Beginners Guide to Performance Profiling

Visual Studio 2010

This topic applies to:

Visual Studio Ultimate	Visual Studio Premium	Visual Studio Professional	Visual Studio Express
✓	✓	✗	✗

This topic describes a basic method for using the Profiling Tools of Visual Studio Premium and Visual Studio Ultimate to analyze performance issues in your application. Although the Profiling Tools provide many options for gathering customized performance data in many kinds of applications, this topic is about how to use the **Profiling Wizard** to gather **Sampling** data for a Visual Studio solution.

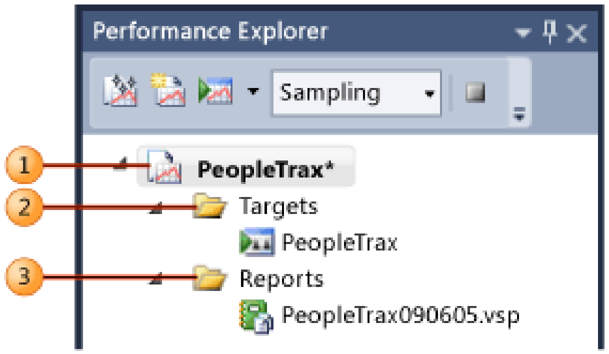
Note

If **Sampling** does not give you the data that you need, other Profiling Tools collection methods provide different kinds of information that might be helpful to you. For more information about these other methods, see How to: Choose Collection Methods.

In this topic

Basic concepts

Performance Session When you use the **Performance Profiler**, you create a *Performance Session*, which contains the configuration data for collecting performance information and the results of one or more profiling runs. After you create a performance session it appears in the Performance Explorer window.



1. The name of the profiling session.

2. The **Targets** folder shows the projects or binaries that are profiled in the session.

3. The **Reports** folder contains the profiling data files from one or more collection runs. You can click a file name and select views of the performance information such as functions calls, memory allocations, and details of specific functions. Each The views are displayed in the main Visual Studio window.

Sampling Method **Sampling** is a statistical profiling method that shows you the functions that are doing most of the user mode work in the application. Sampling is a good place to start to look for areas to speed up your application.

At specified intervals, the **Sampling** method collects information about the functions that are executing in your application. After you finish a profiling run, the **Summary** view of the profiling data appears in the main Visual Studio window. The **Summary** view shows the most active function call tree, called the **Hot Path**, where most of the work in the application was performed, The view also lists which functions were performing the most individual work, and provides a timeline graph you can use to focus on specific segments of the sampling session.

Prerequisites

These are a few things that you can do before you start profiling to make sure that you do not encounter unnecessary problems.

Run as administrator If you are not an administrator on the computer that you are using, you should run Visual Studio as an administrator to make sure that you have the permissions that are necessary for some of the features in the profiling tools. To do this, click the **Start** button, locate the Visual Studio application icon, right-click the icon, and then click **Run as administrator**.

Set the active build configuration to Release Debug builds insert additional diagnostic code into your application and do not include optimizations that the compiler performs in release builds. Profiling the release version of your application provides more accurate data about the performance of your application. To change the active configuration, on the **Build** menu click **Configuration Manager** and in the dialog box, under **Active solution configurations**, select **Release**.

Get Windows symbols files If you profile code that calls Windows functions, you should make sure that you have the most current .pdb files. Without these files your report views will list Windows function names that are cryptic and difficult to understand. For more information about how to make sure that you have the files you need, see How to: Reference Windows Symbol Information.

Step 1: Create and run a performance session

To get the data that you need to analyze, you must first create a performance session and then run the session. The **Performance Wizard** lets you do both.

To create and run a performance session

1. Open the solution in Visual Studio.
2. On the **Analyze** menu, click **Launch Performance Wizard**.
3. Accept the default setting of **CPU Sampling (recommended)** and click **Next**.
4. Accept the default project, and then click **Next**.
5. Make sure that the **Launch profiling after the wizard finishes** check box is selected and then click **Finish**.

   Your application starts and the profiler starts to collect data.
6. Exercise the functionality that might contain performance issues.
7. Close the application as you usually would.

   After you finish running the application, the **Summary** view of the profiling data appears in the main Visual Studio window and an icon for the new session appears in the **Performance Explorer** window.

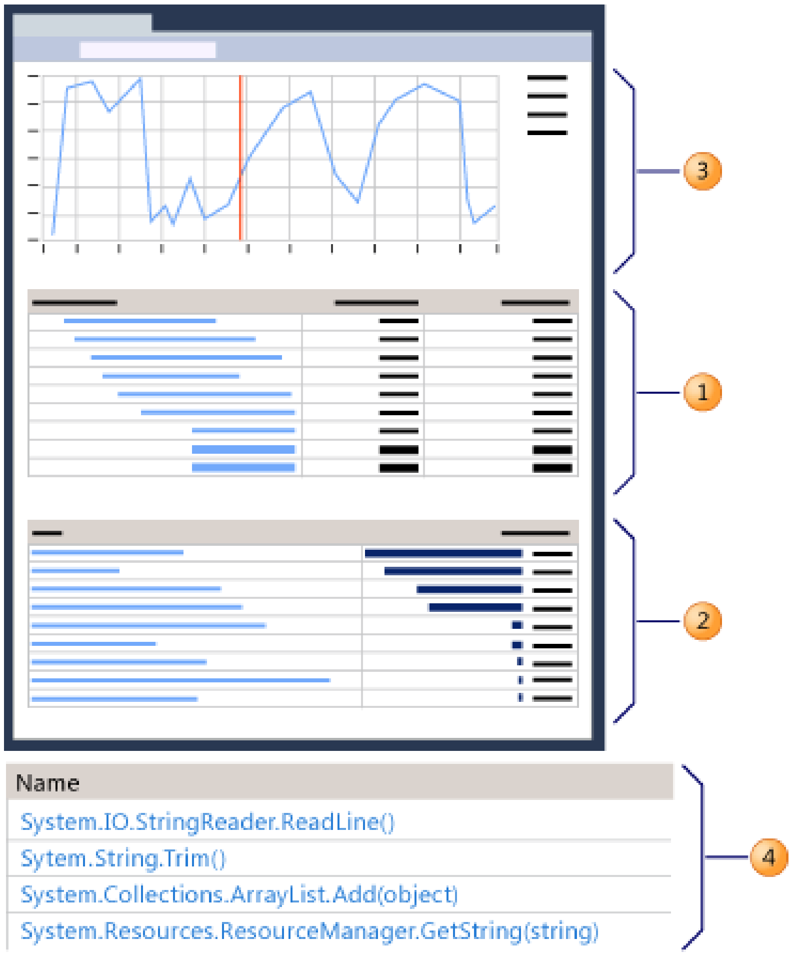
Step 2: Analyze Sampling data

When you finish running a performance session, the **Summary** view of the profiling report appears in the main window in Visual Studio.

We recommend that you begin analyzing your data by examining the **Hot Path,** then the list of functions that are doing the most work, and finally by focusing on other functions by using the **Summary Timeline**. You can also view profiling suggestions and warnings in the **Error List** window.

Be aware that the sampling method might not give you the information that you need. For example, samples are collected only when the application is executing user mode code. Therefore, some functionality, such as input and output operations, is not captured by sampling. The Profiling Tools provide several collection methods that can enable you to focus on the important data. For more information about the other methods, see How to: Choose Collection Methods.

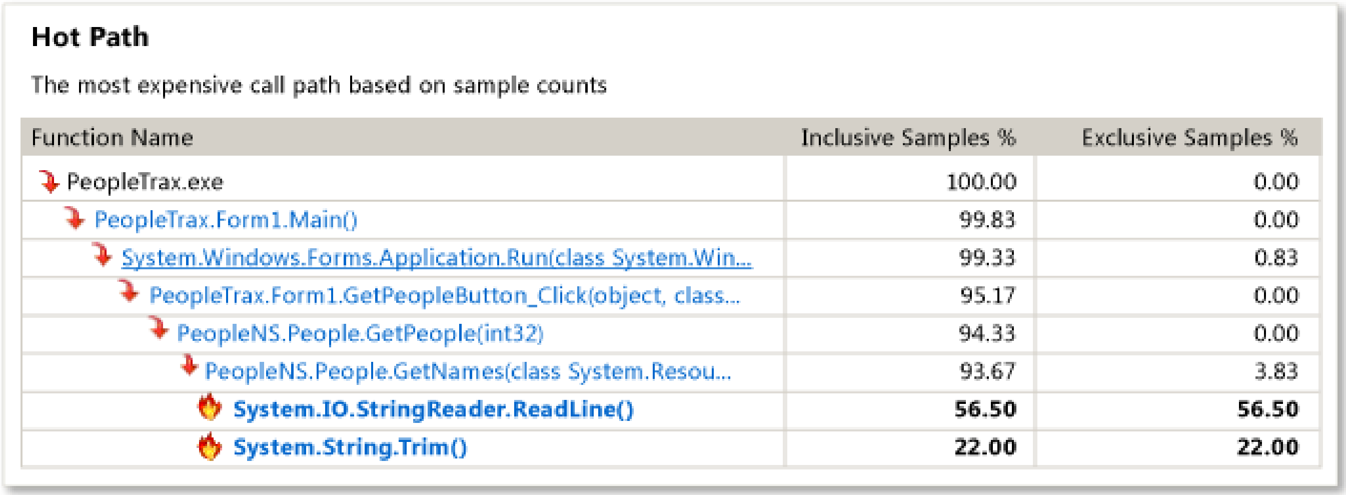
Each numbered area in the figure relates to a step in the procedure.



Name
System.IO.StringReader.ReadLine()
Sytem.String.Trim()
System.Collections.ArrayList.Add(object)
System.Resources.ResourceManager.GetString(string)

To analyze sampling data

1. In the **Summary** view, the **Hot Path** shows the branch of your application's call tree with the highest inclusive samples. This is the execution path that was most active when data was collected. High inclusive values can indicate that the algorithm that generates the call tree can be optimized. Find the function in your code that is lowest in the path. Notice that the path can also include system functions or functions in external modules.



Hot Path

The most expensive call path based on sample counts

Function Name	Inclusive Samples %	Exclusive Samples %
PeopleTrax.exe	100.00	0.00
PeopleTrax.Form1.Main()	99.83	0.00
System.Windows.Forms.Application.Run(class System.Win...	99.33	0.83
PeopleTrax.Form1.GetPeopleButton_Click(object, class...	95.17	0.00
PeopleNS.People.GetPeople(int32)	94.33	0.00
PeopleNS.People.GetNames(class System.Resou...	93.67	3.83
System.IO.StringReader.ReadLine()	**56.50**	**56.50**
System.String.Trim()	**22.00**	**22.00**

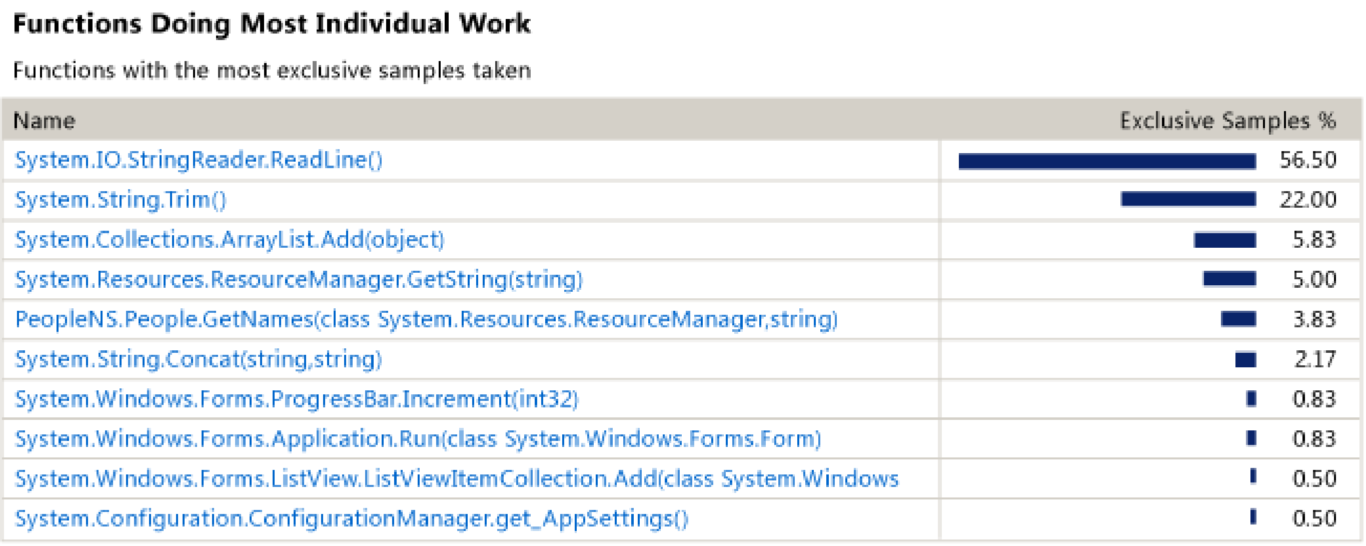
a. **Inclusive Samples** indicate how much work was done by the function and any functions called by it. High inclusive counts point to the functions that are most expensive overall.

b. **Exclusive Samples** indicate how much work was done by the code in the function body, excluding the work done by functions that were called by it. High exclusive counts may indicate a performance bottleneck within the function itself.

2. Click the function name to display the **Function Details** view of the profiling data. The **Function Details** view presents a graphical view of the profiling data for the selected function, showing all the functions that called that function and all the functions that were called by the selected function.

    - The size of the blocks of the calling and called functions represent the relative frequency that the functions called or were called.
    - You can click the name of a calling or called function to make it the selected function of the Function Details view.
    - The lower pane of the **Function Details** windows displays the function code itself. If you examine the code and find an opportunity to optimize its performance, click the source file name to open the file in the Visual Studio editor.

3. To continue your analysis, return to the **Summary** view by selecting **Summary** from the View drop-down list. Then examine the functions in **Functions Doing the Most Individual Work**. This list displays the functions with the highest exclusive samples. The code in the function body of these functions performed significant work and you might be able to optimize it. To further analyze a particular function, , click the function name to display it in the **Function Details** view.



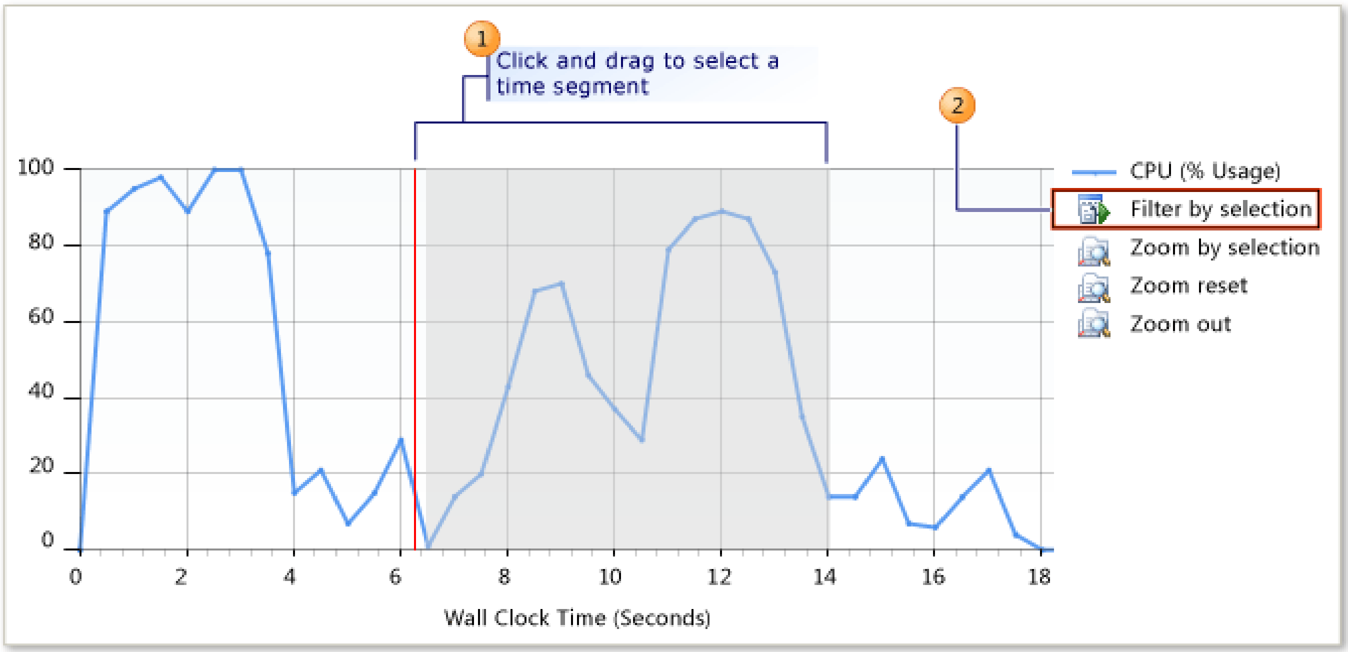
Functions Doing Most Individual Work

Functions with the most exclusive samples taken

Name	Exclusive Samples %
System.IO.StringReader.ReadLine()	56.50
System.String.Trim()	22.00
System.Collections.ArrayList.Add(object)	5.83
System.Resources.ResourceManager.GetString(string)	5.00
PeopleNS.People.GetNames(class System.Resources.ResourceManager,string)	3.83
System.String.Concat(string,string)	2.17
System.Windows.Forms.ProgressBar.Increment(int32)	0.83
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	0.83
System.Windows.Forms.ListView.ListViewItemCollection.Add(class System.Windows	0.50
System.Configuration.ConfigurationManager.get_AppSettings()	0.50

To continue your investigation of the profiling run, you can reanalyze a segment of the profiling data by using the timeline in the **Summary** view to show you the **Hot Path** and **Functions Doing Most Individual Work** from a selected segment. For example, focusing on a smaller peak in the timeline might reveal expensive call trees and functions that were not shown in the analysis of the entire profiling run.

To reanalyze a segment, select a segment inside the Summary Timeline box and then click **Filter by Selection**.



4. The profiler also uses a set of rules to suggest ways of improving the profiling run and to identify possible performance problems. If an issue is found, a warning is displayed in the **Error List** window. To open the **Error List** window, on the **View** menu click **Error List**.

    - To see the function that raised a warning the **Function Details** view, double-click the warning.
    - To view detailed information about the warning, right-click the error and then click **Show Error Help**

Step 3: Revise code and rerun a session

After you find and optimize one or more functions, you can repeat the profiling run and compare the data to see the difference that your changes have made to the performance of your application.

To revise code and rerun the profiler

1. Change your code.
2. To open the **Performance Explorer**, on the **View** menu click **Other Windows** and then click **Performance Explorer**.
3. In the **Performance Explorer**, right-click the session that you want to rerun, and then click **Launch with Profiling.**
4. After you rerun the session, another data file is added to the **Reports** folder for the session in **Performance Explorer**. Select both the original and new profiling data, right-click the selection, and then click **Compare Performance Reports**.

   A new report window opens, displaying the results of the comparison. For more information about how to use the comparison view, see How to: Compare Profiler Data Files.

See Also

Concepts

Analyzing Application Performance by Using Profiling Tools

Getting Started with Profiling Tools

Other Resources

Overviews (Profiling Tools)

Community Additions

Performance Wizard. No launchable project available to profile in the current solution

I am tying to run the Performance Wizard on my XNA WP7. I Select "CPU Sampling" Next->get to page 2 of 4. it asks "Which application would you like to target for profilling"But there are not items in the dropdown list. all I see is "No launchable project available to profile in the current soulution"I saw something about needing SP1 for this to work and I have made sure I have it. Any help please.

thewildword

6/5/2011