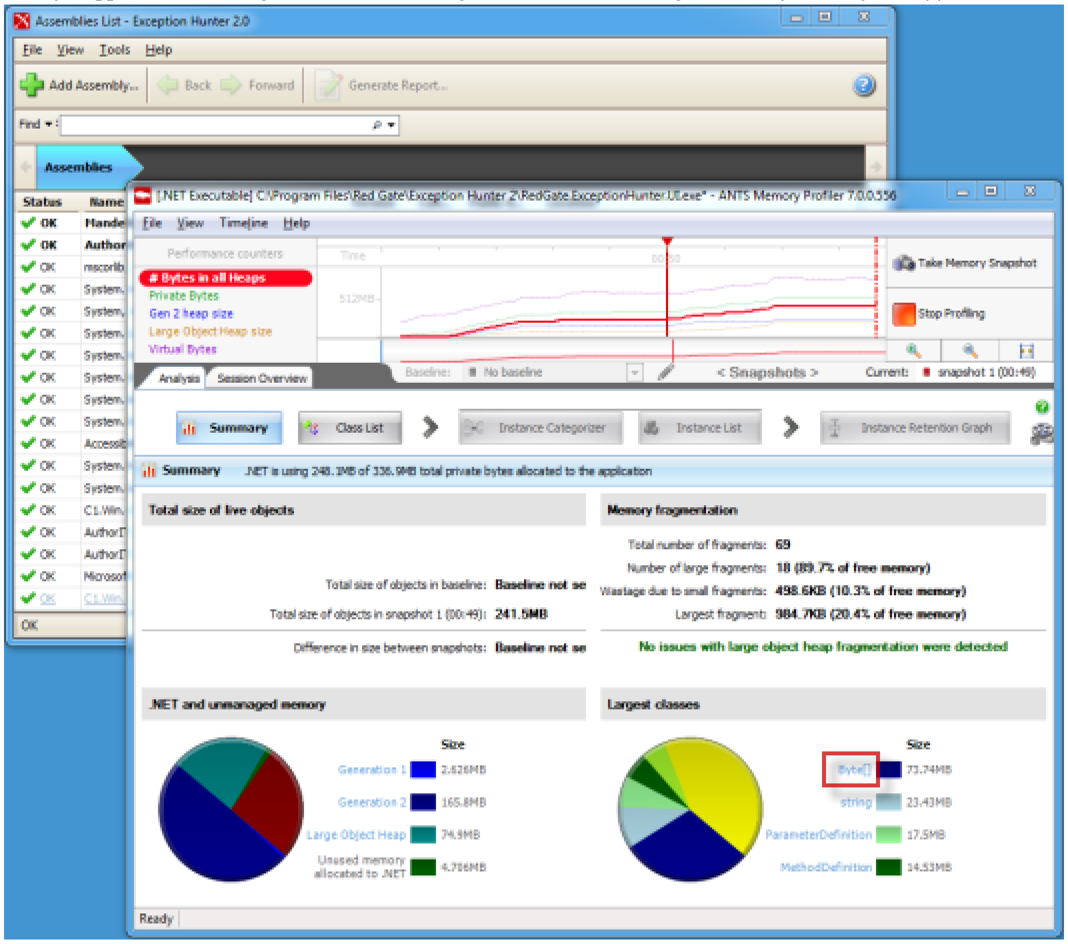
   , your application starts.
4. Snapshots are taken automatically at the trigger points you defined in your code. If required, you can still use the
   

   button to take snapshots manually as well.
5. Profiling results are shown in ANTS Memory Profiler.

Worked example

In this worked example, ANTS Memory Profiler is used to find out what uses most memory in Red Gate's Exception Hunter:

1. The `Byte[]` class is the largest class, accounting for 73.74MB of managed memory used by the application.



Summary .NET is using 248.1MB of 336.9MB total private bytes allocated to the application

Total size of live objects

Total size of objects in baseline: **Baseline not se**

Total size of objects in snapshot 1 (00:49): **241.5MB**

Difference in size between snapshots: **Baseline not se**

Memory fragmentation

Total number of fragments: **69**

Number of large fragments: **18 (89.7% of free memory)**

Wastage due to small fragments: **498.6KB (10.3% of free memory)**

Largest fragment: **984.7KB (20.4% of free memory)**

No issues with large object heap fragmentation were detected

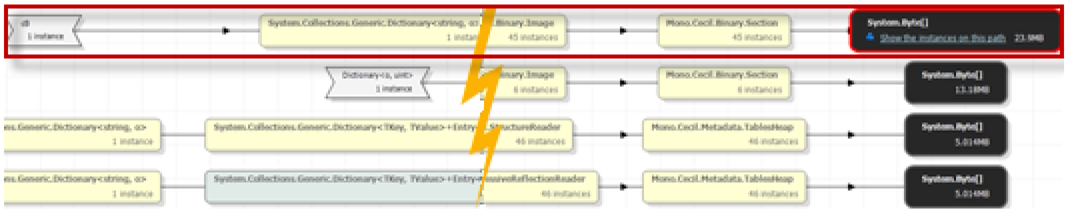
.NET and unmanaged memory

		Size
Generation 1	🟦	2.626MB
Generation 2	🟦	165.8MB
Large Object Heap	🟩	74.9MB
Unused memory allocated to .NET	🟩	4.716MB

Largest classes

		Size
Byte[]	🟦	73.74MB
string	🟦	23.43MB
ParameterDefinition	🟩	17.5MB
MethodDefinition	🟩	14.53MB

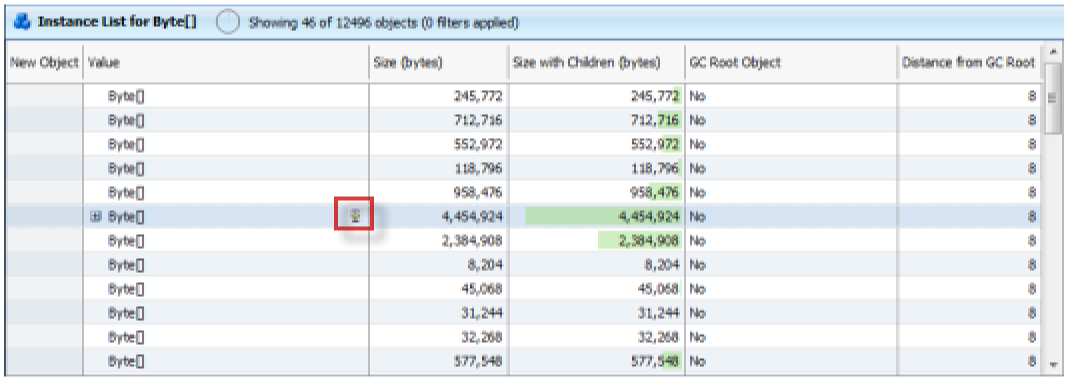
2. Click **Byte[]**.
3. The Instance Categorizer shows that most instances of Byte[] (totaling 23.9MB) are held on a path where the shortest route to a GC root object leads to an object called `cB`. On the `Byte[]` node on this path, click **Show instances on this path**.


cB
1 instance
System.Collections.Generic.Dictionary<string, cc>
1 instance
Binary.Image
45 instances
Mono.Cecil.Binary.Section
45 instances
System.Byte[]
Show the instances on this path
23.9MB
Dictionary<cB, uint>
1 instance
Binary.Image
6 instances
Mono.Cecil.Binary.Section
6 instances
System.Byte[]
13.18MB
Generic.Dictionary<string, cc>
1 instance
System.Collections.Generic.Dictionary<TKey, TValue>+Entry
StructureReader
46 instances
Mono.Cecil.Metadata.TableHeap
46 instances
System.Byte[]
5.014MB
Generic.Dictionary<string, cc>
1 instance
System.Collections.Generic.Dictionary<TKey, TValue>+Entry
ReflectionReader
46 instances
Mono.Cecil.Metadata.TableHeap
46 instances
System.Byte[]
5.014MB

4. The Instance List lists the 46 objects on this path. For instances of `System.String` it might be interesting to use the Instance List to look at the values of the strings, but this is unlikely to be the case for Byte[] instances, so choose any one and click

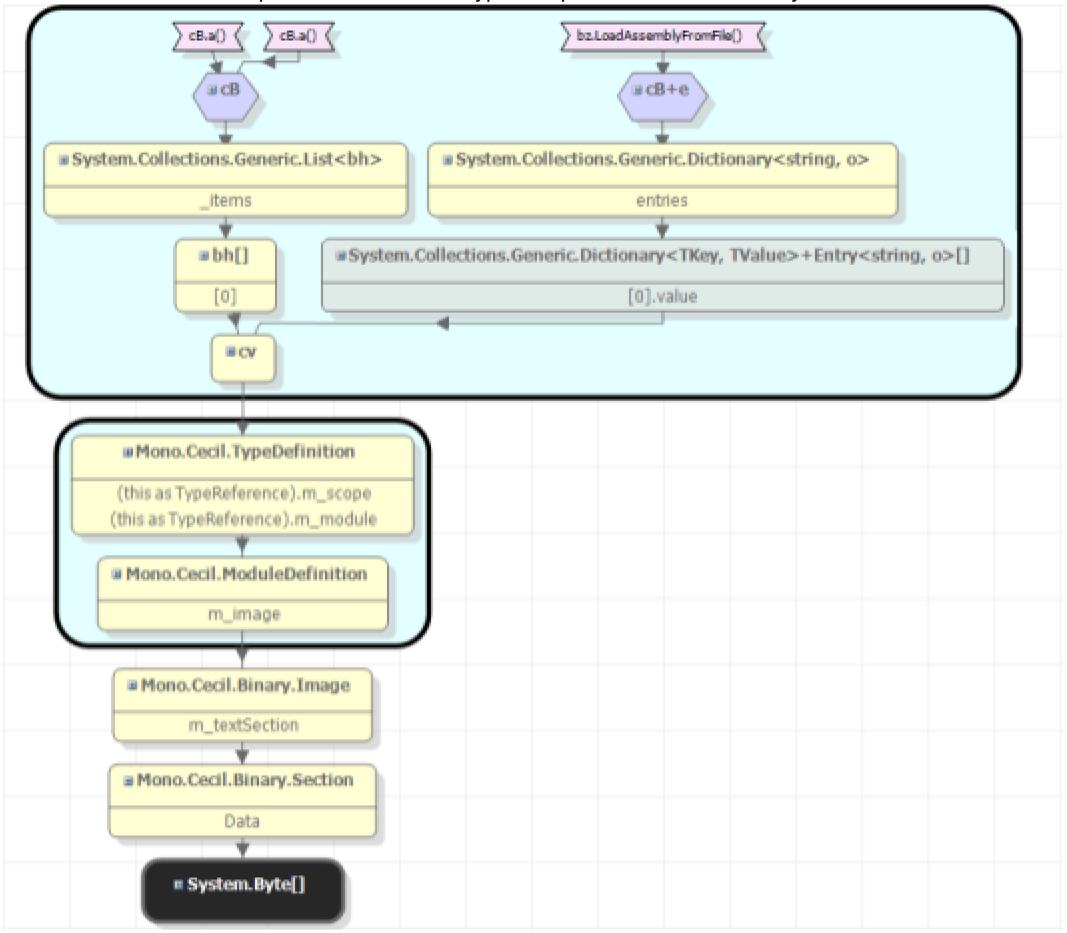
   .

   (All of the instances in the Instance List are all on the same shortest path, so it should make little difference which one you choose. Selecting the instance with the greatest size with children is often a sensible start, however.)


Instance List for Byte[] Showing 46 of 12496 objects (0 filters applied)

New Object	Value	Size (bytes)	Size with Children (bytes)	GC Root Object	Distance from GC Root
	Byte[]	245,772	245,772	No	8
	Byte[]	712,716	712,716	No	8
	Byte[]	552,972	552,972	No	8
	Byte[]	118,796	118,796	No	8
	Byte[]	958,476	958,476	No	8
	⊞ Byte[]	4,454,924	4,454,924	No	8
	Byte[]	2,384,908	2,384,908	No	8
	Byte[]	8,204	8,204	No	8
	Byte[]	45,068	45,068	No	8
	Byte[]	31,244	31,244	No	8
	Byte[]	32,268	32,268	No	8
	Byte[]	577,548	577,548	No	8

5. The Instance Retention Graph shows which other types keep this instance in memory.


cB.a()
cB.a()
bz.LoadAssemblyFromFile()
cB
cB+e
System.Collections.Generic.List<bh>
_items
System.Collections.Generic.Dictionary<string, o>
entries
bh[]
[0]
System.Collections.Generic.Dictionary<TKey, TValue>+Entry<string, o>[]
[0].value
cv
Mono.Cecil.TypeDefinition
(this as TypeReference).m_scope
(this as TypeReference).m_module
Mono.Cecil.ModuleDefinition
m_image
Mono.Cecil.Binary.Image
m_textSection
Mono.Cecil.Binary.Section
Data
System.Byte[]

Conclusions:

1. The `Byte[]` class accounts for most memory used by this program (73.74MB).
2. The largest number of objects from the `Byte[]` class (46 instances, totaling 23.9MB) are held on a path leading back to a GC root object from the class `cB`.
3. The selected instance is being kept in memory by `cv`, which in turn is being kept in memory by both `cB` and `cB+e`.
4. You now need to decide whether this should be the case, or whether there is a problem.

Video tutorials

Watch the video tutorials to see examples of ANTS Memory Profiler in action, and learn more about some areas of functionality:

- Overview of ANTS Memory Profiler
- Understanding the summary screen
- How to use the filters
- Using the instance categorizer

Troubleshooting

Common issues

- Attach to process unavailable with some anti-virus software
- Memory leaks observed when profiling WPF applications
- Silverlight in-browser profiling stops with no results
- Silverlight out-of-browser profiling stops with no results
- Troubleshooting SharePoint profiling
- Troubleshooting the Visual Studio add-in

Error messages

- Couldn't open metabase
- Error stopping IISAdmin profiling IIS web application on Windows XP
- Failed to CoCreate Profiler
- No Disk
- Operation could destabilize the runtime
- The COMplus application has not loaded the .NET Framework
- The snapshot failed because .NET did not report every referenced object
- The system cannot find the file specified
- The type initializer for 'y.layout.hierarchic.ClassicLayerSequencer' threw an exception
- Unable to connect to server

Unexpected behavior and technical questions

- Profiler prompts for location of source code which is not your own source code
- Profiling a Windows service fails if the service uses a system account
- Profiling Microsoft Office managed-code add-ins
- Rebuilding performance counters
- Total size of all objects does not match the memory footprint of an application

Common issues

- Attach to process unavailable with some anti-virus software
- Memory leaks observed when profiling WPF applications
- Silverlight in-browser profiling stops with no results
- Silverlight out-of-browser profiling stops with no results
- Troubleshooting SharePoint profiling
- Troubleshooting the Visual Studio add-in

Attach to process unavailable with some anti-virus software

When using ANTS Memory Profiler or ANTS Performance Profiler, you may find that you cannot attach to any .NET 4 process.

Cause / Possible causes

This can be caused by anti-virus software.

To create the list of .NET 4 processes you can attach to, the profiler checks all running processes to check whether they are managed processes. For managed processes, the profiler obtains the CLR version number.

Processes that are part of some anti-virus programs protect themselves by changing their entries in the Access Control List. If a running process is protected in this way, the profiler cannot check the CLR version of that process, or of any other running processes.

How to fix

To work around this problem, uninstall your anti-virus software while profiling.

If you continue to experience problems, please contact Redgate support.

Memory leaks observed when profiling WPF applications

When profiling some applications based on the Windows Presentation Framework (involving controls with collapsed visibility), the private bytes count may increase continually and never decrease.

This is caused a known bug in WPF for which Microsoft has released a hotfix. For more information, see Microsoft Knowledge Base article 967328. You can find a full description of the issue on Ramon de Klein's blog.

Silverlight in-browser profiling stops with no results

When profiling a Silverlight application running in Internet Explorer, ANTS Memory Profiler may disconnect from the session and start summarizing results.

How to fix

This can be caused by a number of different issues:

Cause	Resolution
An instance of *iexplore.exe* is already running in the background, and the Profiler attaches to that instance.	Use Task Manager to check that no instances of *iexplore.exe* are running.
Internet Explorer restarts in 'protected mode'.	Ensure your Internet Explorer settings allow it to run as administrator.
Javascript in the application closes the web browser window.	Disable the Javascript during profiling.

In other cases, if the Silverlight application is configured to run out-of-browser, profile the application in out-of-browser mode.

If you continue to experience problems, please contact Redgate Support.

Silverlight out-of-browser profiling stops with no results

When profiling an out-of-browser Silverlight application, the profiling session may stop soon after it starts, leaving the application still running. The following message or similar will appear in the ANTS Memory Profiler log file:

```
[Profiler pipe monitor] WARN RedGate.Profiler.Engine.Startup.Basic.PipeConnection - Rejected a connection
attempt from process ID 0x978 'C:\Program Files (x86)\Microsoft Silverlight\sllauncher.exe'
```

Microsoft had released a security patch to Silverlight in October 2011 that changed the way the out-of-browser Silverlight launcher starts and connects to the profiling API. The sllauncher.exe process seems to Profiler as if it starts and exits immediately, and a new process appears that Profiler will not allow to connect.

Until this problem can be addressed there is a workaround available, which is to profile cmd.exe and launch the Silverlight application in a console session.

1. Instead of selecting the option to profile a Silverlight application, use the option to profile a .NET executable.
2. Ensure the **profile child processes** option is enabled
3. Set the executable as **c:\windows\system32\cmd.exe**
4. Set the arguments to be **/K ""c:\program files (x86)\microsoft silverlight\sllauncher.exe"/emulate:"c:\YourOOBAppFolder\YourOOBApp.xap" /origin:"c:\YourOOBAppFolder" /overwrite"**
   Where "YourOOBApp" is the folder the XAP resides in and "YourOOBApp.xap" is the out-of-browser Silverlight application

The /K option to cmd will keep the console session and allow ANTS Memory Profiler to attach to any process started in the console session.

Troubleshooting SharePoint profiling

If you encounter difficulties, the most likely cause is that ANTS Memory Profiler cannot read from the directory to which SharePoint is writing data. 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 Profiling SharePoint with ANTS Performance Profiler 5.2. (The information in this post also applies to ANTS Memory Profiler.)

Troubleshooting problems caused by SharePoint security settings

Security features in ASP.NET may cause problems on some systems. In that case, follow the instructions below:

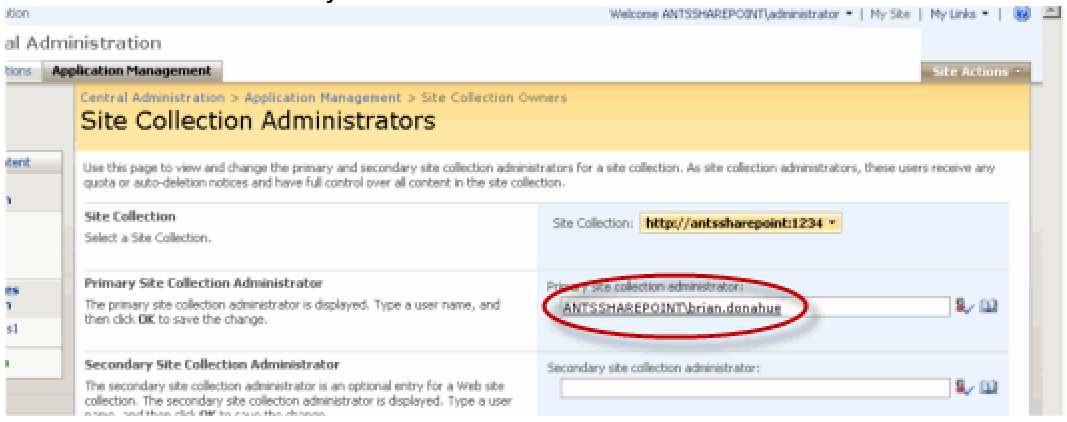
Information you will need

Before profiling a SharePoint 2007 site, you will need to know the following information for the site collection you want to profile:

- the URL for the site collection
- the TCP port it runs on
- 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 **SharePoint Site Management**, click **Site Collection Administrators**.
4. From the dropdown list, select the name of the site collection hosting your web part.
5. Note the account set in the **Primary Site Collection Administrator** box.


Welcome ANTSSHAREPOINT\administrator | My Site | My Links
al Administration
Application Management
Site Actions
Central Administration > Application Management > Site Collection Owners

Site Collection Administrators

Use this page to view and change the primary and secondary site collection administrators for a site collection. As site collection administrators, these users receive any quota or auto-deletion notices and have full control over all content in the site collection.

Site Collection
Select a Site Collection.

Site Collection: http://antssharepoint:1234

Primary Site Collection Administrator
The primary site collection administrator is displayed. Type a user name, and then click OK to save the change.

Primary site collection administrator: ANTSSHAREPOINT\brian.donahue

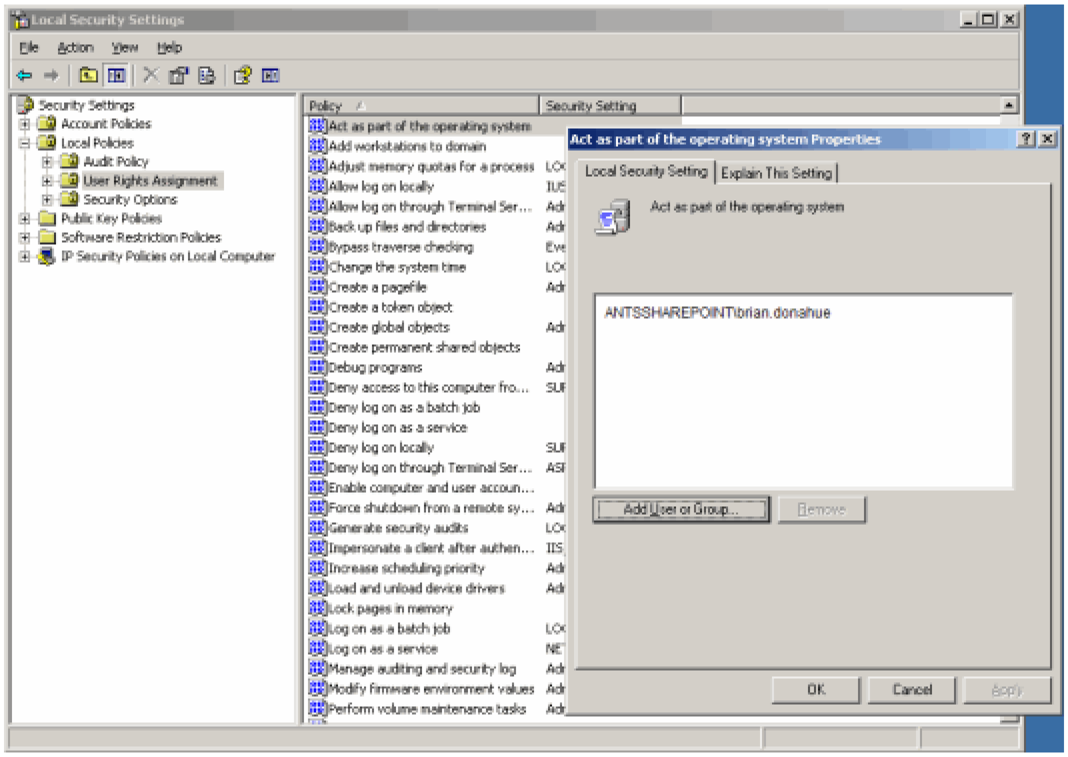
Secondary Site Collection Administrator
The secondary site collection administrator is an optional entry for a Web site collection. The secondary site collection administrator is displayed. Type a user

Secondary site collection administrator:

Grant permissions

The primary site collection administrator must have permission to launch an IIS 6 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.



Policy	Security Setting
Act as part of the operating system	
Add workstations to domain	
Adjust memory quotas for a process	LO
Allow log on locally	IUS
Allow log on through Terminal Ser...	Adr
Back up files and directories	Adr
Bypass traverse checking	Eve
Change the system time	LO
Create a pagefile	Adr
Create a token object	
Create global objects	Adr
Create permanent shared objects	
Debug programs	Adr
Deny access to this computer fro...	SUF
Deny log on as a batch job	
Deny log on as a service	
Deny log on locally	SUF
Deny log on through Terminal Ser...	ASF
Enable computer and user accoun...	
Force shutdown from a remote sy...	Adr
Generate security audits	LO
Impersonate a client after authen...	IIS
Increase scheduling priority	Adr
Load and unload device drivers	Adr
Lock pages in memory	
Log on as a batch job	LO
Log on as a service	NE
Manage auditing and security log	Adr
Modify firmware environment values	Adr
Perform volume maintenance tasks	Adr

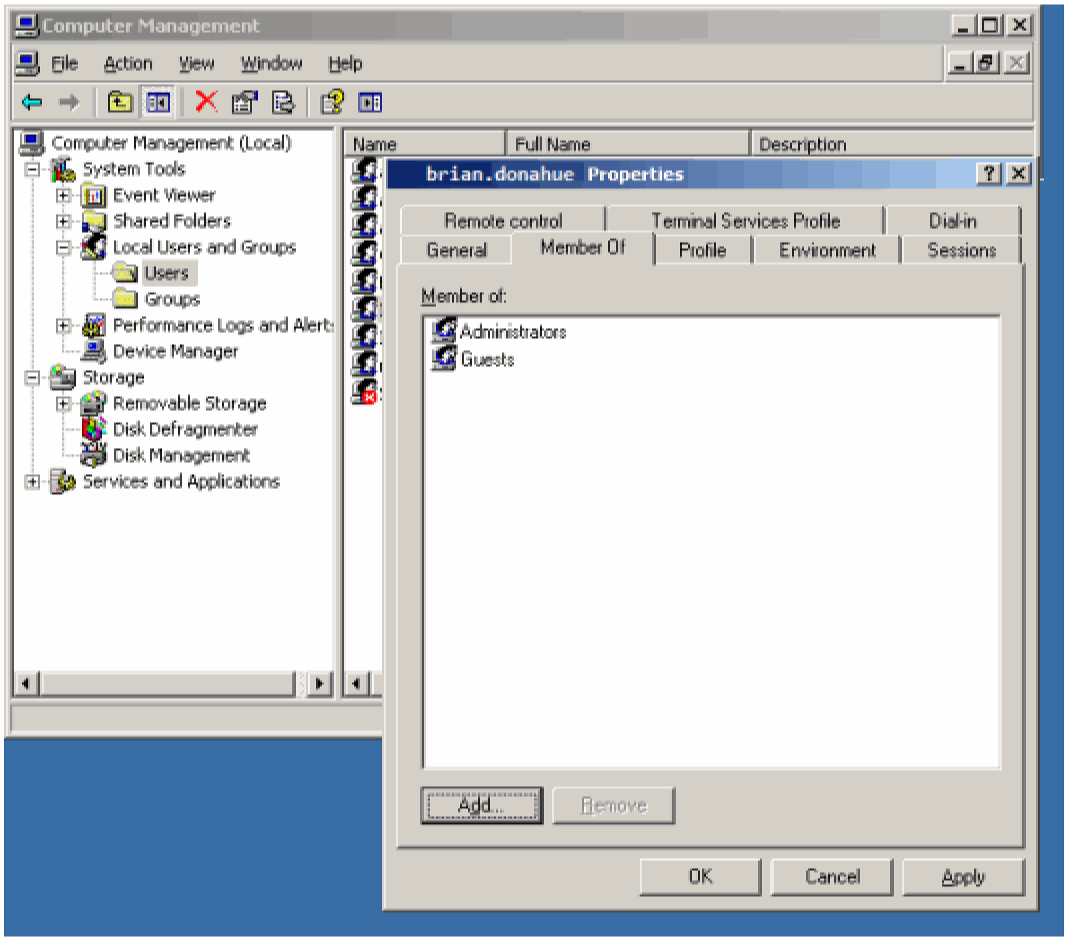
Act as part of the operating system

ANTSSHAREPOINT\brian.donahue

4. Double-click **Impersonate a client after authentication** and add the primary site collection administrator's account.
5. Open a command prompt and run:

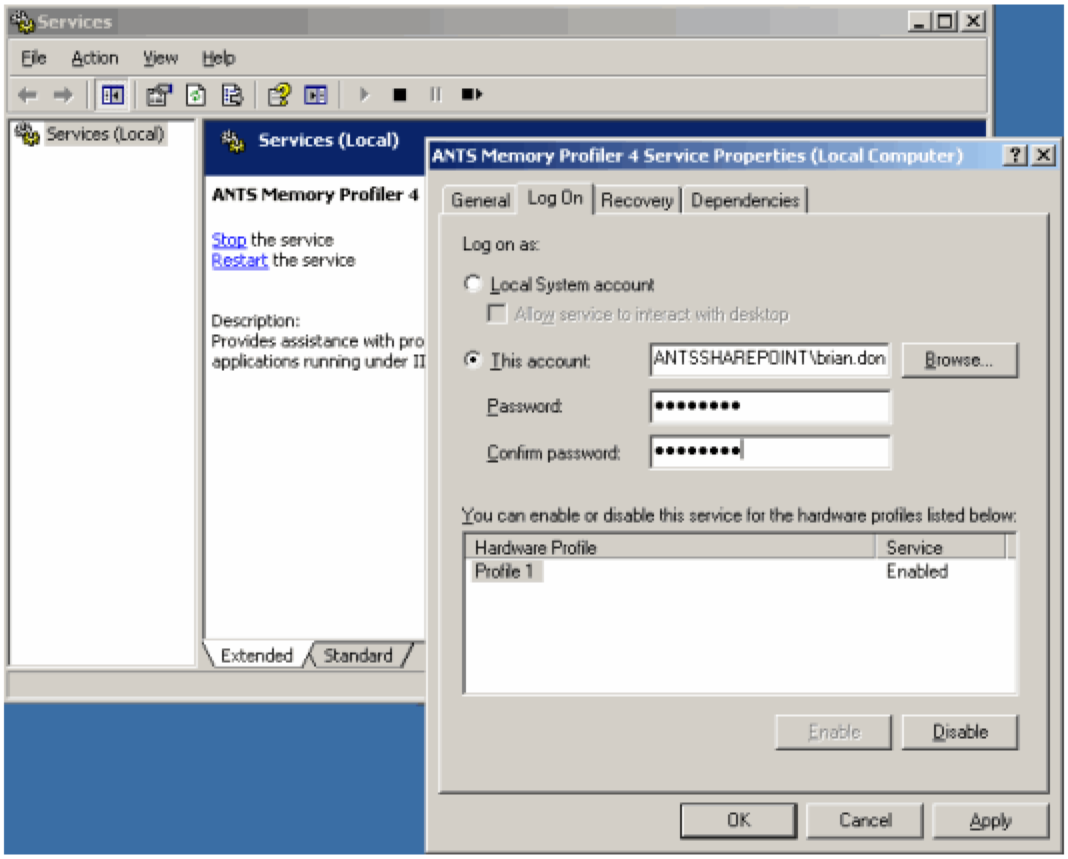
```
gpupdate /force
```

   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 Memory Profiler 7.4 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 the **ANTS Memory Profiler 7.4 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 Memory Profiler 7.4 Service** is *Started*, right-click the service and click **Restart**.


Services (Local)

ANTS Memory Profiler 4

Stop the service
Restart the service

Description:
Provides assistance with pro
applications running under II

ANTS Memory Profiler 4 Service Properties (Local Computer)

Log on as:

Local System account

Allow service to interact with desktop

[✓ This account:] ANTSSHAREPOINT\brian.don

Password: •••••••••

Confirm password: •••••••••

You can enable or disable this service for the hardware profiles listed below:

Hardware Profile	Service
Profile 1	Enabled

Ensure that compilation will be in DEBUG configuration (IIS 6)

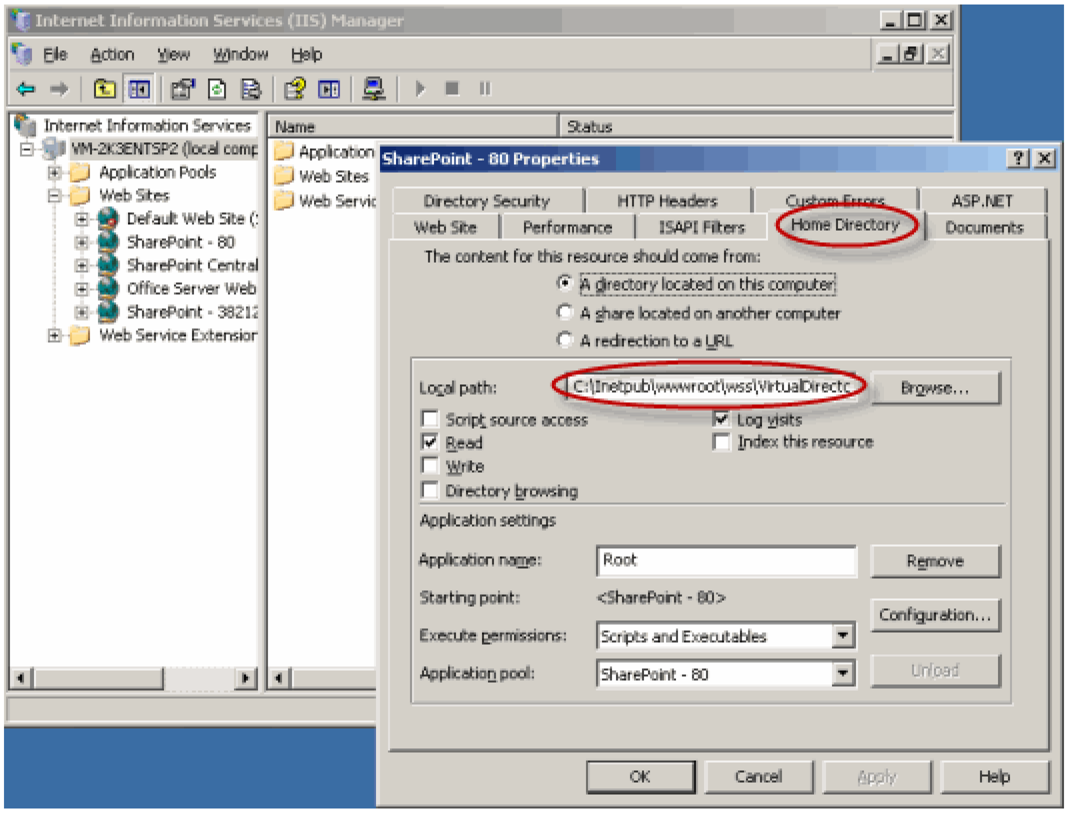
(Instructions for IIS 7 are below)

To profile a SharePoint collection, the ASP .NET compilation must be done in DEBUG configuration. This will allow ANTS Memory Profiler to locate the source code for any web parts or other extensions you have written for the site collection. DEBUG configuration also removes 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.



SharePoint - 80 Properties

The content for this resource should come from:

- (●) A directory located on this computer
- ( ) A share located on another computer
- ( ) A redirection to a URL

Local path: C:\Inetpub\wwwroot\wss\VirtualDirecto

- [ ] Script source access
- [x] Read
- [ ] Write
- [ ] Directory browsing
- [x] Log visits
- [ ] Index this resource

Application settings

Application name: Root

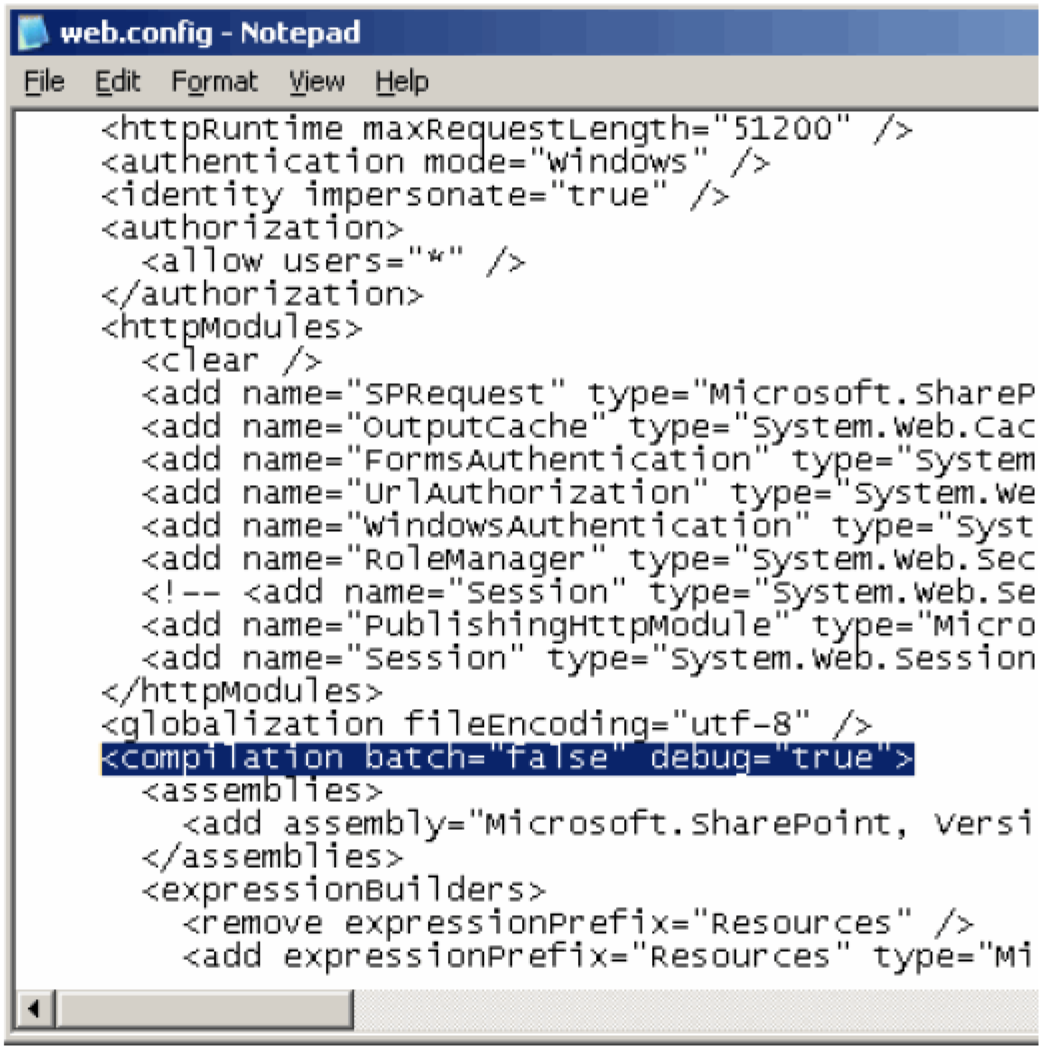
Starting point: <SharePoint - 80>

Execute permissions: Scripts and Executables

Application pool: SharePoint - 80

You must now locate and edit the web.config file for the site collection using notepad.exe (or other text-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 a text editor. Search for the text '**Debug**'.
4. Change
   `<compilation batch="false" debug="false">`
   to
   `<compilation batch="false" debug="true">`.
5. Save the file.



```
    <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
```

Continue reading at the step 'Copy PDBs and web part DLLs to the app_bin directory' below.

Ensure that compilation will be in DEBUG configuration (IIS 7)

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*.

Copy PDBs and web part DLLs to the app_bin directory

To filter out all methods except those run by your code when viewing the profiling results, you must copy the relevant files into the site's app_bin directory. To do this:

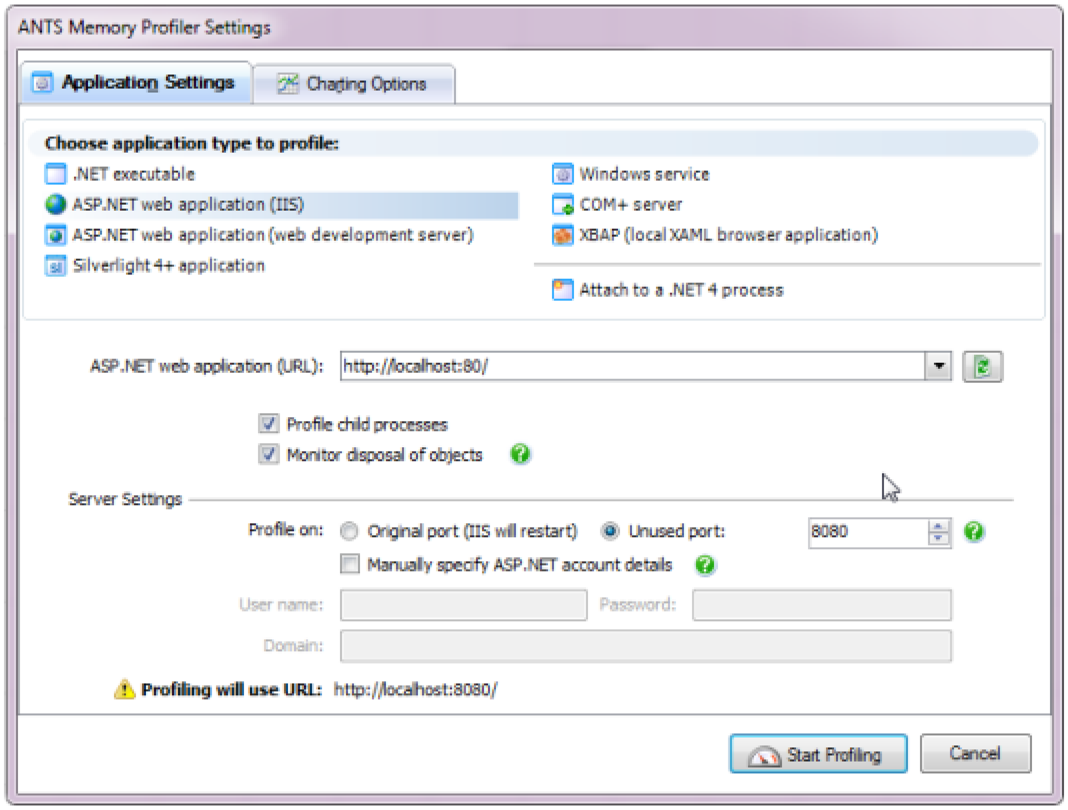
1. Copy all PDB files and any web part DLLs used by your site to the Clipboard.
2. Use Windows Explorer to navigate to the site collection root's physical path.
3. Open the app_bin directory.
4. Paste all PDB files and any web part DLLs used by your site into this directory.

Start profiling

1. If the Internet Information Services Manager is not already displayed, open **Administrative Tools** and open **Internet Information Server (IIS) Manager**.
2. Stop the website.
3. Start ANTS Memory Profiler.
4. Under **Choose application type to profile**, click **ASP .NET Web Application (hosted in IIS)**.
5. 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 that the web application normally runs on. If the site collection is on the root virtual directory for the site, you must include the trailing slash.

6. To profile your application without restarting IIS select **Unused port** and enter a new port number. If your application uses hard-coded port numbers, your application will not work and you must chose **Original port** instead.
7. 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 Memory Profiler results if these objects have been accessed.


ANTS Memory Profiler Settings

Choose application type to profile:

- .NET executable
- ASP.NET web application (IIS) ✓
- ASP.NET web application (web development server)
- Silverlight 4+ application
- Windows service
- COM+ server
- XBAP (local XAML browser application)
- Attach to a .NET 4 process

ASP.NET web application (URL): http://localhost:80/

[✓] Profile child processes
[✓] Monitor disposal of objects

Server Settings

Profile on: ( ) Original port (IIS will restart) [✓ Unused port:] 8080

[ ] Manually specify ASP.NET account details

User name:
Password:
Domain:

Profiling will use URL: http://localhost:8080/

Troubleshooting the Visual Studio add-in

If you experience problems with the ANTS Visual Studio add-in, the following information might help:

- 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 Memory Profiler.
- The Visual Studio add-in is a separate program from ANTS Memory Profiler, and is installed by default as part of Red Gate's .NET bundles. If you purchased ANTS Memory Profiler as a standalone product, you might not have the add-in. In that case, download one of the free .NET bundle trials from the Redgate 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 is not limited to the trial period.

Error messages

- Couldn't open metabase
- Error stopping IISAdmin profiling IIS web application on Windows XP
- Failed to CoCreate Profiler
- No Disk
- Operation could destabilize the runtime
- The COMplus application has not loaded the .NET Framework
- The snapshot failed because .NET did not report every referenced object
- The system cannot find the file specified
- The type initializer for 'y.layout.hierarchic.ClassicLayerSequencer' threw an exception
- Unable to connect to server

Couldn't open metabase

This article relates to both ANTS Performance Profiler and ANTS Memory Profiler.

When profiling an ASP.NET web application hosted in IIS, an exception may occur at the start of profiling. Clicking **Show Details** reveals the following information, as well as some stack traces:

Could not start IIS.
Couldn't open metabase - Please ensure that IIS is installed

This error can happen when you are logged into the computer with an account that has reduced privileges.

To correct this:

1. Log out of the computer and log back in using an Administrator account, or one that has administrative privileges on the machine.
2. Open a command prompt, and change the working directory to the installation directory of the version of Microsoft .NET Framework that the ASP.NET application uses as its runtime.
   For instance, if the web application that you want to profile uses ASP .NET 2.0:

   ```
   cd \
   cd %systemroot%\microsoft.net\framework\v2.0.50727
   ```

3. Run the ASPNET_REGIIS utility, which will grant permissions to the IIS metabase and ASP .NET temporary files locations. To allow a domain user called MyDomain\MyUser, this would be the correct command:
   `aspnet_regiis -ga MyDomain\MyUser`

After logging the administrator account off and logging your low-privilege user back on, the account should now have access to IIS and ASP .NET appropriate for profiling the web application hosted in IIS.

Error stopping IISAdmin profiling IIS web application on Windows XP

This article relates to both ANTS Performance Profiler and ANTS Memory Profiler.

When profiling a web application hosted in IIS 5.x (Windows XP), the following error may occur before code profiling actually takes place:
Error stopping IISAdmin

Because the error occurs while stopping and starting IIS, the error message also covers a failure to START IISAdmin as well, so it is a good idea to check your application event log at this point. ANTS Performance/Memory Profiler will log the error actually returned by Windows at that location.

If the event log contains this entry:
System.Exception: Timed out starting 'IISADMIN'
the problem is actually a failure to start the service due to a timeout. In some cases, the default service timeout setting for Windows does not allow enough time for the service to start.

To increase the timeout, edit the following registry key:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\WaitToKillServiceTimeout

The value of this registry key contains the number of milliseconds that Windows will wait for a service to start, the default being 30000 (30 seconds). Increasing this to 60000 (60 seconds) or slightly higher may allow the IISAdmin service to start successfully.

If you change the registry key, restart the computer so that the changes will take effect.

Failed to CoCreate Profiler

This article applies to both ANTS Performance Profiler and ANTS Memory Profiler.

If ANTS Performance/Memory Profiler is used to profile a Windows Service, it modifies the running environment of the service temporarily, then undoes the changes when profiling is stopped. If the service crashes with an unhandled exception during profiling, however, the modified environment may be left behind and the service will be permanently 'hooked' into ANTS Performance/Memory Profiler so the service will attempt to load the Profiler's "core" component at service startup time. Because ANTS Performance/Memory Profiler is not running, the following error entry is written to the application event log:

```
Source: .NET Runtime
Category: None
Event ID: 1022
Description: .NET Runtime version 2.0.50727.832 - Failed to CoCreate profiler.
```

In order to restore the normal environment of the service, the registry editor must be used. Click the start bar, then 'run', then type regedit and press enter.

Look for the short name of your service in the key:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Open the key that has the same name as your service, and in the right-hand pane, right-click Environment and select Modify.

Locate the entries *COR_ENABLE_PROFILING=1* and *COR_PROFILER={a GUID}* and remove these name/value pairs from the registry value. When the service is restarted, the error should no longer occur.

No Disk

When profiling an application, ANTS Memory Profiler may show an error message similar to:

```
MyApplication - No Disk
There is no disk in the drive.Please insert a disk into drive
\Device\Harddisk1\DR2
```

Cause

This can happen when the source code location matches a removable drive that has been disconnected. It does not matter if the software was built on a completely different computer - if the drive letter is the same as a disconnected removable drive on the computer used for profiling, this error will occur.

How to fix

There are a few possible ways to fix this:

- Reconnect the removable drive
- Disconnect the removable device that normally mounts the drive
- Change the drive letter using Windows Computer Management Console (Disk Management)
- Delete the PDB files applicable to the code you are profiling

If you continue to experience problems, please contact Redgate Support.

Operation could destabilize the runtime

When profiling an ASP .NET web application, the following error may appear in the web browser window:

Operation could destabilize the runtime This error occurs sporadically depending on the nature of the code being run in the website process and the causes of this message can be varied. This is because the error concerns a failure by the runtime to validate Intermediate Language code at a very low-level.

One possible fix for this problem is to enable full trust for the web application. To do this, open the web.config file in the web application's root directory and change the trust level, for instance:

```
<configuration>
 <system.web>
        <trust level="Full"/>
```

The COMplus application has not loaded the .NET Framework

In order for ANTS Memory Profiler to profile a COM+ application, the application must be started in the context of the logged-in user. The security architecture of COM+ does not allow a COM+ application to be started from a Remote Desktop or terminal services session, so the COM+ application will start and then exit almost immediately.

If it is absolutely necessary to use ANTS Memory Profiler against a COM+ application remotely, it is possible to run the remote session in console mode and allow ANTS Memory Profiler to work effectively. This does have the side-effect of logging off any users who are already logged on interactively, so be prepared to ensure no other users are logged on to the computer to prevent them from losing their work.

To start a remote desktop session in console mode, select Start, then Run, and type 'mstsc /console'.

The snapshot failed because .NET did not report every referenced object

The full text of this error message is:

> 'The snapshot failed because .NET did not report every referenced object. If the garbage collector is operating in 'server mode' and the target application has many large objects, a bug in version 2 of the CLR is likely to be the cause.
>
> To work around this bug, manually switch your application to the workstation garbage collector. Search for 'gcServer configuration option' at msdn.microsoft.com for information on how to do this.'

Cause / Possible causes

This message occurs because of a bug in .NET version 2.

How to fix

To work around the problem, you must switch your application to the workstation garbage collector. To do this:

For ASP.NET web applications:

For web applications, you need to make the same change in ASP.NET's configuration file on the server.

- On 32-bit systems, the file is usually located at *C:\Windows\Microsoft.NET\Framework\v2.0.50727\Aspnet.config*
- On 64-bit systems, the file is usually located at *C:\Windows\Microsoft.NET\Framework64\v2.0.50727\Aspnet.config*

Edit this file with an XML editor, adding `<gcServer enabled="false" />` as a child of the `<runtime>` node.

For .NET executables, Windows Services, COM+ Server applications and XBAPs:

1. Navigate to the application.
2. Using an XML editor, open the file *myApplication.exe.config*, where *myApplication.exe* is the name of your application. If the **.exe.config* f ile does not exist, create it.
3. Set the <gcServer> node to *enabled="false"*, as follows:

```
<configuration>
  <runtime>
    <gcServer enabled="false" />
  </runtime>
</configuration>
```

4. Save the file. There is no need to rebuild your application because the configuration file will be loaded automatically by .NET when the application runs.

If you continue to experience problems, please contact Redgate Support.

The system cannot find the file specified

This page relates to both ANTS Performance Profiler and ANTS Memory Profiler.

When profiling an ASP .NET web application, ANTS Performance/Memory Profiler may throw the following exception:

```
Unable to start profiler - exception details
System.ComponentModel.Win32Exception: The system cannot find the file specified
```

ANTS Performance/Memory Profiler relies on an number of programs to be installed on the system to work properly. Some are web browser components and some are to verify that application pools are running. In short, this error can occur if one of the following files are missing:

- *%systemroot%\shdocvw.dll*
- *w3wp.exe*, the path to which is found in the registry at *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\IISAdmin\ImagePath*, substituting *w3wp.exe* for *intetinfo.exe*
- **In IIS 6 only:** *%systemroot%\system32\cscript.exe*
- **In IIS 6 only:** *%systemroot%\system32\iisapp.exe*
- **In IIS 7 only:** *appcmd.exe*, located in the same folder as *w3wp.exe* above

The type initializer for 'y.layout.hierarchic.ClassicLayerSequencer' threw an exception

This error occurs if the Microsoft Visual J# Redistributable was not correctly installed during the installation of ANTS Performance Profiler or ANTS Memory Profiler.

To repair Microsoft Visual J# Redistributable

1. In **Control Panel**, select **Uninstall a program**.
2. Select **Microsoft Visual J# Redistributable**.
3. Click **Repair**.

To install a new copy of Microsoft Visual J# Redistributable

Download and install the appropriate redistributable for your system from the Microsoft website: (64-bit systems or 32-bit systems)

Unable to connect to server

When attempting to profile a web application, the browser may show an error similar to "could not connect to the remote server" after the browser is launched by ANTS Memory Profiler.

Cause / Possible causes

When Internet Explorer runs in Protected Mode, it may launch an instance of Internet Explorer that quickly exits and restarts before showing the website. This makes ANTS Memory Profiler believe that profiling had been completed, making it shut the site down.

How to fix

If your web browser uses "Protected Mode" you may try adding "localhost" to the list of trusted sites as a workaround.

If you continue to experience problems, please contact Redgate Support.

Unexpected behavior and technical questions

- Profiler prompts for location of source code which is not your own source code
- Profiling a Windows service fails if the service uses a system account
- Profiling Microsoft Office managed-code add-ins
- Rebuilding performance counters
- Total size of all objects does not match the memory footprint of an application

Profiler prompts for location of source code which is not your own source code

This article relates to both ANTS Performance Profiler and ANTS Memory Profiler.

ANTS Performance/Memory Profiler may prompt for the location of a source code file that is not your own source code when viewing the results.

When ANTS Performance/Memory Profiler produces performance profiling results in detailed mode or during memory profiling, information is collected about the assembly being profiled, including the method and object names, as well as the path to the source code files for these items. In some methods internal to the Microsoft .NET Framework, assemblies are compiled dynamically from source code internally without your knowledge. ANTS Performance/Memory Profiler also adds these methods and the source code file to its result set. Because the source code is created, compiled, and disposed of by the .NET Framework, it is no longer available and clicking on some internal Framework objects will trigger ANTS Performance/Memory Profiler to ask for the new source code location because the source code used to create these temporary assemblies no longer exists.

When this happens, the source code file name being requested is normally in the form of a globally unique identifier, for instance 'aaaaaaaa-bbbb-bbbb-bbbb-cccccccccccc.vb'

This message can be safely ignored. Alternatively, .NET Reflector can be used to decompile source code from the executable.

Profiling a Windows service fails if the service uses a system account

Profiling a Windows service may fail if the service runs under the *System*, *LocalService* or *NetworkService* account. If the failure occurs, profiling will stop shortly after starting and the service will not run.

The message "System.ServiceProcess.TimeoutException: Time out has expired and the operation has not been completed" may be written to the profiler log.

To work around this problem, run the service under a specific user account instead of a predefined system account.

This failure happens because ANTS Memory Profiler sets a registry value when it enables profiling: if the service runs as *System*, *LocalService* or *NetworkService*, the profiler cannot acquire the correct value, and sets an erroneous value instead. The service then uses the erroneous registry value to generate a write path: because the value is incorrect, the path is unusable, and the service fails. Running under a specific user account ensures ANTS Memory Profiler can read the system user environment variable, and will set the correct registry value.

Profiling Microsoft Office managed-code add-ins

This article relates to both ANTS Performance Profiler and ANTS Memory Profiler.

ANTS Performance/Memory Profiler can profile an add-in produced for any Microsoft Office application using Visual Studio .NET. Add-ins are typically implemented as a dll assembly loaded into the relevant Office application's process. Therefore, you would profile the application that loads your dll in order to get the performance and memory usage data for it from ANTS Performance/Memory Profiler.

For instance, to profile the performance of an add-in for Microsoft Excel, you would:

1. Start the profiler
2. Choose .NET Desktop Application.
3. Browse for and select Excel.exe (*c:\program files\Microsoft Office\OFFICE12\Excel.exe*).
4. Choose to profile **Only .NET methods that have source code** if your assembly has a corresponding PDB file, or choose another option.
5. Excel will start; perform whatever actions are necessary to invoke methods in your add-in.
6. Take snapshots (ANTS Memory Profiler) or view the performance profiling results in ANTS Performance Profiler.

The results reflect the managed code performance of your add-in's .NET code.

Rebuilding performance counters

In certain situations, the performance counters may be missing from the performance counter results during a profiler session.

There are many possible causes for the problem, but they all can usually be fixed by rebuilding the performance counter objects.

To rebuild your performance counters:

- If you're running **Windows 2000, Windows XP** or **Windows Server 2003**, to rebuild the **Windows performance counters**, open a command prompt and run the following commands:

  ```
  cd %systemroot%\system32
  lodctr /R
  ```

  and then reboot the computer. Other performance counters may exist outside the *system32* folder.
  If you need to rebuild the **.NET performance counters**, locate the file *corperfmonsymbols.ini*. This should reside in the %systemroot%\microsoft.net\framework\v2.0.50727 directory, but may be in %systemroot%\inf\.netFramework as well. Use lodctr against this file to rebuild the performance counters from a command prompt by running the following commands:

  ```
  unlodctr .NetFramework
   cd %systemroot\inf\.netFramework (or the folder containing
  corperfmonsymbols.ini)
   lodctr corperfmonsymbols.ini
  ```

  For more information, see How to manually rebuild Performance Counter Library values (Microsoft KB)

  It is also possible that performance objects are disabled and therefore unavailable to query.

  The *EXCTRLST.exe* tool from Microsoft will show you whether or not a particular performance object is enabled and allow you to enable it. This tool can be downloaded for free from Microsoft

  If any of the performance counters are not enabled, they will not appear in the ANTS Profiler performance counter graph.

  On some versions of Windows, performance counters can be disabled globally by setting the registry value HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib to "1".
- If you're on a **newer operating system**, please follow the following instructions:
  1. Open a command prompt as Administrator
  2. Enter the following commands:

     ```
     C:
     cd windows\system32
     LodCtr.exe /S:Backup_Original.INI
     LodCtr.exe /R:PerfStringBackup.INI
     ```

     (The /S: switch lets you backup the performance counters before you rebuild with the /R: switch.)

     Use *perfmon* to check the counters display correctly.

For more information see How to rebuild performance counters (TechNet)

Total size of all objects does not match the memory footprint of an application

There is usually a disparity between the total size of all objects allocated on the managed heap (reported by ANTS Memory Profiler) and the amount of memory shown to be used by the entire process, for example the memory shown in Windows Task Manager.

This is not a bug in ANTS Memory Profiler, but rather a difference in the type of memory being tracked by ANTS Memory Profiler.

The total memory used by a .NET application comprises:

- the application image itself
- the .NET runtime
- any application extensions (dlls) loaded by the process being profiled.

These items are not tracked by ANTS Profiler and therefore are not reflected in the results.

Additionally, .NET memory management typically allocates more heap memory than what is actually requested by your application. Because memory allocation is expensive, .NET will request extra memory for the managed heap and work inside this memory rather than continually requesting more memory from the operating system in smaller increments. For this reason, even the process managed heap size reported by the ".NET CLR Memory" performance object may be larger than what is reported by ANTS Memory Profiler.

In order to see really specific usage information about the memory of a process, the most accurate way is to use a debugger. Redgate have produced a free tool that graphically displays a .NET application's memory space. Memory Tracker can illustrate some of the memory issues discussed in this article. You may download this tool from Redgate Labs

Finding the ANTS Memory Profiler log files

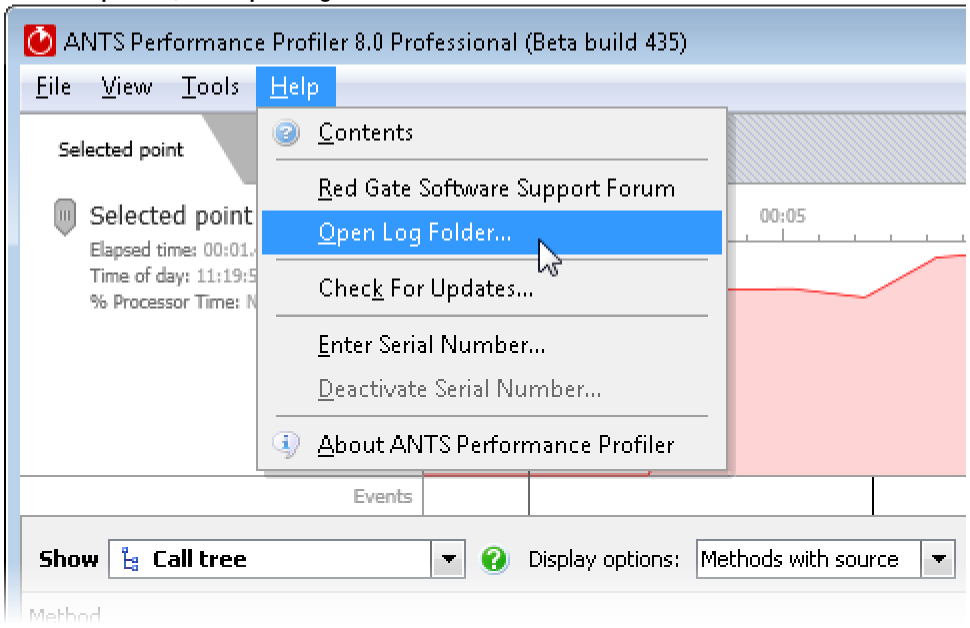
The information on this page relates to ANTS Performance Profiler, ANTS Memory Profiler and Exception Hunter only.

For other products, see Log file locations

Log files collect information about the application while you are using it. These files are useful to us if you have encountered a problem.

To open the folder where the log files are stored:

- On the **Help** menu, click **Open Log Folder**:



By default the log files are stored in:

%ALLUSERSPROFILE%\Local Settings\Application Data\Red Gate\ProductName

If you are profiling an ASP.NET application in IIS, additional log files for the service and the trigger process are created. These are stored by default in:

%LOCALAPPDATA%\Red Gate\ProductName

The local app data folder used is that of the user who launched the product.

Release notes and other versions

Version 8.8 (current)	October 13th, 2015 (latest)	Release notes	Documentation
Version 8.7	July 17th, 2015	Release notes	
Version 8.6	April 24th, 2015	Release notes	
Version 8.5	February 18th, 2015	Release notes	
Version 8.2	December 19th, 2013	Release notes	
Version 8.1	November 5th, 2013	Release notes	
Version 8.0	October 2nd, 2013	Release notes	
Version 7.4	April 5th, 2012	Release notes	Documentation
Version 7.3	March 6th, 2012	Release notes	
Version 7.2	November 30th, 2011	Release notes	
Version 7.1	August 4th, 2011	Release notes	
Version 7.0	January 18th, 2011	Release notes	
Version 6.0	August 23rd, 2010	Release notes	Documentation (PDF)
Version 5.2	January 20th, 2010	Release notes	Documentation (PDF)
Version 5.1	July 13th, 2009	Release notes	*See version 5.2 documentation*
Version 5.0	June 18th, 2009	Release notes	Documentation (PDF)

Before version 5.0, ANTS Memory Profiler was part of the ANTS Profiler product. No documentation is available for ANTS Profiler.

If you need to install an old version of ANTS Memory Profiler, go to Download old versions of products.

ANTS Memory Profiler 7.4 release notes

April 5th, 2012

Fixes

- Various bug fixes

Known issues

- Results files created in ANTS Memory Profiler 7.4 cannot be opened in earlier versions of the product.

ANTS Memory Profiler 7.3 release notes

March 6th, 2012

New features

- This version adds support for profiling websites in IIS Express and for SharePoint 2010.

ANTS Memory Profiler 7.2 release notes

November 30th, 2011

Fixes

- Re-enabling custom naming for snapshots taken via the snapshot API
- Preventing the instance list from refreshing on every click when accessed from the instance categorizer
- Resolving a rare race condition in the profiler core

ANTS Memory Profiler 7.1 release notes

August 4th, 2011

New features

- As of this version, profiling with Silverlight 5 beta is supported.
- Although full support for Sharepoint 2010 is not built into this version, a workaround may allow you to profile managed code running on a Sharepoint 2010 server with IIS 7. For details, see Profiling Sharepoint 2010.

Fixes

This version fixes a small number of bugs present in version 7.0. These include improvements to DEP (data execution prevention) handling, and clearer error messaging if a specified ASP.NET site cannot be found.

This release also fixes a bug in Red Gate's ANTS Performance Profiler, should you have this program installed. The fix, for a bug that silently disabled the "Check for updates on startup" setting, sets a registry key value; this value is set even if you do not have ANTS Performance Profiler installed. The key can safely be ignored.

This fix will not overwrite any "check for updates" preference that you have set manually, and the setting can also still be changed: on the **Help** m enu in ANTS Memory Profiler or ANTS Performance Profiler, click **Check for updates...** and select or clear the "Check for updates on startup" checkbox.

ANTS Memory Profiler 7.0 release notes

January 18th, 2011

New features

- Redesigned summary. This gives immediate feedback on where your memory is being used, and now warns you if problems with large object heap fragmentation are detected.
- New Instance Categorizer graph. Groups the instances of the class that you are investigating by the shortest path to the GC root, allowing you quickly to spot any paths that should not hold references to the instances.
- Redesigned and reorganized filters.
- New 'show only survivors in growing classes' filter highlights any classes where no instances were disposed between two snapshots, even though the number of instances of that class increased.
- Field properties now available on the Instance Retention Graph.
- Improved support for WPF: Dependency properties now shown with other field properties.
- Snapshot API lets you take snapshots from your code.

See Upgrading from ANTS Memory Profiler version 6 to version 7.

Known issues

Limitations of the .NET 1.1 framework restrict the memory information ANTS Memory Profiler can retrieve. Specifically, with .NET 1.1 applications:

- Object disposal cannot be tracked.
- LOH fragmentation detection is less reliable.
- The generation in which an object is held cannot be identified.
- Field values and static variable values cannot be displayed.
- Profiling runs more slowly.

The following feature requires the .NET 4.0 runtime and Windows Vista or later:

- Attach to process

Additionally:

- For some types of application, you may have to run ANTS Memory Profiler as an administrator.
- If you have Internet Explorer 7 on Windows Vista x64 or Windows Server 2008 x64, it is not possible to profile ASP.NET websites on IIS. Upgrading to Internet Explorer 8 will solve this problem.
- On Windows Vista and Windows 7, after profiling IIS using attach to process, ANTS Memory Profiler 7 sometimes cannot stop the session. If this happens, restart ANTS Memory Profiler before starting a new profiling session.
- In some cases, Windows Services do not restart properly after profiling. Use services.msc to restart services if required.
- To profile an XBAP application, Internet Explorer must be set as your default browser and it must be closed before profiling.
- To profile a .NET 4 process, you must first disable concurrent garbage collection.
- A limitation in Windows means that it is not possible to successfully attach to a .NET 4 process more than once.
- SharePoint 2010 is not supported.
- The snapshot API does not work with Silverlight applications, due to security restrictions in Silverlight.

ANTS Memory Profiler 6.0 release notes

August 23rd, 2010

Features, Enhancements and Bug Fixes

This release of ANTS Memory Profiler adds full support for memory profiling applications running under .NET 4 and Silverlight.

It also includes the vastly improved IIS support, along with XBAP support, that we first shipped with ANTS Profiler 4.3.

ANTS Performance Profiler 6, also available from www.red-gate.com, is our new performance profiler, which includes support for .NET 4 and Silverlight applications and also allows you to record and analyze SQL and File I/O activity.

Supported .NET Framework Versions

- 1.1
- 2.0
- 3.0
- 3.5
- 4.0
- Silverlight 4

The .NET 2.0 runtime must be installed in order for ANTS Memory Profiler to run.

Supported OS Versions

- Windows XP SP2 or later
- Windows Server 2003
- Windows Vista
- Windows Server 2008
- Windows 7
- Windows Server 2008 R2

Supports 32-bit and 64-bit versions of all listed OSs. Windows 2000 is no longer supported.

The following feature requires the .NET 4.0 runtime and Windows Vista or
later:

- Attach to process

Other System Requirements

512 MB RAM (minimum)
Internet Explorer 6+
1 GB free hard disk space
Supported Visual Studio Versions

The following versions of Microsoft Visual Studio are supported by the ANTS Memory Profiler 6 add-in:

- Visual Studio 2005
- Visual Studio 2008
- Visual Studio 2010

Known Issues

For some types of application, you may have to run ANTS Memory Profiler as an administrator.

On Windows Vista and Windows 7, after profiling IIS using attach to process, ANTS Memory Profiler 6 sometimes cannot stop the session. If this happens, restart ANTS Memory Profiler before starting a new profiling session.

In some cases, Windows Services do not restart properly after profiling. Use services.msc to restart services if required.

To profile an XBAP application, Internet Explorer must be set as your default browser and it must be closed before profiling.

To attach to a .NET 4 process, you must first disable concurrent garbage collection.

ANTS Memory Profiler 5.2 release notes

January 20th, 2010

This latest version of Red Gate's superfast, award winning memory profiler contains numerous important bug fixes, and is a recommended upgrade for all users of ANTS Memory Profiler 5.0 and 5.1:

- Fixed infinite progress hang when taking a snapshot while profiling Sharepoint.
- Now warns about object reference fix-up problem when running application with Server GC; this has been raised as a bug on Microsoft Connect.
- Fixed CouldNotMapFileException when profiling an application with more than 46,000,000 objects on a 32-bit system.
- Fixed problem where Dispose tracking only worked on debug, non-optimized builds.
- Fixed problem where cross-assembly references can cause a crash when multiple AppDomains are used.
- Fixed intermittent GDI+ errors caused by summary view graphs.
- Fixed server error from "What do I look for?" help link at bottom of class graph.
- Corrected support forum URL on Getting Started form.

ANTS Memory Profiler 5.1 release notes

July 13th, 2009

- Installer fixes

ANTS Memory Profiler 5.0 release notes

June 18th, 2009

ANTS Memory Profiler 5 is our new, completely rewritten, memory profiler. It offers the following features and benefits:

- Unparalleled performance
- Ability to take and analyse an arbitrarily large number of memory snapshots
- Support for snapshots up to 2GB (32-bit operating systems), and 4GB (64-bit operating systems)
- Arbitrary snapshot comparisons
- Summary information for profiling session, individual snapshots, and snapshot comparisons
- Large object heap fragmentation statistics
- Extremely powerful filtering options to narrow down to the objects you really need to know about
- A unique class graph view which allows you to quickly see where instances of a given class are being referenced
- A unique object retention graph, which quickly allows you to see shortest reference paths to all GC roots, which will need to be broken to fix memory leaks
- Ability to profile .NET executables, ASP.NET applications and web services in IIS and web development server, services, COM+ applications, and XBAP applications.

Before version 5.0, ANTS Memory Profiler was part of the ANTS Profiler product.