DevPartner 11.0.0

User's Guide

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa , CA 92626

2012-09-05

Table of Contents

Preface

This guide describes how to get started using your Micro Focus® DevPartner® Studio software.

Who Should Read This Guide

This guide is intended for new DevPartner Studio users. Chapter 1 presents an overview of DevPartner Studio concepts; subsequent chapters describe individual DevPartner components. Each component chapter begins with a brief Ready, Set, Go procedure to get new users up and running with DevPartner Studio.

Users of previous versions of DevPartner should read the Preface to the *DevPartner Installation Guide* to see how this version of DevPartner differs from previous versions.

This guide contains information relevant to all DevPartner Studio products, including the Professional and Enterprise Editions, and the DevPartner for Visual C++ BoundsChecker Suite.

Note: The DevPartner for Visual C++ BoundsChecker Suite analyzes unmanaged code only. The DevPartner memory analysis, static code analysis, and Performance Expert features analyze managed code only, and are therefore not supported in the DevPartner for Visual C++ Bounds-Checker Suite.

This guide assumes that you are familiar with the Windows interface, with Microsoft Visual Studio, and with software development concepts.

What This Guide Covers

This guide contains the following chapters and appendixes:

- **Chapter 1,** ***Introducing DevPartner*** describes the concepts and components of DevPartner.
- **Chapter 2,** ***Error Detection,*** explains how to use DevPartner to uncover errors in your C and managed and unmanaged C++ code.
- **Chapter 3,** ***Static Code Analysis,*** explains how DevPartner helps you locate a variety of errors in Visual Basic and Visual C# code.
- **Chapter 4**, ***Automatic Code Coverage Analysis***, describes how to use DevPartner to track how much of your code is covered by your tests.
- **Chapter 5,** ***Finding Memory Problems***, describes how to use DevPartner to diagnose application anomalies that can be caused by misuse of memory and objects.

- **Chapter 6**, ***Automatic Performance Analysis***, explains how DevPartner helps you locate bottlenecks and code in need of optimization.
- **Chapter 7**, ***In-Depth Performance Analysis***, explains how DevPartner helps you analyze a variety of full system performance issues.
- **Chapter 8, *System Comparison,*** describes how DevPartner helps you identify differences between computer systems to assist with troubleshooting application development problems.
- **Appendix A, *DevPartner Studio Supported Project Types***, contains tables listing project types supported by each DevPartner Studio feature.
- **Appendix B, *Starting Analysis from the Command Line***, describes the **`DPAnalysis.exe`** command line interface.
- **Appendix C, *Analysis Session Controls***, describes creating a session control file for coverage, memory, performance, and Performance Expert sessions.
- **Appendix D, *Exporting Analysis Data to XML***, describes exporting coverage, performance, and Performance Expert data to an XML file.

Conventions Used In This Manual

This book uses the following conventions to present information.

- Screen commands and menu names appear in **bold typeface**. For example:

  Choose **Item Browser** from the **Tools** menu.

- File names appear in **`monospace typeface`**. For example:

  The *DevPartner Studio User's Guide* (**`Understanding DevPartner.pdf`**) describes...

- Variables within computer commands and file names (for which you must supply values appropriate for your installation) appear in ***`italic monospace type.`*** For example:

  Enter **`http://`*`servername`*`/cgi-win/itemview.dll`** in the `Destination` field...

For More Information

Refer to the feature-level online help for step-by-step instructions for specific DevPartner Studio tasks.

You can use the feature-level online help to learn more about DevPartner Studio functions and procedures.

You can use the following resources to learn more about DevPartner components. Manuals in Adobe Acrobat (.pdf) format and the DevPartner Studio Release Notes are available through the Micro Focus SupportLine web site product page for DevPartner Studio at http://supportline.microfocus.com/, the InfoCenter option in the **Start > Programs > Micro Focus >** DevPartner Studio menu, and are also included on your DevPartner Studio DVD.

- The *Distributed License Management Licensing Guide* describes DevPartner Studio licensing.
- The *DevPartner Studio User's Guide* contains details about using DevPartner.
- The DevPartner Studio *Quick Reference* provides an at-a-glance summary of DevPartner features accompanied by quick-start advice.
- The DevPartner *Advanced Error Detection Techniques* manual provides concepts and procedures to help you understand the use of Micro Focus DevPartner Error Detection software.
- The DevPartner Studio *Release Notes* contains a list of known issues and technical notes for DevPartner Studio. The Release Notes document is available from the DevPartner Studio installation setup and the DevPartner Studio InfoCenter.

Chapter 1

Introducing DevPartner

This chapter provides an introduction to DevPartner Studio Professional Edition and DevPartner for Visual C++ BoundsChecker Suite. Use this manual to understand the concepts underlying both DevPartner products.

Note: The DevPartner for Visual C++ BoundsChecker Suite analyzes unmanaged code only. The DevPartner memory analysis, static code analysis, and Performance Expert features analyze managed code only, and are therefore not supported in the DevPartner for Visual C++ BoundsChecker Suite.

What is DevPartner Studio?

DevPartner Studio provides a variety of programmer productivity features, such as automated error detection, source code analysis, coverage analysis, memory analysis, performance profiling, system performance analysis, and system comparison.

DevPartner can analyze a broad range of both managed and unmanaged applications written in a variety of languages. Refer to Appendix A, “DevPartner Studio Supported Project Types” for a complete list of supported project types and languages.

The following sections summarize the features of DevPartner Studio.

Error Detection

DevPartner Studio provides automated error detection for 32-bit and 64-bit managed and unmanaged applications. DevPartner error detection is built on BoundsChecker™ technology, and is designed to locate the following hard-to-find errors in your Windows-based applications:

- Memory, resource, and COM interface leaks
- Invalid use of Windows API calls
- Invalid use of memory or pointers
- Memory overrun errors
- Un-initialized memory usage
- Use of dangling pointers
- Errors in .NET finalizers

DevPartner error detection monitors your application from the moment of creation until the final moments before the process is unloaded from memory. You can monitor all DLL loads and unloads, static constructors and destructors as well as the normal flow of your application. You can also tune DevPartner error detection to collect only information necessary to solve a particular problem by filtering out specific files or portions of your application.

See “Error Detection” on page 23 for more information about DevPartner error detection.

Static Code Analysis

DevPartner helps developers write compliant Visual Basic and C# code within Visual Studio. DevPartner identifies programming and naming violations in the .NET Framework, analyzes method call structures, and tracks overall code complexity.

The DevPartner software detects a variety of coding errors:

- Variable naming inconsistencies
- Violations of coding covenants
- Win32 API validation errors
- Common logic errors
- .NET portability issues
- Structured exception handling errors

Using an extensive and extensible rule set, DevPartner also assists in the porting of legacy Visual Basic code by identifying constructs that do not work in the .NET environment.

See “Static Code Analysis” on page 59 for more information about DevPartner static code reviews.

Coverage Analysis

The DevPartner coverage analysis feature allows developers and test engineers to be sure they are testing all of an application’s code. When you run your tests with coverage analysis, DevPartner tracks all 32-bit and 64-bit applications, components, images, methods, functions, modules, and individual lines of code covered by your tests. When your tests end, DevPartner displays information about what code was exercised and what code was not exercised.

DevPartner collects coverage data for managed code applications, including Web and ASP.NET applications, as well as unmanaged C++ applications. (See Appendix A, “DevPartner Studio Supported Project Types” for a complete list of supported technologies).

You can run the coverage analysis and error detection features simultaneously. Knowing the percentage of code covered in your tests increases confidence in your error detection results.

See “Automatic Code Coverage Analysis” on page 109 for more information about code coverage analysis.

Memory Analysis

Memory analysis assesses how .NET applications allocate memory. When you run a 32-bit or 64-bit application under memory analysis, DevPartner shows you the amount of memory consumed by an object or class, tracks the references that are holding an object in memory, and identifies the lines of source code within a method that are responsible for allocating the memory.

More importantly, DevPartner presents memory data in context. This enables you to navigate chains of object references and calling sequences of the methods in your code. Presenting memory data in context provides both an in-depth understanding of how your program uses memory and the critical information you need to optimize memory use.

See “Finding Memory Problems” on page 131 for more information about memory analysis.

Performance Analysis

The DevPartner performance analysis feature analyzes your code for performance bottlenecks. It pinpoints these bottlenecks to individual lines of source code, and provides method-level insight into the way 32-bit and 64-bit applications use third-party components, the operating system, and, most importantly, the .NET Framework.

DevPartner supports performance profiling in all supporte4d releases of Microsoft Visual Studio. (See Appendix A, "DevPartner Studio Supported Project Types" for a complete list of supported technologies).

To improve performance of critical parts of your code, use DevPartner performance analysis to locate performance bottlenecks and to verify improvements you make impact performance.

See "Automatic Performance Analysis" on page 179 for more information about analyzing an application's performance.

In-Depth Performance Analysis

The DevPartner Performance Expert feature takes performance profiling a few steps further than DevPartner's performance analysis feature. For 32-bit and 64-bit managed applications, Performance Expert provides a deeper analysis of the following hard-to-solve problems:

- CPU/thread usage
- File/disk I/O
- Network I/O
- Synchronization wait time

Performance Expert analyzes application at run time and locates the problematic methods in your code. It then lets you view details about individual lines in the method, or examine parent-child calling relationships to help determine the best way to fix the problem. When you decide on an approach, Performance Expert lets you jump directly to the problem lines in source code, so you can quickly fix problems.

See "In-Depth Performance Analysis" on page 207 for more information.

System Comparison

The DevPartner System Comparison utility compares two computer systems, or compares the current state of a computer with a previous state, allowing you to determine why your application:

- Works on one computer but not on another
- Works differently on different computers
- No longer works on a computer on which it previously worked

To compare systems, System Comparison creates XML files, called snapshot files, that contain information about a computer system, such as its installed products, system files, drivers, and many other system characteristics. It then compares snapshot files and reports the differences between them.

Unlike the other DevPartner features, System Comparison is not integrated into the Visual Studio environment. It runs as a standalone utility to minimize its impact on target systems.

System Comparison consists of a service that takes nightly snapshots of a system, and the user interface that enables you to take snapshots manually and to compare snapshots to find differences. System Comparison also includes a command line interface and a Software Development Kit (SDK). The SDK allows software developers to gather additional information for comparison and to embed snapshot functionality in deployed applications.

See “System Comparison” on page 243 for more information about the System Comparison utility.

DevPartner and Visual Studio

DevPartner integrates seamlessly into supported Visual Studio environments. This integration makes it easy for you to use the capabilities of the product as you write and debug applications. You can perform code analysis frequently as you develop an application without leaving the development environment.

DevPartner simultaneously supports application development within the supported Visual Studio environments. This support assists developers as they migrate code from the older Microsoft environments to the latest .NET Frameworks.

The following tables identify the DevPartner features available in various Visual Studio environments.

Table 1-1. Installed features for DevPartner Studio Professional Edition

Microsoft Visual Studio 2012	Microsoft Visual Studio 2010	Microsoft Visual Studio 2008 and 2005
Performance Analysis	Performance Analysis	Performance Analysis
Coverage Analysis	Coverage Analysis	Coverage Analysis
Error Detection	Error Detection	Error Detection
Static Code Analysis	Static Code Analysis	Static Code Analysis
Memory Analysis	Memory Analysis	Memory Analysis
Performance Expert	Performance Expert	Performance Expert

Table 1-2. Installed features for DevPartner for Visual C++ BoundsChecker Suite

Microsoft Visual Studio 2012	Microsoft Visual Studio 2010	Microsoft Visual Studio 2008 and 2005
Performance Analysis	Performance Analysis	Performance Analysis
Coverage Analysis	Coverage Analysis	Coverage Analysis
Coverage Analysis	Coverage Analysis	Error Detection

Menus and Toolbars in Visual Studio

DevPartner adds a menu and several toolbars to Visual Studio, and it adds menu commands to several Visual Studio menus, including context (right-click) menus. Menu commands and toolbars provide access to session controls, the rules for static code reviews, options, and instrumentation controls.

DevPartner adds a toolbar to Visual Studio to provide quick access to DevPartner features. The following graphic shows the DevPartner toolbar in Visual Studio 2010.

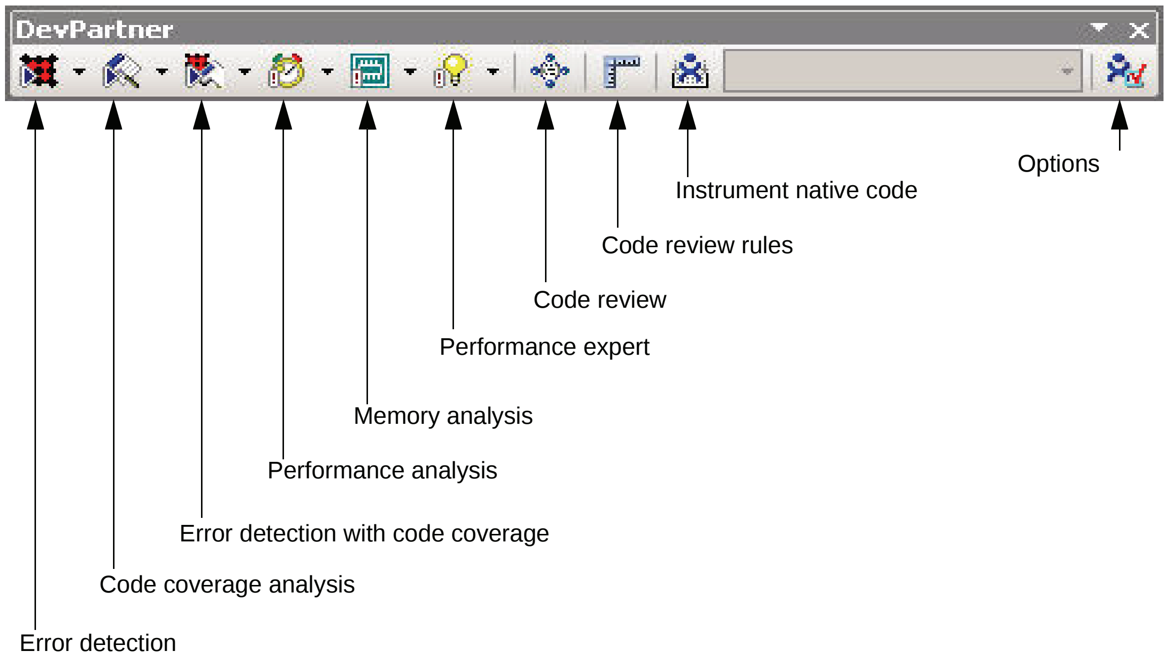


Figure 1-1. DevPartner toolbar

DevPartner also places a session control toolbar in the IDE. When the coverage analysis, memory analysis, performance analysis, and Performance Expert features are active, the session control toolbar is active.

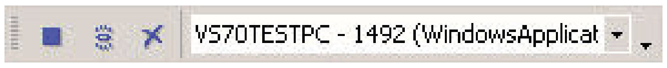


Figure 1-2. The Session Control toolbar

The Session Control toolbar consists of three icons and a process list.

- ■ Stops data collection and takes a final data snapshot
- Takes a data Snapshot
- ✕ Clears data collected to the point at which the Clear action executes

The process list focuses data collection on a single process for applications that use multiple processes.

In addition to menus and toolbars, DevPartner uses the Visual Studio dockable windows and panes to display the results of analysis sessions. It also uses the Solution Explorer to display the names of session files. DevPartner also adds pages to Visual Studio Options, Solution Properties, and Project Properties for configuring DevPartner code analysis operations.

Using DevPartner in Visual Studio

The general work flow for using DevPartner within Visual Studio consists of one or more of these general-purpose tasks:

- Open or create a Solution in Visual Studio
- Set options for code analysis operations
- Enable the analysis you want to perform from the DevPartner menu or toolbar
- Run your application
- View the session results returned by DevPartner

DevPartner gives you wide flexibility in choosing the parts of your application to monitor, selecting what data to view, and creating filters to eliminate unwanted information.

DevPartner also gives you the option to perform many functions from the command line. This capability provides a way to use DevPartner functionality in automated batch processing operations, such as nightly-build smoke tests.

Integrated Online Help

DevPartner Studio provides extensive online help about each of its features. This help should be the first place you turn for how-to and reference information.

Provided in the same format as the rest of Visual Studio help, the DevPartner online help appears in the Visual Studio help. The DevPartner Studio help contains a volume for each DevPartner component.

Visual Studio Team System Support

Visual Studio Team System is Microsoft's version control, defect tracking, and process management software for Visual Studio software development projects. DevPartner Studio supports Microsoft Visual Studio Team System if the Team System client software is installed and a Team Foundation Server connection is available.

DevPartner Studio supports submission of a Work Item of the type **Bug** to Visual Studio Team System. When you submit a bug, DevPartner automatically populates the Work Item form with selected session data. To submit a bug from DevPartner, the active Team System project must support a Work Item of the type **Bug**. DevPartner automatically adds data only to this type of Work Item.

You can submit a Work Item that includes DevPartner data from any of the following views in a DevPartner session file:

- A method list or method table in a Coverage, Memory, or Performance Analysis session file, or in a Performance Expert session file

- The code review Problems or Naming tabs

- A list of errors or leaks in any Error Detection tab, or list of instances in the Error Detection Modules or .NET Performance tabs

To submit a Team System Work Item from DevPartner, right-click on a method or other item in a session file and choose **Submit Work Item**. DevPartner populates the Title and Description or Symptom field. Fill in any other required data and save the Work Item.

Note: If you use the Team Explorer context menus in Visual Studio, DevPartner does not automatically populate the Work Item with session data.

For more information about submitting data from DevPartner Studio to Team System, see the Visual Studio Team System sections in the chapters of this guide. Consult the Microsoft Visual Studio Team System documentation for complete information on how to use Team System to support your development and project management activities.

Using Terminal Services and Remote Desktop

DevPartner Studio supports Windows Terminal Services. You can use Terminal Services to do anything you would be able to do using the computer directly, such as:

- Configure DevPartner options on remote systems.
- Enable or disable analysis on remote systems.
- Profile an application that runs on a remote system.

Licensing

A Terminal Services connection requires one DevPartner concurrent license per user display. A server connected through a Terminal Services connection does not require the DevPartner Studio Remote Server license.

Running Multiple Sessions Under Terminal Services

Multiple DevPartner sessions can run simultaneously on the Terminal Server. Multiple sessions can be started by a single user or multiple users. If a single user launches two instances of the console, both instances share the same workspace settings because DevPartner stores workspace settings on a per-user basis. If different users launch instances of coverage analysis, workspace settings can be configured separately for each instance.

Collected data includes activity in the server process from all users on the Terminal Server. To better focus data collection during a session, eliminate or limit extraneous application activity that uses the monitored processes or invokes the monitored targets.

Chapter 2

Error Detection

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with error detection. The second section provides reference information for an in-depth understanding of DevPartner error detection features.

Refer to the DevPartner online help for additional task-oriented information about error detection. For information that goes beyond the basics, refer to *Advanced Error Detection Techniques,* provided in PDF format with the DevPartner software installation.

What is Error Detection?

DevPartner is a comprehensive debugging solution for C and C++ development. Incorporating frequent checks with DevPartner error detection into your application development cycle allows you to produce stable, error-free code. DevPartner automates error-detection and analysis without adding time to the development process. The following features help you identify elusive bugs in 32-bit and 64-bit applications that are beyond the reach of traditional debugging and testing techniques:

- Comprehensive error detection
- Flexible debugging environment
- Integration with the Visual Studio debugger
- Integration with Microsoft Visual Studio
- Advanced error analysis
- Open error-detection architecture

Using Error Detection Out of the Box

The following Ready, Set, Go procedure introduces you to using DevPartner error detection.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject being described in the shaded box, read the additional text following the box.

Analyzing an application with DevPartner Studio does not require elevated system privileges. The system privileges used to create and debug the application are sufficient for DevPartner to analyze the application.

Ready: Deciding the Scope of Error Detection Analysis

Consider how you want to run DevPartner error detection on your code, as well as the types of errors and memory leaks you need to locate.

Note: DevPartner error detection creates data files for each target application. You must ensure that you have write access to the folder containing the target executable before starting error detection.

The following procedure assumes:

- ◆ Your solution contains unmanaged source code.
- ◆ You are running error detection in a supported Visual Studio release.

Refer to "Error Detection Supported Project Types" on page 268 for a comprehensive list of supported project types for DevPartner error detection.

Deciding How to Run the Session

You can run DevPartner error detection in several ways, depending on the needs of your situation:

- ◆ Run error detection interactively as part of your routine code validation process (daily or weekly) from inside Microsoft Visual Studio, or using the standalone application.
  - ◇ All error detection features can be accessed in the Visual Studio environment. You can configure DevPartner settings, check your program, and review detected errors.
  - ◇ You can run DevPartner as a standalone application outside of Visual Studio, but access to the Visual Studio editor to edit your code is unavailable.
- ◆ Automate error detection to run from a batch file or the command line using **`bc.exe`**.
  - ◇ When starting DevPartner from the command line, you can set up automated testing scripts. See "Running Error Detection from the Command Line" on page 55 for more information.
- ◆ Instrument your code with FinalCheck for a thorough validation at major development milestones (in Visual Studio only).

Deciding the Types of Errors to Locate

Use error detection to locate a wide variety of errors and leaks that may arise in your code and track down any errors.

- ◆ Enable COM object tracking to ensure there are no COM object errors in code.
- ◆ Enable Deadlock Analysis to ensure that synchronization objects are used properly, or for an application that deadlocks occasionally and the cause is no known.
- ◆ Extend the memory tracking system to include any custom allocators to ensure they are implemented without leaks or errors. To describe custom allocators, add descriptive information about your allocators to the `UserAllocators.dat` file. Refer to "Working with User-Written Allocators" in *Advanced Error Detection Techniques*.

Set: Configuring Options and Settings

You can customize DevPartner error detection to report specific types of errors, while ignoring or filtering out any “noise” that you do not care about.

For this procedure, use the default DevPartner properties and options. No changes to the settings are required.

Depending on how you are running DevPartner error detection, there are several menu options to access the Settings dialog box (see “Using the Settings Dialog Box” on page 40).

By default, error detection finds simple memory leaks, some memory errors, and resource leaks. Error detection can also find every instance the following types of errors, leaks, and events if you edit the default configuration.

- API calls and validation errors
- Potential deadlock situations
- COM interface leaks
- Memory allocations and deallocations
- Windows messages and other significant events (see “Tracking Windows Messages and Event Logging” on page 53)

Additionally, you can configure DevPartner to use FinalCheck. With FinalCheck, error detection instruments a C or C++ application, allowing it to pinpoint errors to the exact statement where they occur. FinalCheck takes longer to run and uses more resources, but locates and pinpoints difficult-to-find memory, pointer, and leak errors.

In addition to configuring error detection for specific errors and leaks, the error detection settings lets you:

- Define error detection parameters
- Change the fonts and colors used in the display
- Save parameters as a configuration file to use again
- Load different configuration files into your current session

Go: Running Your Solution with Error Detection

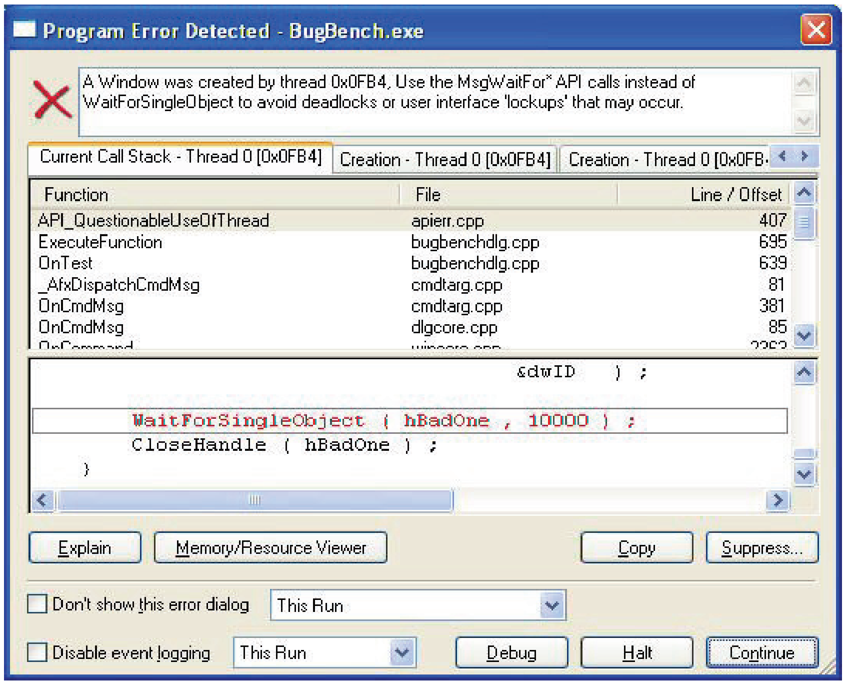
You are now ready to run your solution under DevPartner error detection.

1 Open your solution in Visual Studio.

2 Select **DevPartner** > **Start with Error Detection**.

3 Exercise the parts of your program that you want to check for errors.
Error detection displays the **Program Error Detected** dialog box each time it encounters a severe error (see Figure 2-1). Other errors are not severe enough, or are common enough, and are recorded and displayed in the **Results** pane (located in the upper left of the error detection main window) so you can address them later.

The Program Error Detected dialog box (see "Using the Program Error Detected Dialog Box" on page 32) displays a description of the error, followed by call stack information, and finally the code segment where the error was detected (when available). For further explanation about the error that was detected, click the **Explain** button.


Program Error Detected - BugBench.exe
Current Call Stack - Thread 0 [0x0FB4]
Creation - Thread 0 [0x0FB4]
Creation - Thread 0 [0x0FB
Explain
Memory/Resource Viewer
Copy
Suppress...
Debug
Halt
Continue

A Window was created by thread 0x0FB4, Use the MsgWaitFor* API calls instead of WaitForSingleObject to avoid deadlocks or user interface 'lockups' that may occur.

Function	File	Line / Offset
API_QuestionableUseOfThread	apierr.cpp	407
ExecuteFunction	bugbenchdlg.cpp	695
OnTest	bugbenchdlg.cpp	639
_AfxDispatchCmdMsg	cmdtarg.cpp	81
OnCmdMsg	cmdtarg.cpp	381
OnCmdMsg	dlgcore.cpp	85

```
                                      &dwID    ) ;

        WaitForSingleObject ( hBadOne , 10000 ) ;
        CloseHandle ( hBadOne ) ;
    }
```

Don't show this error dialog: This Run
Disable event logging: This Run

Figure 2-1. The Program Error Detected dialog box

4 If the Program Error Detected dialog box appears, respond in one of the following ways, and then continue exercising your program. Skip this step if the Program Error Detected dialog box does not appear.

- ◇ **Explain** - Provides a more detailed explanation of the error what can be done to resolve it.
- ◇ **Suppress** - Opens the Suppression dialog box and pre-populate it with this error. Suppressing an error prevents future occurrences from being acted upon by error detection. See "Understanding the Suppression and Filtering Dialog Boxes" on page 35.
- ◇ **Debug** - Opens code in the Visual Studio debugger at the line that generated the error.
- ◇ **Halt** - Stops the program and displays the Results pane.
- ◇ **Continue** - Acknowledges the error and continues. The error displays in the Results pane for further review after the session completes.

5 Stop your program when you are done checking it.

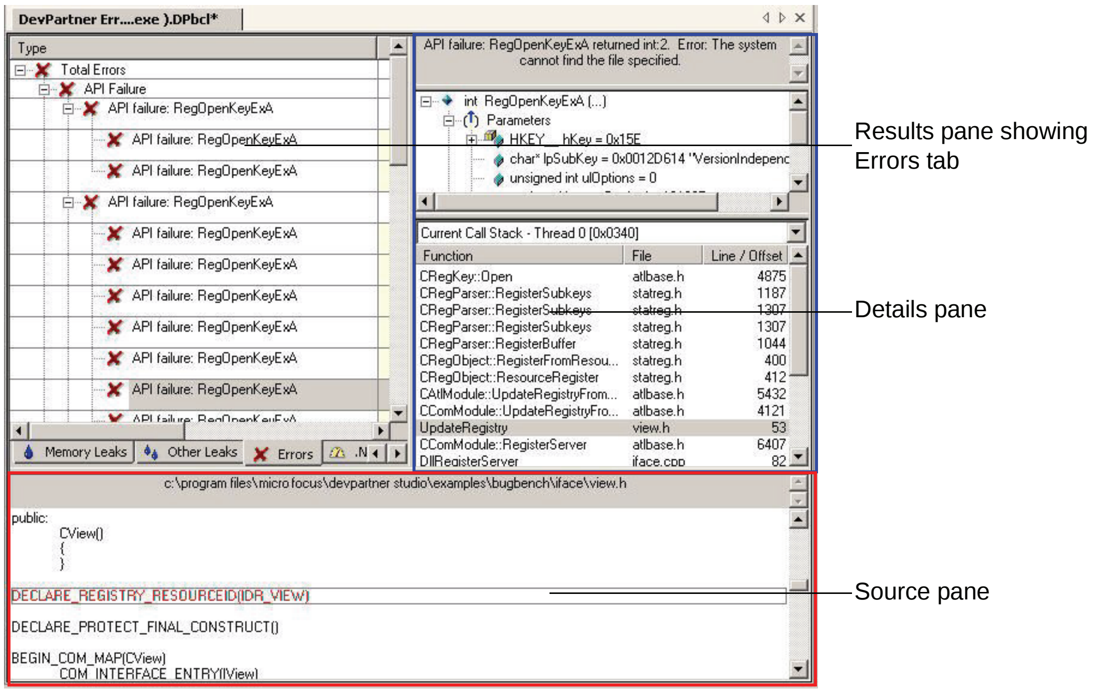
The program may not have a natural end, so close it when you have collected enough data. Use one of the following methods to close your program:

- ◇ Click **Halt** on the Program Error Detected dialog box.
- ◇ Select **Stop Debugging** from the **Debug** menu.
- ◇ Stop your application.

You have completed running a basic error detection session. Look in the **Results** pane to analyze the data from the errors and leaks detected.

Analyzing the Data in the Results Pane

The Results pane, located in the upper left of the main error detection window (see Figure 2-2), uses a series of tabs for navigating through the various types of information.


DevPartner Err....exe).DPbcl*
Results pane showing Errors tab
Details pane
Source pane
Memory Leaks
Other Leaks
Errors
.N

```
Type
Total Errors
  API Failure
    API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
    API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
      API failure: RegOpenKeyExA
```

```
API failure: RegOpenKeyExA returned int:2.  Error: The system
cannot find the file specified.

int RegOpenKeyExA (...)
  Parameters
    HKEY__ hKey = 0x15E
    char* lpSubKey = 0x0012D614 "VersionIndepenc
    unsigned int ulOptions = 0
```

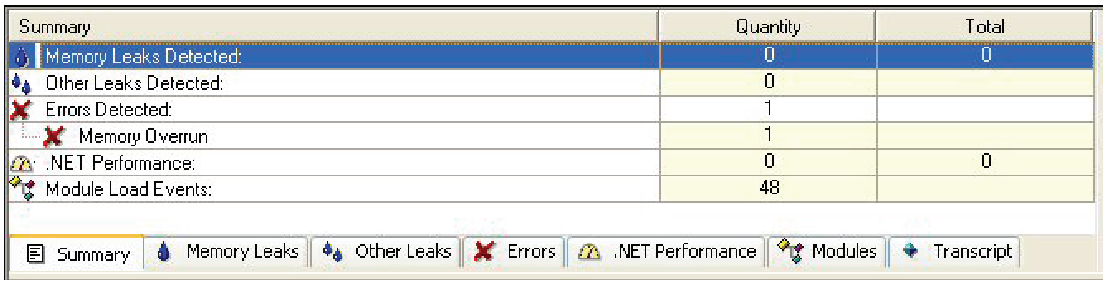
Current Call Stack - Thread 0 [0x0340]

Function	File	Line / Offset
CRegKey::Open	atlbase.h	4875
CRegParser::RegisterSubkeys	statreg.h	1187
CRegParser::RegisterSubkeys	statreg.h	1307
CRegParser::RegisterSubkeys	statreg.h	1307
CRegParser::RegisterBuffer	statreg.h	1044
CRegObject::RegisterFromResou...	statreg.h	400
CRegObject::ResourceRegister	statreg.h	412
CAtlModule::UpdateRegistryFrom...	atlbase.h	5432
CComModule::UpdateRegistryFro...	atlbase.h	4121
UpdateRegistry	view.h	53
CComModule::RegisterServer	atlbase.h	6407
DllRegisterServer	iface.cpp	82

```
c:\program files\micro focus\devpartner studio\examples\bugbench\iface\view.h

public:
        CView()
        {
        }

DECLARE_REGISTRY_RESOURCEID(IDR_VIEW)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CView)
        COM_INTERFACE_ENTRY(IView)
```

Figure 2-2. Error Detection main window

After a session completes, use the **Summary** tab of the **Results** pane to begin reviewing the data (see Figure 2-3).



Summary	Quantity	Total
Memory Leaks Detected:	0	0
Other Leaks Detected:	0	
Errors Detected:	1	
Memory Overrun	1	
.NET Performance:	0	0
Module Load Events:	48	

Figure 2-3. Results Pane displaying summary tab

The **Summary** tab provides an overview of all errors and leaks detected in the current session. Double-click on an event to navigate to a tab containing more detail about the selected event.

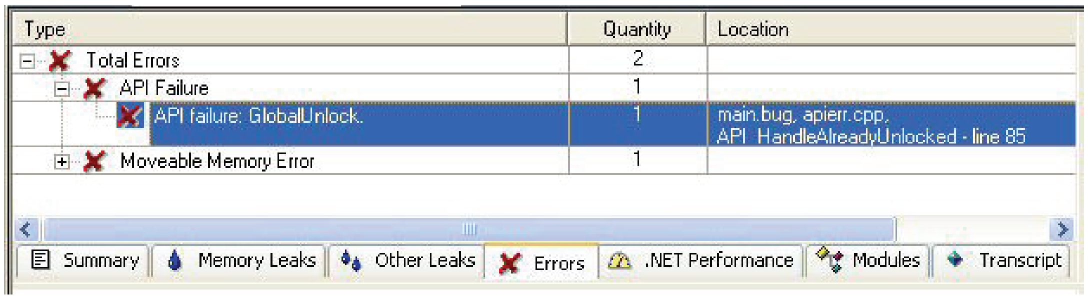
1 Examine the **Summary** tab of the Results pane for an overview of the errors and leaks detected.

2 Double-click on an error listed on the **Summary** pane.

The tab pertaining to the type of error or leak appears. This tab categorizes errors and makes it easier to focus on recurring errors, so you can diagnose and fix them.

3 Fully-expand a category and select a specific leak, error, or event.

The highest level of the list presents the categories of leaks, errors, and events. Fully-expanding the category displays the individual errors, leaks, and events detected (see Figure 2-4 on page 28).



Type	Quantity	Location
Total Errors	2	
API Failure	1	
API failure: GlobalUnlock.	1	main.bug, apierr.cpp, API_HandleAlreadyUnlocked - line 85
Moveable Memory Error	1	

Figure 2-4. Errors tab displaying selected error

The Results pane includes the following tabs: **Memory Leaks**, **Other Leaks**, **Errors**, **.NET Performance**, **Modules**, and **Transcript** (see Figure 2-4 on page 28). From these tabs, you can perform other actions to categorize or evaluate the data presented:

You can also access source code for a specific error by right-clicking on the specific error in one of the Results pane tabs, and selecting **Edit Source**. This opens the source file in the Source pane at the line of code that generated the error.

- To sort data on these tabs, click a column header (such as **Type**, **Quantity**, or **Deallocator** in the **Other Leaks** tab).
- For additional information about an event on a tab, right-click on the event and choose **Explain.**
- To view an event in the context of other events in your application, right-click on the event and choose **Locate in Transcript**. The **Transcript** tab provides a chronological list of all events that occurred within your application

4 Examine the Details pane (see Figure 2-2 on page 27).

The top of the Details pane describes the selected error in detail. Below the description is the current call stack.

The upper right section of the Error Detection window shows the Details pane. Information displayed in the Details pane depends on the currently selected event. The error or event is always described in more detail, but the Details pane can also display call stacks, P/Invoke use-count graphs, COM use-counts, and more.

5 Examine the call stack.

Depending on the type of error, the call stack can show where the error or leak was detected, or where it was allocated. If more than one call stack is available, you can switch between them using the drop-down list.

The bottom section of the main error detection window is called the Source pane. The Source pane displays the source file associated with the currently selected call stack. The source code changes when you select a different call stack in the Details pane.

6 Examine the Source pane.

The Source pane displays the code associated with the currently selected call stack, and highlighting the location where the error or leak was detected or allocated.

7 Review the source code to determine why an error or leak was detected.

8 Right-click in the Source pane and select **Edit Source**.

The source file opens in the Visual Studio editor, at the same location displayed in the Source pane.

9 Edit your source code to repair the error, and save your solution.

You have now identified and been placed inside the editor to resolve an error or leak in your code using error detection.

Saving Session Files

Tip: You can configure error detection to prompt you to save session files by selecting **File > Close** using the **General** settings.

Saving results in a session file lets you:

- Look back at the kinds of leaks and errors you have previously encountered
- Export session data to XML (see "Exporting Data to XML" on page 54). Exporting lets you share data with others, compare data between sessions, and build a database of trends
- Fix errors discovered in this session at any time.

Session files are saved with a `.dpbcl` extension. The default location for session files is in the same folder as your executable.

10 Select **File** > **Save Selected Items As** to save your session file.

11 Use the Save File As dialog box to select a location and name for the session file.

Steps to save the session file vary based on whether error detection is run in Visual Studio or in the standalone application.

Saving the Session File from Visual Studio

1. Select **File > Save Selected Items As**.
2. Use the File Save As dialog box to select a location and name for the session file.

Saving the Session File from the Standalone Application

1. Choose **File > Save Session Log As**.
2. Use the File Save As dialog box to select a location and name for the session file.

This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running an error detection session, continue reading the rest of this chapter for additional information. Refer to the Advanced Error Detection Techniques guide for more in-depth discussion of advanced topics, or refer to the DevPartner online help for task-based information.

Deciding When to Use ActiveCheck vs. FinalCheck

DevPartner can analyze Windows applications with both ActiveCheck™ and FinalCheck™ technologies.

Understanding ActiveCheck

ActiveCheck technology refers to the standard operation of checking for errors, leaks, and events without instrumentation of the source code. Because it does not require code instrumentation, ActiveCheck detects errors without requiring a recompile or relink. ActiveCheck is enabled in every error detection session.

ActiveCheck can do the following:

- Report API validation errors at run-time
- Report memory and resource leaks when your program terminates
- Isolate errors to the line where the memory or resource is allocated or the error is generated
- Identify potential deadlocks

DevPartner analyzes the program when using error detection with ActiveCheck. It monitors API calls, memory allocations and deallocations, windows messages, and other significant events. It then uses this data to detect errors and to provide a complete trace of program execution. You can even check programs that do not have source code available.

Because ActiveCheck requires no compilation or relinking overhead, you can use it daily. Use ActiveCheck throughout the software development cycle to find API validation errors, deadlocks, resource leaks, and COM interface leaks.

Table 2-1 and Table 2-2 list errors detected by ActiveCheck.

Table 2-1. API, COM, and memory errors detected by ActiveCheck

API and COM Errors	Memory Errors
• COM interface method failure • Invalid argument • Invalid COM interface method argument • Parameter range error • Questionable use of thread • Windows function failed • Windows function not implemented	• Dynamic memory overrun • Freed handle is already unlocked • Handle is already unlocked • Memory allocation conflict • Pointer references unlocked memory block • Stack memory overrun • Static memory overrun

Table 2-2. Deadlock-related, .NET, pointer, and leak errors detected by ActiveCheck

Deadlock-related Errors	.NET Errors	Pointer and Leak Errors
• Deadlock • Potential deadlock • Thread deadlocked • Critical section errors • Semaphore errors • Mutex errors • Event errors • Handle errors • Resource usage and naming errors • Suspicious or questionable resource usage • Windows event errors	• Finalizer errors • `GC.Suppress` finalize not called • Dispose attributes errors • Unhandled native exception passed to managed code	• Interface leak • Memory leak • Resource leak

Understanding FinalCheck

FinalCheck is a patented technology that inserts diagnostic logic into your code when compiled. With FinalCheck, DevPartner can pinpoint errors to the exact statement where they occur.

Use FinalCheck for key project milestones and for detecting errors that are difficult to find. FinalCheck is a superset of ActiveCheck that finds all the errors ActiveCheck finds, plus those listed in the following table.

Table 2-3. Additional errors detected by FinalCheck

Memory Errors	Pointer and Leak Errors
• Reading overflows buffer • Reading uninitialized memory • Writing overflows buffer	• Array index out of range • Assigning pointer out of range • Expression uses dangling pointer • Expression uses unrelated pointers • Function pointer is not a function • Memory leaked due to free • Leak due to leak • Memory leaked due to reassignment • Memory leaked leaving scope • Returning pointer to local variable • Leak due to unwind • Leak due to module unload • Leak due to thread ending

Comparing ActiveCheck and FinalCheck — An Example

DevPartner records memory block allocations that use `new` or `malloc` and stores the pointer in a local variable. If the variable value is re-assigned without first either deallocating the memory block or assigning the pointer to another variable, a leak occurs in the application.

- **Using ActiveCheck:** DevPartner reports that the block allocated by `malloc` or `new` leaked and points to the line where the memory is allocated. The error is reported when the application stops.
- **Using FinalCheck:** DevPartner reports the location where the block is allocated and highlights the line where new value is assigned into the last remaining variable referencing the block. The error is reported when it occurs.

Using the Program Error Detected Dialog Box

DevPartner displays the Program Error Detected dialog box (see Figure 2-1 on page 26) when it detects a severe application error.

The top of the Program Error Detected dialog box describes the error detected. Below the error description is one or more tabs, each associated with a call stack corresponding to a location within the application. Review the reported error and the source information to help locate the source of the problem and correct it.

Understanding the Actions You Can Take

Explain, **Memory and Resource Viewer**, **Debug**, **Copy** and **Suppress** buttons appear on the Program Error Detected dialog box.

Explain

Click **Explain** to obtain detailed explanations of each error, sample code, and a list of possible solutions to correct the problem.

Memory and Resource Viewer

Click **Memory/Resource Viewer** to view a detailed accounting of memory and resources that have not been freed. For more information, see "Understanding the Memory and Resource Viewer Dialog Box" on page 33, and the *Advanced Error Detection Techniques* guide.

Copy

Click **Copy** to transfer the contents of all windows and tabs (except the **Source** pane) to the clipboard. You can then paste this information into other applications.

Suppress

Click **Suppress** to open a dialog box that enables you to suppress the current error. For more information on how and why to use suppressions, refer to "Understanding the Suppression and Filtering Dialog Boxes" on page 35, and the *Advanced Error Detection Techniques* guide.

Debug

Debug appears at the bottom of the dialog box when you are working in Visual Studio, but is not available in the standalone application. Click **Debug** to open your code in the Visual Studio debugger.

Halt

Click **Halt** to stop the application. This effectively kills the process. There may be other means by which you would rather stop the application.

Continue

Click **Continue** to acknowledge the error, close the dialog box and continue executing the application. The error is saved to the session file for later reviewing in the Results pane.

Understanding the Memory and Resource Viewer Dialog Box

Access the Memory and Resource Viewer the following ways:

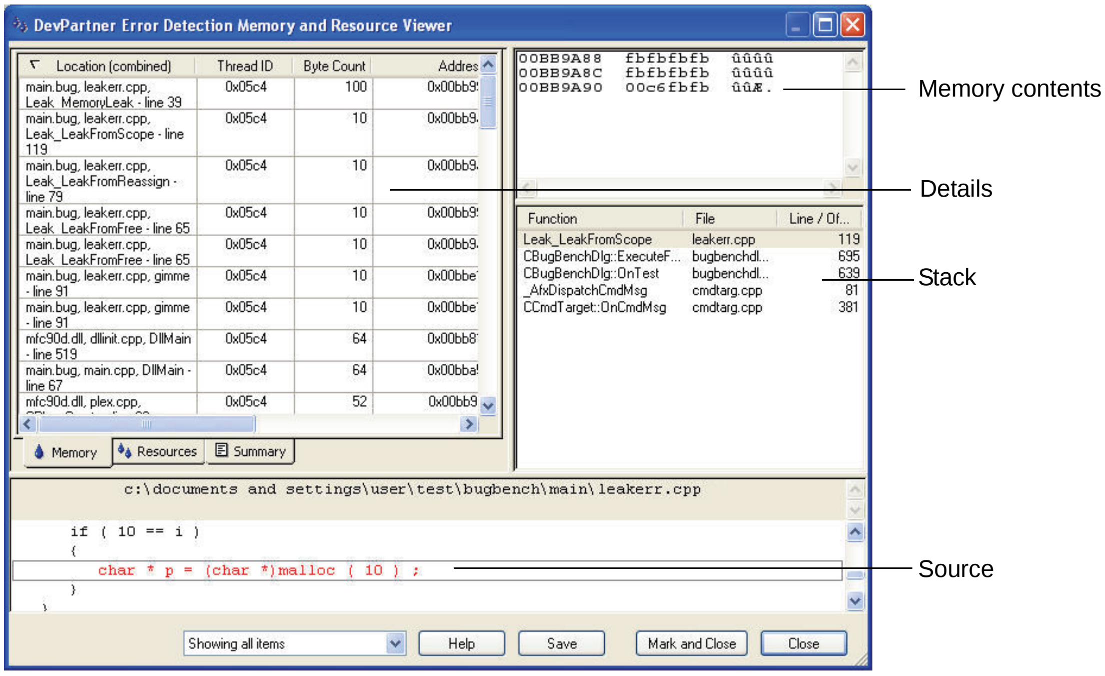
- Choose **Debug > DevPartner / Memory Resource Viewer** from the Visual Studio menu while running or debugging an application.
- In standalone BoundsChecker, choose **Program > Memory Resource Viewer** from the menu, or click the **Memory Resource Viewer** button.
- Click the **Memory/Resource Viewer** button in the Program Error Detected dialog box.

The Memory and Resource Viewer lets you analyze memory and resource allocations that have not been freed.

For example, most memory analysis tools can not determine what happens with memory during the execution of an application. Leaked memory or resources are only reported after the application stops. The Memory and Resource Viewer provides a “snapshot” of the memory and resources, taken at any point in a program’s execution. You can also “mark” the currently allocated memory blocks or resources, limiting the view of blocks allocated after a program’s initialization or over the course of a transaction.

These capabilities can be especially useful in situations where:

- 24/7 server applications may never end during regular use
- An application may stop responding from resource exhaustion
- An application may consume large amounts of memory that is automatically cleaned up at program termination



Location (combined)	Thread ID	Byte Count	Addres
main.bug, leakerr.cpp, Leak_MemoryLeak - line 39	0x05c4	100	0x00bb9
main.bug, leakerr.cpp, Leak_LeakFromScope - line 119	0x05c4	10	0x00bb9.
main.bug, leakerr.cpp, Leak_LeakFromReassign - line 79	0x05c4	10	0x00bb9.
main.bug, leakerr.cpp, Leak_LeakFromFree - line 65	0x05c4	10	0x00bb9
main.bug, leakerr.cpp, Leak_LeakFromFree - line 65	0x05c4	10	0x00bb9.
main.bug, leakerr.cpp, gimme - line 91	0x05c4	10	0x00bbe
main.bug, leakerr.cpp, gimme - line 91	0x05c4	10	0x00bbe
mfc90d.dll, dllinit.cpp, DllMain - line 519	0x05c4	64	0x00bb8
main.bug, main.cpp, DllMain - line 67	0x05c4	64	0x00bba
mfc90d.dll, plex.cpp,	0x05c4	52	0x00bb9

```
00BB9A88   fbfbfbfb   ûûûû
00BB9A8C   fbfbfbfb   ûûûû
00BB9A90   00c6fbfb   ûûÆ.
```

Function	File	Line / Of...
Leak_LeakFromScope	leakerr.cpp	119
CBugBenchDlg::ExecuteF...	bugbenchdl...	695
CBugBenchDlg::OnTest	bugbenchdl...	639
_AfxDispatchCmdMsg	cmdtarg.cpp	81
CCmdTarget::OnCmdMsg	cmdtarg.cpp	381

```
        c:\documents and settings\user\test\bugbench\main\leakerr.cpp

    if ( 10 == i )
    {
        char * p = (char *)malloc ( 10 ) ;
    }
}
```

Figure 2-5. The Memory and Resource Viewer dialog box

Exploring the Memory and Resource Viewer User Interface

To access the Memory and Resource Viewer dialog box, click **Memory/Resource Viewer** in the Program Error Detected dialog box.

The Memory and Resource Viewer dialog box is made up of four panes:

- **Memory contents pane**

  Displays the content of memory blocks in a variety of formats. Not available for resources.

- **Details pane**

  Includes separate **Memory**, **Resources**, and **Summary** tabs. Displays details about each memory and resource allocation.

- **Stack pane**

  Displays a memory dump and callstack information for entries in the **Memory** tab; displays a description and callstack information for entries in the **Resources** tab.

- **Source pane**

  Displays the source code corresponding to a callstack entry (when it is available).

Saving Memory and Resource Viewer Contents

Click **Save** to record the current contents of the Memory and Resource Viewer dialog box as a text file that you can review later.

Setting a Reference Point

Click **Mark and Close** to set a reference point for recording memory and resource data. This lets you compare memory and resource allocations before and after the event where you marked the reference point.

Understanding the Suppression and Filtering Dialog Boxes

DevPartner provides Suppression and Filter dialog boxes, which allow you to reduce the data collected or displayed. The intent of either method is to limit the data to a manageable subset for analysis.

For example, you can suppress call validation errors from `FindResourceA` in Kernel32 or for all calls in Kernel32. After you make this selection, you can apply it to a variety of different selection criteria within your application. DevPartner defaults to the least restrictive option (see Figure 2-6).

When you apply a suppression or filter, you can also:

- Enter a comment to describe why a given suppression or filter was created.
- Choose to apply the suppression or filter to the current run or future runs.
- Create suppression or filter files as a way to store the suppression or filter instructions for reuse or to share.

Suppressing Errors

Suppressing errors instructs DevPartner to skip over any future occurrences of those errors. Suppressed errors are not recorded in the log and are not displayed in the Program Error Detected dialog box. To suppress an error:

- Click **Suppress** when the error appears in the Program Error Detected dialog box.
- Right-click on a specific error in one of the panes of the error detection main window, and select **Suppress.**

Creating and Saving Suppression Files

You can create multiple suppression files, which creates additional suppression libraries for the DLLs that make up a large application. You can easily reuse or share suppressions among members of a development team.

When you first open an `.EXE` in error detection, a default suppression file is created in the same folder as the `.EXE` you are checking.

The following sections describe the ways you can create a suppression file in error detection.

From the Suppression Files Dialog Box

To create a suppression file from the Suppression Files dialog box, follow these steps:

1 Access the Suppression Files dialog box.

- ◇ **Visual Studio:** Select **DevPartner > Error Detection Rules > Suppressions**.
- ◇ **Standalone:** Select **Program > Rules > Suppressions**.

2 Click **Add**.

3 Type the name you want to assign to the suppression file in the **File Name** text box, then click **Open**.

4 Click **Yes** to confirm.

The suppression file you created is added to the **Available Suppression Files** list in the top pane of the Suppression Files dialog box.

5 Click **OK**.

The suppression file has been created, but is empty until you add suppressions (see “Adding Entries to a Suppression File” on page 37). Added suppressions are not saved until you close the current error detection session.

From the Suppression Dialog Box

To create a suppression file from the Suppression dialog box, follow these steps:

1 After you complete a session, right-click on a specific error in the **Memory Leaks**, **Other Leaks**, **Errors**, **.NET Performance**, or **Modules** tab, and select **Suppress**.

2 In the Suppression dialog box, click the browse button (**...**) to the right of the **Location** field. The Add Suppression File dialog box opens.

3 Type the name you want to assign to the suppression file in the **File Name** text box, then click **Open**.

4 Click **Yes** to confirm.

5 Click **OK**.

The suppression file contains the instruction to suppress the error that you right-clicked in Step 1. Any added instructions and suppressions are not saved until you close the current error detection session.

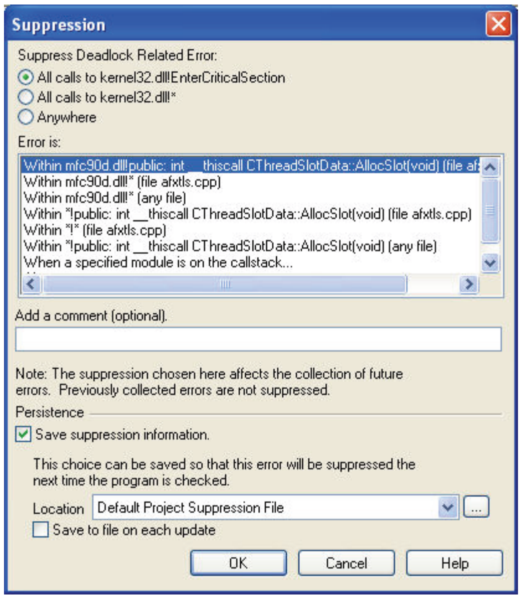
Adding Entries to a Suppression File

To add an entry to a suppression file, follow these steps:

1. In the Results pane, select a tab: **Memory Leaks**; **Other Leaks**; **Errors**; **Modules**; or **Transcript**.
2. Right-click a specific error, leak, or module within that tab and select **Suppress**. The Suppression dialog box opens (see Figure 2-6 on page 38).
3. Select the type of suppression to add.

   The top pane displays various suppression options. The selections vary depending on the event you select (whether it is an error, a leak, or a module), and the context in which error detection encountered it.
4. If needed, type a comment to describe the suppression entry.

   Comments can be valuable when you update a suppression file, especially if the suppression is address-based and a third-party vendor ships a new or updated library.
5. To save this suppression, select the **Save Suppression Information** check box.
6. To specify the location of the suppression file in which to add this entry, select a file from the **Location** drop down menu.
   - ◇ If you make no selection, the entry is added to the default program suppression file.
   - ◇ If you have not yet added suppression files to your program, the default program suppression file is the only choice.
   - ◇ To specify a suppression file in another location, click the browse button (**...**) (immediately to the right of the **Location** drop down menu) and select a different suppression file.
7. Click **OK** to continue. The entries you add are not saved until you close the current error detection session.


Suppression

Suppress Deadlock Related Error:

[✓ All calls to kernel32.dll!EnterCriticalSection]
All calls to kernel32.dll!*
Anywhere

Error is:

```
Within mfc90d.dll!public: int __thiscall CThreadSlotData::AllocSlot(void) (file af
Within mfc90d.dll!* (file afxtls.cpp)
Within mfc90d.dll!* (any file)
Within *!public: int __thiscall CThreadSlotData::AllocSlot(void) (file afxtls.cpp)
Within *!* (file afxtls.cpp)
Within *!public: int __thiscall CThreadSlotData::AllocSlot(void) (any file)
When a specified module is on the callstack...
```

Add a comment (optional).

Note: The suppression chosen here affects the collection of future errors. Previously collected errors are not suppressed.

Persistence

[✓ Save suppression information.]

This choice can be saved so that this error will be suppressed the next time the program is checked.

Location: Default Project Suppression File

Save to file on each update

Figure 2-6. The Suppression dialog box

Filtering Errors

Filtering hides events already recorded in a **`.dpbcl`** log file. DevPartner finds these errors but either hides them from view in the Results pane or displays them with the appearance you specified under **Fonts and Colors**. To select errors that you want to filter:

- ◆ Right-click on a specific error in one of the panes of the error detection main window, and select **Filter.**
- ◆ Select a specific error in one of the panes of the error detection main window, and click the **Filter** button on the toolbar.

If you remove a filtering instruction, the associated errors are no longer filtered and appear in the Results pane.

Creating a Filter File

There are two ways to create a filter file.

From the Filter Files Dialog Box

To create a filter file from the Filter Files dialog box, follow these steps:

1. Open the Filter Files dialog box.
2. Click **Add**.

   The Add Filter File dialog box opens.
3. Type the name you want to assign to the filter file in the **File Name** text box, then click **Open**.
4. Click **Yes** to confirm.

   The filter file is added to the **Available Filter Files** list in the top pane of the Filter Files dialog box. The filter file is empty and has no effect on your view of program results.

Added filters are not saved until you close the current error detection session.

From the Filter Dialog Box

To create a filter file from the Filter dialog box, follow these steps:

1. After you complete a session, right-click a specific event or error in the **Memory Leaks**, **Other Leaks**, **Errors**, **Modules**, or **Transcript** tab, then select **Filter**. The Filter dialog box opens (see Figure 2-6 on page 38).
2. Click the browse button (**...**) to the right of the **Location** field.

   The Add Filter File dialog box opens.
3. Type the name you want to assign to the filter file in the **File Name** text box, then click **Open**.
4. Click **Yes** to confirm. The filter file contains the instruction to hide the error or event that you selected in Step 1. Added filters are not saved until the current error detection session is closed.

Adding Entries to a Filter File

To add an entry to an existing filter file, follow these steps:

1. In the Results pane, select a tab: **Memory Leaks**; **Other Leaks**; **Errors**; **Modules**; or **Transcript**.
2. Right-click a specific error, leak, or module within that tab and select **Filter**. The Filter dialog box appears (see Figure 2-6 on page 38).
3. Select an option.

   These options are listed in the top pane of the dialog box. The selections vary depending on the event you select (whether it is an error, a leak, or a module), and the context in which error detection encountered it.
4. If needed, type a comment to describe the entry. Comments can be valuable when you update a filter file, especially if the filter is address-based and a third-party vendor ships a new or updated library.
5. If you want to save this entry to use again, select the **Save Filter Information** check box.
6. To specify the location of the filter file in which to save this entry, select a file from the **Location** drop down menu.
   - ◇ If you make no selection, the entry is added to the default program filter file.
   - ◇ If you have not yet added filter files to your program, the default program filter file is the only choice.
   - ◇ To specify a filter file in another location, click the browse button (**...**) (immediately to the right of the **Location** drop down menu) and select a different filter file.
7. Click **OK** to continue. Added entries are not saved until you close the current error detection session.

Viewing and Hiding Filtered Errors

Use the **View Filtered Errors** toolbar icon to toggle between viewing and hiding filtered errors in the Results pane.

Removing Filter Entries

To remove a filter entry you no longer want, select **DevPartner > Error Detection Rules > Filters**. Select the filter file containing the entry, and clear its associated check box.

Understanding Call Validation

When you enable Call Validation, DevPartner validates over 5,000 Windows API calls. DevPartner checks for a large number of events, including (but not limited to) the following:

- Handle and pointer errors
- Flags
- Range checks
- API and method failures
- Invalid structure sizes
- Memory access failures

If you determine that flag checking or range checking generates unwanted errors that do not apply to the problem you are solving, clear the **Flag, range and enumeration arguments** check box. **Call Validation** continues checking return values and, more importantly, handles, and pointers passed to or from Windows calls.

Enabling Memory Block Checking

When you enable memory block checking, **Call Validation** performs a more detailed analysis of all calls to the C run-time library and a number of other calls. Memory block checking decreases overall performance, but may be useful when diagnosing hard-to-find errors. By default, this setting is disabled.

Using the Settings Dialog Box

Tip: Use the configuration file management functions in the Settings dialog box to save sets of error-checking parameters as configuration files. When you are working with multiple projects, you can load, edit, and associate these configuration files with the different projects on which you are working.

The DevPartner settings enable you to:

- Select only the types of data collection needed for a particular problem
- Enable or disable portions of each major type of data collection
- Control what portions of your program are analyzed
- Use the default DevPartner settings to find the most common errors with the minimum impact on performance

You can access the Settings dialog box in the following ways:

- **Standalone:** Select **Program > Settings**
- **Visual Studio:** Select **Tools > Options** and then select **DevPartner > Error Detection** from the tree view.

The Settings dialog box has a tree view that shows the major settings categories. When you select a category, the dialog box displays the detailed settings for the category.

DevPartner Studio and the DevPartner standalone application use the same tree view and settings dialog boxes.

All groups of settings follow the same basic structure. You can enable or disable major types of data collection by selecting the top-level check box in the dialog box.

There are other settings under each top-level check box that further define how DevPartner analyzes an application. Change the settings to customize the error detection process.

For example, you can make trade-offs between detecting a broad or narrow range of errors:

- Broad range — Many data types, many related settings selected
  - Detects more errors
  - Has potential for more false positives
  - Reduces performance (due to larger number of errors detected)
  - Creates larger log files
- Limited range — Few data types, few related settings selected
  - Provides a narrow focus on a particular function
  - Detects fewer errors
  - Can miss relevant errors
  - Has a greater chance of seeing only those errors pertaining to the problem at hand
  - Provides faster performance
  - Creates smaller log files

Setting General Properties

General properties are the first to display when you access the Program Settings dialog box.

- **Log events:** Select to enable event logging. (You can also enable event logging from other parts of DevPartner error detection.)
- **Display error and pause:** Controls the display of the Program Error Detected dialog box, which pops up for certain errors, pausing execution of your program.
- **Prompt to save program results:** When selected, you are prompted to save program results before you stop the program or close the error detection session.
- **Show memory and resource viewer when application exits:** When selected, DevPartner error detection opens the Memory/Resource Viewer dialog box upon stopping the application you are testing.
- **Source file search path:** Specify full paths to the source file(s) you want to include in this configuration.

The following settings are only available in the error detection standalone application:

- **Override symbol path:** Specify the full path to the symbol file(s) you want to include in this configuration. Click the ellipsis button (**...**) to the right of this field to open the Symbol Path dialog box.
- **Working directory:** Specify the working folder for the target process.

Note: If an application does not start, the problem might be caused by a working folder that is read-only. Some applications require write access to the working folder.

- **Command line arguments:** Specify any arguments to be passed through the command line to applications.

Note: If DevPartner error detection does not correctly start an application, check the command line arguments. These arguments are especially important for COM server applications.

Setting Data Collection Properties

The Data Collection program settings control the following parameters for error detection:

- **Call parameter encoding depth:** Specify the amount of detail gathered on the parameters of a call. A low value speeds up processing, but does not report deeper levels of detail referenced by pointers. A higher value reports deeper call details, but slows down processing and increases the size of the log file.
- **Use raw stack capture method:** Select this check box to use a predetermined method that optimizes performance and speed when capturing the allocation stack.
- **Maximum call stack depth on allocation:** Specify the maximum depth of the call stack tracked for every allocation. Because allocations are made frequently and do not often result in errors, a large value may impact performance. This field is disabled if **Use raw stack capture method** is selected.
- **Maximum call stack depth on error:** Specify the maximum depth of the call stack walked through for reported errors. You can set this value as high as you require without an adverse effect on performance, as long as you have enough log file space.
- **NLB file directory:** (This field is required) Select the location where generated NLB (optimized type library) files are saved. Typically this is the same location as your project, so that removal of the project and NLB files is simplified. If you specify a location that does not exist, error detection prompts you to select a valid location when you run your application. You can also use the **Browse** button (**...**) to browse your system and specify the folder where you want to save generated NLB files. NLB files contain all API description file information required for error detection.

Setting API Call Reporting Properties

Tip: Do select an API function when you know the program makes API calls you do not need to check. Limiting data collection helps improve performance.

Use API call reporting to record calls that your application makes to system functions, as well as the parameters and return values. DevPartner error detection records structure information for return values and parameters based on the **Encoding Depth** specified under Data Collection settings.

To enable API call reporting, and make the API check boxes and modules active, select the **Enable API call reporting** check box. The following settings control API logging for error detection:

- **Collect window messages:** Select to collect windows control messages as part of API logging.
- **Collect API method calls and returns:** Select to collect the API method calls and returns for the modules selected in the API module tree view.
- **View only modules needed by this application:** Select this check box to display only the API modules needed by your program in the tree view. Clear the check box to expand this tree view and display all available API modules.
- **API Modules Tree View:** Displays the API modules associated with the project. Click the plus (+) symbol next to an item to see the functions it contains. Check boxes next to each item enable you to select specific modules or functions for API logging.

Tip: Selecting **Collect window messages** increases log file size. For best results, select this feature only to debug window message problems.

Enabling call reporting can significantly increase the size of the log file. To minimize log file size, consider collecting call reporting data only for a selected portion of the application. Here are some ways to limit the portion to be checked:

- Use the check boxes in the Modules tree view to deselect API modules that do not need to be checked.
- Use **Modules and Files** to limit the scope of logging.
- Add API calls that enable or disable event logging. Refer to the comments included in `NmApiLib.h`. This file, part of the DevPartner installation, defines the event reporting APIs exported by DevPartner error detection.
- Turn off event logging.

Setting Call Validation Options

When enabled, Call Validation monitors calls from the application to the operating system libraries and COM method calls. It attempts to validate the parameters passed, and check that the call returned a value indicating success. The following elements control aspects of Call Validation for error detection:

- **Enable call validation:** Select this check box to enable call validation components.
- **Enable memory block checking:** Select this check box to enable validation of more extensive memory checking of parameters referring to memory. This feature is inactive until you select **Enable memory tracking** under **Memory Tracking** in the **Program Settings** tree view.

  When you select **Enable memory block checking**, DevPartner error detection performs more extensive checking. The results might be more accurate and might catch more bugs. Sessions with this option enabled take longer to complete.

- **Fill output arguments before call:** Select this check box to fill output arguments with the pattern specified in the Memory Tracking settings under **Fill on allocation**.
- **COM failure codes:** Select this check box to enable checking of any COM method return values.

  Many COM methods in normal use report a “Not Implemented” error. Disabling this check might significantly reduce the number of errors reported.
- **Check for COM "Not Implemented" return code:** Select this check box to enable checking for the `HRESULT E_NOTIMPL` ("Not Implemented") return code. DevPartner error detection checks only COM interfaces that are included in DLLs selected under **DLLs to check for API errors (failures or invalid arguments)** in this dialog box.
- **API failure codes:** Select this check box to enable the checking of return values from APIs residing in the selected DLLs.
- **Check invalid argument errors (COM or API):** Select one or both of these check boxes to enable the checking of the arguments (parameters) to APIs in the selected DLLs and/or COM interfaces that error detection supports.

Tip: To improve performance and reduce the number of errors reported, select these options only as required. To reduce the number of false call validation errors, select **Handle and Pointer Arguments** and clear **Flag, Range and Enumeration Arguments**.

- **Category:** (**Handle and pointer arguments** or **Flag, range and enumeration arguments**) Available when you select one or both **Check invalid argument errors selections (COM or API)**. Select one or both of these check boxes to enable argument checking based on the type of argument.
- **Check statically linked C run-time library APIs:** Available when you select **API failure codes** or **Invalid argument errors: API**. Select this check box to enable the checking of static C run-time calls. If you are not using the static C run-time library, clear this selection to avoid seeing errors in third-party libraries.

Tip: Disabling DLLs from this list can reduce the number of unwanted errors. It can also improve performance.

- **DLLs to check for API errors (failures or invalid arguments):** Available when you select **API failure codes** or **Invalid argument errors: API**. Select this check box to enable API argument and return value checking in the listed DLLs.

  You can use a tool (such as Depends, provided with Visual Studio) to find the DLLs and APIs within a DLL that an application uses.

Enabling Memory Overwrite Detection on API Calls

Checking for damage to memory blocks caused by API calls (such as `strcpy`) is not enabled by default. To enable memory overwrite detection on API calls, follow these steps:

1. Select the **Enable memory tracking** check box.
2. In the **Program Settings** tree view, select **Call Validation**.
3. Select the **Enable call validation** check box.

4 Select the **Enable memory block checking** check box.

Setting COM Call Reporting Properties

Use COM call reporting to record calls to COM interfaces as well as the returns for the interfaces selected in the **All Interfaces** tree. DevPartner error detection records parameter values and the returned `HRESULT`.

Select only the interfaces you need to check. Decreasing the number of interfaces checked decreases the size of the log file and improves performance.

To enable COM call reporting, and activate the list of COM interfaces, select the **Enable COM method call reporting on objects that are implemented in the selected modules** check box. Use the following controls to configure COM call reporting:

- **Report COM method calls on objects implemented outside of the listed modules:** Select this check box to configure error detection to report the COM method calls and returns for the interfaces not listed in the **All Interfaces** tree.
- **All Components Tree View:** Displays the COM interfaces associated with the project. Click the plus (+) symbol next to the **All Components** entry to see a complete list of COM interfaces. Check boxes next to each item let you select specific interfaces for COM call reporting.

Setting COM Object Tracking Options

Use COM object tracking to monitor your program for leaked `COM` objects. Object leaks are displayed in the **Other Leaks** tab of the Results pane. When you select an object leak error in the **Other Leaks** tab, you can examine the calls to `AddRef()` and `Release()` on your object to try to locate the missing call to `Release()`.

To improve performance, select a subset of All `COM` Classes. Consider selecting all `COM` classes only when running an initial pass of your application, or when making a final QA pass.

To enable `COM` object tracking, and activate the **All COM Classes** tree view, select the **Enable COM object tracking** check box.

Using the **All COM Classes** tree view, select the `COM` classes that you want to monitor. If you do not see the `COM` class for your application, click **Refresh from Registry** to update the list.

Tip: Most vendors name their objects with a common prefix.

Setting Deadlock Analysis Options

Use Deadlock Analysis to monitor multi-threaded applications for deadlocks. This includes the following types of analysis:

- Monitoring and reporting of deadlocks as they occur in the application.
- Monitoring the usage patterns of the synchronization objects within your application for potential deadlocks.

To enable Deadlock Analysis, and activate the other Deadlock Analysis controls, select the **Enable deadlock analysis** check box.

The following settings control the Deadlock Analysis behavior:

- **Assume single process:** When selected, error detection assumes that all named synchronization objects used within your application are used only within the process. Clear this check box to relax some of the deadlock detection rules associated with named synchronization objects.
- **Enable watcher thread:** Select this check box to create a watcher thread in your application to monitor for localized deadlocks. By default, this feature is disabled to prevent error detection from interfering with your application.

  If your application becomes unresponsive and appears to deadlock, enabling this feature allows error detection to perform more detailed analysis of your application.

  If you write complex `DLL_THREAD_ATTACH` logic that does not expect to encounter extra threads in the process, you should not enable this option.
- **Generate errors when:** Use the following selections to specify when error detection should report deadlock errors:
  - **A critical section is re-entered:** Select to generate a warning if you attempt to re-enter a critical section that your thread already owns. Although re-entering a critical section is not an error, your application must enter and leave the critical section the same number of times.
  - **A wait is requested on an owned mutex:** Select to generate a warning if you attempt to wait on a mutex that your thread already owns.
  - **Number of historical events per resource:** Enter the number of call stacks to record for each synchronization object reported in an error or warning.

    The stack information associated with each synchronization object enables you to determine why a synchronization object is in a given state. This can help you debug deadlock situations.

Note: Increasing the number of call stacks maintained for each synchronization object consumes additional memory in your application and has an effect on application performance.

  - **Report synchronization API timeouts:** Select to report an error when a wait on a synchronization object times out without the wait being successfully completed.

    Enable this option to monitor synchronization object API failures without having to enable API call reporting for all Windows calls.

    You can use the **Report wait limits or actual waits exceeding** option to enforce a maximum wait policy within your application.
  - **Report wait limits or actual waits exceeding (seconds):** Active after you select Report timeouts. Error Detection checks the time-out value passed to synchronization object wait calls. If the time-out values exceed the limit you specify here, the call is reported as an error.

Note: Any wait specified as `INFINITE` is not flagged as an error.

- **Synchronization Naming Rules:** Select from these object standards:
  - **Don't warn about resource naming:** If selected, error detection does not warn you about named or unnamed resources encountered in your application.

Tip: If you are performing security audits, consider enabling the **Warn about named resources** option to determine if unexpected named resources are visible outside the process. Named resources are visible outside the process and should have proper security applied to them to prevent unauthorized use.

- ◇ **Warn about named resources:** Select to generate warnings for each named synchronization resource encountered within your application. You can use this check to locate named resources that can be manipulated outside of the application.
- ◇ **Warn about unnamed resources:** Select to generate warnings for each unnamed synchronization resource encountered in your application. You can use this check to find unnamed resources that might need to be named to be used by other processes or to meet a corporate naming convention.

By default, error detection does not produce warnings for either named or unnamed resources encountered during program execution.

Setting Memory Tracking Options

When you enable Memory Tracking, DevPartner error detection:

- Monitors all calls in your application that allocate and free memory
- Reports on memory not freed at the end of the application

Additionally, if you have built your application with FinalCheck instrumentation and you select **Enable FinalCheck**, error detection:

- Records instances where the last reference to an allocated block of memory goes out of scope
- Reports memory and pointer errors at the statement level throughout the run

To enable Memory Tracking, and activate all of the memory tracking options, select the **Enable memory tracking** check box. You must select the **Enable memory tracking** check box before you can enable **Memory block checking** under the Call Validation settings.

The following settings control the behavior of Memory Tracking:

- **Enable leak analysis only**: Select this check box to disable everything in Memory Tracking, with the exception of monitoring for leaks. Memory Tracking does not look for overruns, use of uninitialized memory, or dangling pointers. Call Validation memory block checking is also disabled because Memory Tracking is not evaluating any memory allocated by system modules.

  Some of the COM interface hooks do not get handled completely when this option is enabled.
- **Enable FinalCheck:** Select this check box to enable FinalCheck. When selected, error detection performs additional checks on FinalCheck instrumented modules. When disabled, these checks are not performed.
- **Show leaked allocator blocks:** Select this check box to enable the reporting of leaks on blocks that are used for suballocations. Suballocated blocks are normally created by memory allocation functions such as `malloc` or `new`. If you are writing your own memory allocators, enable this feature to monitor all memory in your application, including buffers

that are suballocated into blocks returned from functions such as `malloc` or `new`. DevPartner error detection monitors your custom memory allocators only after you list them in **`UserAllocators.dat`**. For more information about **`UserAllocators.dat`**, read the chapter "Working with User-Written Allocators" in the *Advanced Error Detection Techniques* guide.

- **Enforce strict reallocation semantics:** Select this check box to enable strict enforcement of semantics. When error detection enforces strict reallocation semantics, a pointer to memory that has been reallocated is treated as though it were a dangling pointer, and using that pointer generates an error. If strict reallocation semantics are not enabled, a reallocated pointer may be used as long as it points to the same memory location as the new pointer, and no errors are generated. For example:

```
char *ptrA = (char *) malloc(17);

// ptrA is now validly pointing to 17 bytes of memory.

char *ptrB = (char *) realloc(ptrA, 15);

// ptrB is now validly pointing to 15 bytes of memory.

// With strict semantics, ptrA is now an invalid pointer, regardless of the
value.

// Without strict semantics, ptrA is still valid as long as it equals ptrB
```

- **Enable Guard Bytes:** When enabled, guard bytes are inserted at the end of allocated memory blocks to detect memory overrun errors. Overruns can cause heap corruption or stack corruption, that, in turn, can cause random crashes and unexpected data overwrites.
  - **Pattern:** Enter the hexadecimal guard byte pattern. This pattern is used to determine if allocated memory blocks are overrun.
  - **Count:** Select the number of guard bytes to be used. If you encounter random heap corruption errors but error detection is not reporting heap-overrun errors, consider increasing the number of guard bytes. Doing so increases memory usage, but might detect a hard to find heap corruption error.
- **Check heap blocks at runtime:** Specify how often the entire heap is checked to see if guard bytes have been overwritten. DevPartner error detection always checks each block for overrun when it is freed. There are three options for additional checks:
  - On free
  - Use adaptive analysis
  - On all memory API calls
- **Enable fill on allocation:** When enabled, the fill pattern specified is applied to memory as it is allocated.
  - **Pattern:** Specify the hexadecimal fill pattern to be used.
- **Check uninitialized memory:** When selected, newly allocated memory is initialized with a known pattern and then checked for that pattern when the memory is referenced.
  - **Size:** Select the minimum number of bytes to check for the fill pattern. To reduce the number of false error reports, increase this value.

- **Enable poison on free:** Select this check box to enable poisoning of memory upon freeing it.
  - **Pattern:** Enter the pattern to be written to the memory location that is being poisoned.

Setting .NET Framework Analysis Options

Use .NET Framework analysis when you develop applications that use a mixture of unmanaged and managed code and unmanaged resources. Applications that use both managed and unmanaged code might incur a performance penalty. The data you gather here can help you evaluate the extent and severity of any such penalty. If you discover problems and lack the time to fix all of them, this analysis can help you decide which are the most serious.

To enable the .NET Framework analysis controls in this panel, select the **Enable .NET analysis** check box. Use the following controls to configure .NET Framework analysis:

- **Exception monitoring:** Select this check box to monitor instances where unmanaged or legacy code throws exceptions that are not handled, and are passed back to the managed code.

Note: Exceptions passed from unmanaged to managed code are likely to generate errors because the necessary handles are no longer in unmanaged code. You should carefully review any exceptions noted. Possible errors include partially initialized data structures, memory leaks, resource leaks, and so on.

- **Finalizer monitoring:** Select this check box to monitor incorrect use of unmanaged resources, such as failing to call the appropriate dispose method (leaks) or incorrect implementation of classes that encapsulate unmanaged resources.
- **COM interop monitoring:** Select this check box to monitor which class IDs are causing transitions between managed and unmanaged or legacy code. This function also identifies which interface IDs are used.

  You can use COM interop monitoring to determine which methods are being called frequently. If you find methods called many times, consider porting the object to avoid transitions. If re-writing is not an option, consider adding a new method that transfers data in bulk to reduce the number of transitions.
- **PInvoke interop monitoring:** Select this check box to count the number of times unmanaged or legacy code is called (broken out by DLLs and, if possible, by APIs). This helps you determine why your application is going into unmanaged or legacy code.

  `PInvoke` interop monitoring provides a count of the `PInvoke` calls that your application makes. The `PInvoke` interop monitoring report can be used to monitor managed to unmanaged transitions. Review the list to determine if excess calls are being made.

- **Interop reporting threshold:** Assuming ***`x`*** is the value specified in this field: when the number of times the application makes ***`call_A`*** is greater than or equal to ***`x`***, add ***`call_A`*** to the .NET Analysis results. This enables you to filter out calls that happen only a limited number of times. As you lower this threshold, more calls are included in your results.

  The interop reporting threshold allows you to exclude COM transitions from being reported in the COM interop and PInvoke interop monitoring reports. DevPartner error detection only reports transitions if the number of transitions is greater than or equal to the value specified.

Setting .NET Framework Call Reporting Properties

Tip: .NET Framework Call Reporting can generate a large amount of data, and cause system slowdowns. Enable .NET Framework Call Reporting only when necessary to debug and understand the framework, and even then select only the assemblies you need to check. Limiting the number of assemblies selected in the **All types** tree view decreases the size of the log file and improves performance.

Use .NET Framework Call Reporting to record calls to, and returns from, .NET interfaces. DevPartner error detection attempts to differentiate between “User Assemblies” and “System Assemblies” for .NET modules based on where the NLB file for an assembly is found.

To enable .NET Framework Call Reporting, and activate the list of .NET assemblies, select the **Enable .NET method call reporting** check box.

The **All Types Tree View** displays the .NET assemblies associated with the project. Click the plus (+) symbol next to the **All types** entry to expand the tree. The tree contains branches for both .NET User Assemblies and .NET System Assemblies. Check boxes next to each item enable you to select specific assemblies for .NET Framework call reporting.

Setting Resource Tracking Options

When you enable Resource Tracking, error detection:

- Monitors all application calls that allocate and free system resources other than memory
- Reports resources that have not been freed when an application ends

To enable Resource Tracking, and activate the list of resources, select the **Enable resource tracking** check box. When you select a check box in the associated list, error detection tracks the resources created by that DLL.

You can further refine resource tracking to a particular set of resources by selecting from one or more resource de-allocation APIs. For example, to exclude all registry related resources, clear the **RegCloseKey** check box under the **`advapi.dll`** resource.

Setting Modules and Files Options

Use the Modules and Files settings to specify the modules that make up the application.

Excluding modules, or components of modules, does not affect instrumentation. You can only limit instrumentation by using the Instrumentation Manager.

Tip: You may need to explicitly add a module to this list so that you can then exclude it from evaluation. DevPartner error detection automatically includes any modules that are not listed under Modules and Files. When you need to exclude a module that is not listed, you must first add the module, and then clear its check box to exclude it.

DevPartner error detection automatically evaluates all modules in your program. Use the Modules and Files settings to:

- Exclude modules from evaluation
- Exclude components within a module from evaluation
- Add modules you want to evaluate

Modules and Files includes the following settings:

- **Modules and Files List:** Shows the modules being checked.
  - To exclude an entire module from checking, clear the check box next to that module.
  - To expand a module and view its contents, click the plus symbol to the left of the module path.
  - To exclude specific items within a module, expand the module then clear the check box next to the item(s) to be excluded.

    After you clear a check box next to a module or an item within a module, it appears on the list but is not analyzed when you run error detection on your application.

    The check box next to a module name appears in yellow if one or more of its components has been cleared.

    Disabling all the modules in the Modules and Files settings does not prevent reporting of some error types. DevPartner error detection always reports memory overruns within any module, and other types of events originating from the **`MFCxxxx.dll`** libraries.
- **Show leaks and errors only if source code is available:** Select this check box to limit reported leaks and errors to those having source code available. When enabled, this option might reduce the number of leaks and errors reported. When disabled (the default condition), all leaks and errors are reported.
- **Add module:** Click to open the **Choose Module to Add** dialog box; use this dialog box to select and add the module.
- **Remove module:** Click to remove a selected module from the Modules and Files list. Active only when a module is selected. You cannot remove the main executable.
- **System directories:** Click to open the System Directories dialog box.

Setting System Directories Options

Use the System Directories dialog box to exclude entire folders that you do not need to check. For example, a folder may contain modules that generate errors that you have already dealt with. Excluding folders that you do not need to check can speed up error detection sessions.

DevPartner error detection reports all errors of undetermined origin and all errors that cause catastrophic failure of your application. Such errors are reported even though they might occur in modules within an excluded folder.

The following settings are available for the System Directories dialog box.

- **Add:** Opens the System Directory to Add dialog box. Use this dialog box to select a folder to add to the list of folders excluded from checking by error detection.
- **Remove:** Click to remove a selected system folder from the list.
- **OK:** Click to close the System Directories dialog box and save any changes you have made.
- **Cancel:** Click to close the System Directories dialog box and discard any changes you have made.

Directory Icon

The directory icon next to each path name in the System Directories list indicates two different possible conditions:

- **Single Directory:** Indicated by a single folder icon. Only the immediate contents of the selected folder are included.
- **Directory and All Sub-directories:** Indicated by a three-folder icon. The selected folder and all sub-folders are included.

To toggle between the two options, click the icon next to the folder name.

Note: In some instances you may find that you need to explicitly add critical third-party DLLs that are contained in an excluded folder. Explicitly adding these third-party DLLs might reveal problems you may not otherwise locate. To explicitly add a DLL, use the Modules and Files settings.

Setting Fonts and Colors Options

Fonts and Colors control the appearance of items that appear in the tabs of the error detection window. For example, you can increase the font size of the error data you view most frequently, or decrease font sizes to display more information in a tab.

Use the following controls to define the fonts and colors:

- **Show settings for:** Lists the different tabs that appear in the Results pane. Select the tab for which you are changing fonts and colors.
- **Use Defaults:** Click to discard all current settings and restore the original fonts and colors.
- **Displayable items:** Select an item from this list to change its font or color properties.
- **Font:** Select a font to use for the currently selected item under **Displayable items**.
- **Size:** Select a font size to use for the currently selected item under **Displayable items**.
- **Item foreground:** Shows the foreground color for the current selection in the **Displayable items** list. Select a foreground color from this drop-down menu or click **Custom** to the left of the menu to define a custom foreground color.

- **Item background:** Shows the background color for the current selection in the **Displayable items** list. Select a foreground color from this drop-down menu or click **Custom** to the left of the menu to define a custom background color.
- **Bold:** When selected, the text for the displayable item appears in a bold font.
- **Tab Size:** Use this control to specify the indent size for code displayed in the Source Code pane.

  This control is available only when the **Show settings for** selection is Source Pane and the **Displayable items** selection is **Main**.
- **Sample Text Box:** A text box at the bottom of the Fonts and Colors window shows how the current displayable item appears with the combination of fonts and colors selected.

Setting Configuration File Management Options

Use the Configuration File Management settings to manage configuration files. The title bar of the Program Settings dialog box displays the configuration file currently in use.

When changing a setting in the Program Settings dialog box, an asterisk appears after the configuration file name until you save the properties, reload the file, or load a different file. If you load a different file or reload the file without saving, changes to the current file are lost.

Use the following controls to define the configuration file functionality:

- **Configuration file name:** The full path and name of the configuration file.
- **Reload:** Loads the current configuration file again, discarding any changes. This returns you to the last saved version of the current configuration file.
- **Load:** Opens the Load From dialog box.
  - Select **Internal User Defaults** to load your user default settings.
  - If you select **Configuration File**, the Load Configuration File dialog box appears. Use this to select a different configuration file to load.
- **Save:** Saves all active changes in the currently loaded configuration file.
- **Save As:** Opens the Save Configuration File dialog box. Use this to save the current configuration settings under a different file name.
- **Reset:** Resets all the program property settings to the default factory settings.
- **Save Defaults:** Save current settings as user defaults. All new projects use these settings.
- **Delete Defaults:** Delete the user default configuration settings and revert to factory settings. All new projects use the factory settings.

Tracking Windows Messages and Event Logging

Windows is an event-driven environment in which much of your program executes in response to Windows messages and other events. DevPartner intercepts events as they occur, and logs them. You can use these logs to see a complete history of events that led to a problem.

DevPartner logs the following events:

- Windows messages.

  These events show how your program reacted to Windows messages.
- API calls and API returns along with argument information. These events define the order in which procedures are executed in your program.
- Output debug string messages from the program you are checking.
- Error messages.

Exporting Data to XML

DevPartner lets you export comprehensive session results data to XML, providing you with a simple way to port your results data into various report formats, email, an internal Web page, etc.

Exporting Data from within Visual Studio

Follow these steps when you are using Error Detection from within Visual Studio to export all data from the currently displayed session file to an XML file:

1. Open an Error Detection session file.
2. Select **File > Export DevPartner Data**.

   The Save As dialog box appears.
3. Choose the location for the exported data file.
4. Click **OK**.

Exporting Data from the Error Detection Standalone Application

Follow these steps when you are using the Error Detection standalone application to export all data from the currently displayed session file to an XML file:

1. Open an Error Detection session file.
2. Select **File > Export Data**.

   The Save As dialog box appears.
3. Choose the location for the exported data file.
4. Click **OK**.

Exporting Data from the Command Line

You can elect to generate an XML file from the session file data when you run error detection from the command line by passing the proper flags to `BC.exe` at execution time, or by calling `BC.exe` and specifying a pre-existing session file.

DevPartner error detection uses the **`DPSErrorDetection.xsd`** schema file located in the error detection installation folder when generating the XML output. Do not edit this file.

If DevPartner cannot export your session data to XML, it generates an error message describing the problem it encountered.

Running a Session and Exporting Data

When you specify an executable, error detection runs a session on the executable and then generates XML output from the results:

```
BC.exe [/B session.DPbcl] [/X[S|D] xmlfile.xml] target.exe [args]
```

The `S` and `D` flags used with `/X` allow you to export either Summary or Detail information to XML.

Note: When you specify an executable, you must still specify an corresponding session file using the `/B` flag.

Converting an Existing File

When you specify a session file only (**`session.DPbcl`**), error detection converts the specified session file to XML and saves the output:

```
BC.exe [/B session.DPbcl]  [/X[S|D] xmlfile.xml]
```

Running Error Detection from the Command Line

You can run DevPartner from a DOS command line, using **`bc.exe`** or **`bc.com`**.

Note: For legacy support of the 7.x versions, DevPartner error detection allows you to continue using `bc7.com` in your script files.

- **`bc.exe`** starts the UI for DevPartner standalone.
- **`bc.com`** is a small console program that spawns **`bc.exe`** and waits for it to complete.

The difference between **`bc.exe`** and **`bc.com`** is important for batch scripts. Invoking **`bc.exe`** directly starts DevPartner and continues on to the next command without waiting for **`bc.exe`** to complete. If the next step in the script is to check for a result, it is not available.

If you type only **`bc`**, the OS chooses **`bc.com`** instead of **`bc.exe`**.

For more information, refer to Using Error Detection from the Command Line in *DevPartner Advanced Error Detection Techniques*.

Command Line Options and Syntax

Brackets `[ ]` indicate that a command is optional.

```
BC.exe [/?]
```

```
BC.exe session.DPbcl
BC.exe [/B session.DPbcl] [/C configfile.DPbcc] [/M] [/NOLOGO]
[/X[S|D] xmlfile.xml] [/OUT errorfile.txt] [/S] [/W workingdir]
target.exe [args]
```

Table 2-4. Command Line options

Option	Action
/?	Display usage information
session.DPbcl	Open an existing session file
/B *session*.DPbcl	Run in batch mode and save the session file to a log file **`session.DPbcl`**
/C *configfile*.DPbcc	Use the **`configfile.DPbcc`** options
/M	Start **`bc.exe`** and minimize when running
/NOLOGO	Do not show the splash screen when loading **`bc.exe`**
/X *xmlfile*.xml	Generate XML output and save to the specified file. When you specify an executable, error detection runs a session on the executable and then generates XML output from the results. When you specify a session file only (**`session.DPbcl`**), error detection converts the specified session file to XML and saves the output. **Note:** When you specify an executable, you must still specify an corresponding session file using the /B switch.
/XS *xmlfile*.xml	The `/X` parameter used with the `S` modifier instructs error detection to only save Summary data to the xml file. Information about the running of the error detection session (Session data) is always exported.
/XD *xmlfile*.xml	The `/X` parameter used with the `D` modifier instructs error detection to only save Details data to the xml file. Information about the running of the Error Detection session (Session data) is always exported.
/OUT *errorfile*.txt	Output any error messages to a text file
/S	Run in silent mode. Do not open the Program Error Detected dialog box on errors.
/W *workingdirectory*	Set the target's working folder
target.exe [*args*]	The executable to launch and its arguments

You must specify the full path to your program executable if it is not located on the current path (the environment variable listing the locations the system searches in order to find an executable).

Running FinalCheck from the Command Line

You can also run FinalCheck from the command line. For more information, refer to the following topics in the *Checking a Program with FinalCheck* section of the online help.

- Running FinalCheck from the Command Line
- NMCL Options
- NMLINK Options

Monitoring Hooked Function Activity While Running an Application

DevPartner lets you monitor real-time function activity occuring while an application loads and runs under Error Detection or Performance Analysis.

Monitoring called function activity with the DevPartner Activity Monitor helps determine whether a started application is continuing to load or has stopped responding. It also provides insight into called function volume while critical parts of the application are exercised.

The DevPartner Activity Monitor automatically monitors application activity of any application started with Error Detection. The Activity Monitor icon appears in the system tray by default when active and displays notifications when activity monitoring starts and stops.

The Activity Monitor icon changes to reflect the monitoring state.

🟢	Application activity is currently monitored.
🚫	Monitor is inactive, no application is being monitored.

In either state, double-click the Activity Monitor icon in the system tray to access the Activity Monitor. The Activity Monitor graphs the activity of hooked function calls for the application currently run under error detection. Some performance activity can also be monitored while running an application under performance analysis.

Numbers along the left side of the Activity Monitor indicate the number of hooked function calls at specific points in time, while the bottom of the graph shows timing intervals. The file path and name of the application currently being monitored appear below the graph. The graph automatically adjusts the hooked function call number range to show activity at given time intervals.

See the *Error Detection online help* for information about customizing Activity Monitor behavior.

Submitting Data to Visual Studio Team System/Team Foundation Server

DevPartner Studio supports Microsoft Visual Studio Team System if the Team Explorer client is installed and a Team Foundation Server connection is available.

Visual Studio Team System and Team Foundation Server Support in DevPartner Error Detection

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected item. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected item. Access work item submission in the following Error Detection tabs:

- Errors tab — submit a selected error
- Memory leaks tab — submit a selected leak
- Modules tab — submit a selected instance
- Other leaks tab — submit a selected leak
- .NET Performance tab — submit a selected instance

When you submit a bug, DevPartner populates the Work Item form with data from the tab. For more information about DevPartner Studio integration with Visual Studio Team System, see "Visual Studio Team System Support" on page 20.

Chapter 3

Static Code Analysis

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with code review. The second section provides reference information for an in-depth understanding of some DevPartner Studio code review functions.

Refer to the DevPartner Studio online help for additional task-oriented information about code review.

What is Code Review?

DevPartner code review helps developers write best-practices compliant Visual Basic and Visual C# code in Visual Studio. DevPartner code review identifies programming and naming violations, analyzes method call structures, and tracks overall code complexity.

Note: The code review feature analyzes managed code only, and is not supported in the DevPartner for Visual C++ BoundsChecker Suite.

The DevPartner code review feature delivers the following functionality:

- **Static code analysis and review**

  DevPartner code review performs a comprehensive static code analysis of your source code in Visual Studio, and displays results in the DevPartner Code Review window.

- **Automated command-line batch processing**

  You can execute a batch review of your solution from the command line. You can run these automated batch reviews in conjunction with a nightly build. You can also save time by using an automated batch review on large applications while you perform other tasks.

- **Data export to XML**

  DevPartner code review allows you to export session results in XML format, providing you with a simple way to transform your results data into report formats, e-mail, an internal Web page, etc. You can export your data to XML from code review after running a session, from the command line, or as part of an automated batch process.

- **Rules management and customization**

  The Rule Manager lets you configure rules used by code review to enforce code compliance with the standards you set. You can also group rules into sets for use in a review session, and create your own custom rules.

Using Code Review Out of the Box

The following Ready, Set, Go procedure introduces you to using DevPartner code review.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject being described in the shaded box, read the additional text following the box.

DevPartner code review creates data files for each target application. You must ensure that you have write access to the folder containing the target executable before starting code review.

Analyzing an application with DevPartner Studio does not require elevated system privileges. The system privileges you use to create and debug your application are sufficient for DevPartner to analyze the application.

Ready: Deciding How You Want to Run the Review

DevPartner code review is very flexible, with several different configurations you should consider for any session.

The following procedure assumes:

- ◆ You are running a review of a Visual Basic or Visual C# single-developer solution.
- ◆ You are running code review in a supported release of Visual Studio.
- ◆ All of the projects in your solution compile without errors.
- ◆ All projects to be reviewed are set to output debug information.

Refer to “Code Review Supported Project Types” on page 274 for a comprehensive list of supported project types for code review.

- ◆ Deciding What Rules to Enforce — You can use a wide variety of code review rules to enforce industry best practices in your code. You can also create custom rules and rule sets using the Rule Manager if you have additional standards to enforce.
- ◆ Selecting the Naming Guidelines to Enforce — DevPartner code review can use built-in naming analyzers to ensure your code follows industry-accepted naming standards.
- ◆ Collecting Metrics Data — You can collect metrics data during your review, which displays code complexity results (complexity, bad fix probability, and understanding level), based on McCabe Metrics.
- ◆ Collecting Call Graph Data — You can collect call graph data (representing all potential inbound and outbound calls) during your review.
- ◆ Excluding Projects in Your Solution — DevPartner code review includes all projects in your solution by default. If you know there are projects in your solution that you do not want code review to analyze, you can exclude them.

Note: You must have all selected projects set to output debug information. If a selected project is not set to output debug information for any available build configurations, when code review runs you are warned about build errors, and that project is excluded from future sessions.

Set: Selecting Options and Settings

DevPartner code review is flexible and customizable. Use the General options page to customize code review. To access the General options page, select DevPartner > **Options** and then select **Code Review** from the Options tree view.

For this procedure, you can use the default DevPartner properties and options. No changes to the settings are required.

- Selecting a Rule Set — You can choose a rule set from the Rule Set list prior to running your review. The **Default** rule set includes all Medium and High priority rules supplied by code review, which enables you to enforce common best practices in the industry. Table 3-2 on page 67 provides a list of the standard rule sets that come with code review.

  You can use the Rule Manager (see “Using the Code Review Rule Manager” on page 94) to create custom rules and rule sets.

- Selecting a Naming Guideline — You can choose a naming guideline from the **Naming Analysis To Use** list. The default behavior is for code review to enforce naming guidelines modeled after the Microsoft .NET naming conventions. However, you can enforce the Hungarian Notation naming convention instead, or enforce none at all.
- Enabling or Disabling Collection of Metrics Data — Select the **Collect Metrics** check box to enable collection of McCabe Metrics data (see “Collecting McCabe Metrics” on page 68). Clear the check box to disable this functionality.
- Enabling or Disabling Collection of Call Graph Data — Select the **Collect Call Graph Data** check box to enable collection of static method call data. Clear the check box to disable this functionality.

  If you run your review with this function enabled, the **Call Graph** tab in the Results window displays a static graphical representation of the inbound and outbound call path corresponding to the method or property selected from the solution tree in the far left pane.

  Call paths are statically generated. This means the graph shows the potential method calls in the call path, rather than the dynamic method calls made during program execution.

- Excluding Unwanted Projects — A check box next to each project in the **Projects To Be Reviewed** text box controls whether that project is analyzed by code review. Clear the check box associated with any projects you do not want code review to analyze.

Go: Starting Your Code Review Session

The process of DevPartner analyzing your code is referred to as the session. When the review is completed, the session data is displayed in the Results window, and is saved to a file when you stop code review.

1 Open your solution in Visual Studio.

2 Select **DevPartner** > **Perform Code Review**.

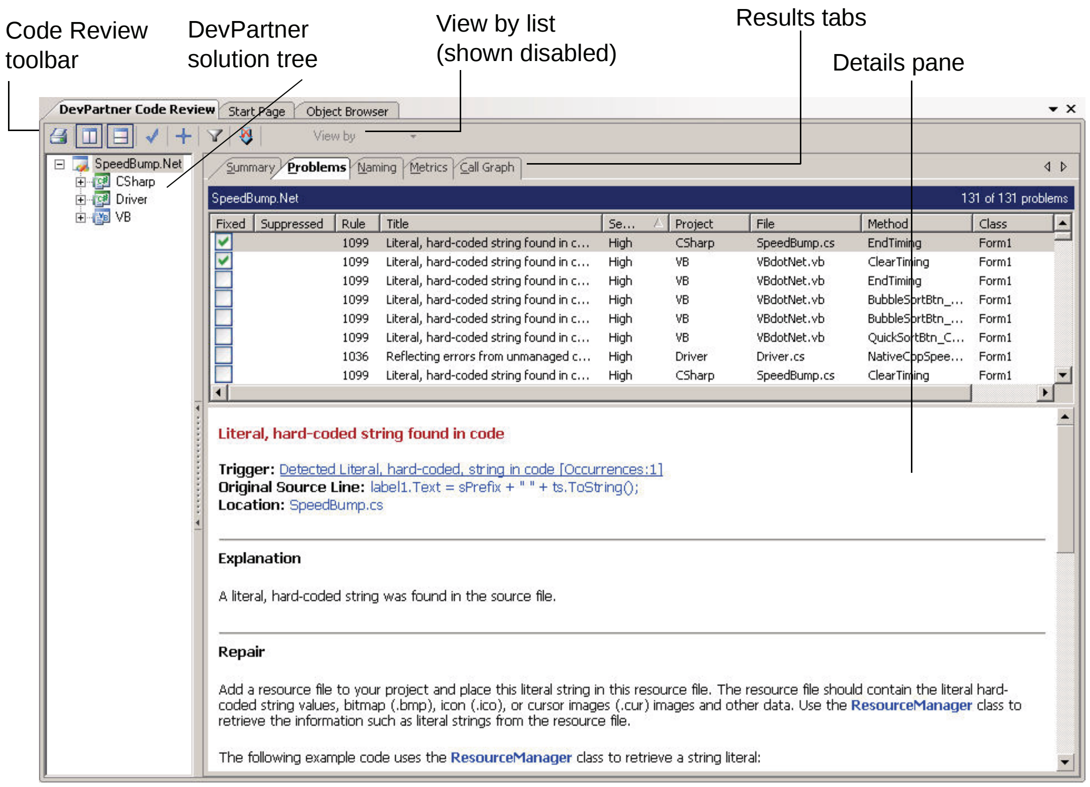
DevPartner performs a code review on all projects in the solution. The Results window opens, and a status bar on the Summary pane tracks the progress of the session.

You have completed running a basic code review session, and the data has been compiled in the Results window for you to analyze.

Analyzing the Results and Repairing Violations

The Results window is your focus once you have completed running a review of your solution. The session data is displayed in the Results window, and you use it to begin identifying, locating, and repairing violations.

1. Examine the **Problems** tab of the Results window to see code violations found during the review.
2. Ensure the **Severity** column is sorting the violations from highest to lowest priority (default behavior). Toggle the column between ascending and descending order, if required, by clicking the column heading.



SpeedBump.Net 131 of 131 problems

Fixed	Suppressed	Rule	Title	Se...	Project	File	Method	Class
✓		1099	Literal, hard-coded string found in c...	High	CSharp	SpeedBump.cs	EndTiming	Form1
✓		1099	Literal, hard-coded string found in c...	High	VB	VBdotNet.vb	ClearTiming	Form1
☐		1099	Literal, hard-coded string found in c...	High	VB	VBdotNet.vb	EndTiming	Form1
☐		1099	Literal, hard-coded string found in c...	High	VB	VBdotNet.vb	BubbleSortBtn_...	Form1
☐		1099	Literal, hard-coded string found in c...	High	VB	VBdotNet.vb	BubbleSortBtn_...	Form1
☐		1099	Literal, hard-coded string found in c...	High	VB	VBdotNet.vb	QuickSortBtn_C...	Form1
☐		1036	Reflecting errors from unmanaged c...	High	Driver	Driver.cs	NativeCppSpee...	Form1
☐		1099	Literal, hard-coded string found in c...	High	CSharp	SpeedBump.cs	ClearTiming	Form1

Literal, hard-coded string found in code

Trigger: Detected Literal, hard-coded, string in code [Occurrences:1]
Original Source Line: label1.Text = sPrefix + " " + ts.ToString();
Location: SpeedBump.cs

Explanation

A literal, hard-coded string was found in the source file.

Repair

Add a resource file to your project and place this literal string in this resource file. The resource file should contain the literal hard-coded string values, bitmap (.bmp), icon (.ico), or cursor images (.cur) images and other data. Use the **ResourceManager** class to retrieve the information such as literal strings from the resource file.

The following example code uses the **ResourceManager** class to retrieve a string literal:

Figure 3-1. Code Review Results window

Typically, you want to correct the most critical code violations first. The **Problems** tab is designed to sort the code violations in order of severity, allowing you to easily select the highest priority violations first.

There are several tabs available in the Results window that separate the session data into distinct categories.

- Select the **Summary** tab to examine a report summarizing the violations of various types that were discovered during the review (see "Viewing Summary Data" on page 71).

- Select the **Problems** tab to view the code violations discovered during the review. The default behavior of the **Problems** tab is to sort the list of violations from highest severity to lowest (see "Viewing Code Violations" on page 73).
- Select the **Naming** tab to view the naming violations discovered during the review. This list also provides suggestions for repair when applicable, and is empty if the review was configured to ignore naming (see "Viewing Naming Violations" on page 74).
- Select the **Metrics** tab to view code complexity results (Complexity, Bad Fix Probability, and Understanding Level), based on McCabe Metrics (see "Viewing Collected Metrics" on page 77).
- Select the **Call Graph** tab for a graphical representation of the method calls (see "Viewing Call Graph Data" on page 80).

Filtering Results

After running a code review session, the results can include a lot of data, making it difficult to focus on one area to repair. Use the Code Review solution tree to filter the results by selecting a project, file, or method. Filtering the data limits what is displayed, allowing you to focus on the results that are most important to you.

Analyzing Code Violations

By default, the **Problems** tab has focus in the Results window following a code review. The **Problems** tab displays the code violations found in the current solution. An associated Details pane (see Figure 3-1 on page 62) below the **Problems** tab provides more explanation, examples, references to MSDN and other sources explaining the problem, and suggested repairs (when available) for the selected code violation.

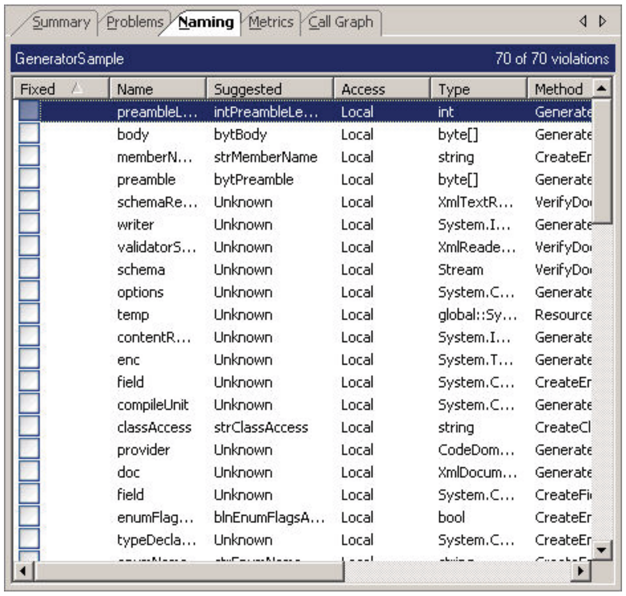
3. Select the first code violation listed on the **Problems** tab (highest priority). The Details pane shoes information about the selected code violation. The Trigger and Location headings tell you why a code violation occurs and where the violation is located.
4. Scroll down and examine the **Explanation**, code samples (if available), and suggested **Repair** for the code violation. Follow any external links to more explanations about the violation for additional information.
5. Double-click on the code violation listed on the **Problems** tab.
6. DevPartner opens a new window containing the Visual Studio editor and your source code, with focus placed at the line of code where the problem exists.Repair the code violation using the Visual Studio editor.
7. Return to the **Problems** tab in the code review Results window.
8. Select the **Fixed** check box to indicate that you corrected the violation.
9. Select the next code violation in the **Problems** tab, repeating steps 5 through 8 until you feel you addressed enough code violations for this session.

Note: DevPartner code review attempts to keep track of line numbering changes, and maintain synchronization between violations and source code. After enough modifications, the results can lose synchronization with the source code, and line numbers may change. You can re-run a code review session after modifying the source code, and continue repairing violations with the new **Problems** tab list.

You have now resolved code violations in your solution using code review.

Analyzing Naming Violations

The **Naming** tab lists naming violations that code review finds during a review. **Naming** tab content varies based on the naming analysis type selected on the General options page prior to the review. An associated Details pane (like the one associated with the **Problems** tab in Figure 3-1 on page 62) below the **Naming** tab provides more explanation, resources, and suggested repairs (when available) for the selected naming violation.



GeneratorSample 70 of 70 violations

Fixed	Name	Suggested	Access	Type	Method
☐	preambleL…	intPreambleLe…	Local	int	Generate
☐	body	bytBody	Local	byte[]	Generate
☐	memberN…	strMemberName	Local	string	CreateEr
☐	preamble	bytPreamble	Local	byte[]	Generate
☐	schemaRe…	Unknown	Local	XmlTextR…	VerifyDo
☐	writer	Unknown	Local	System.I…	Generate
☐	validatorS…	Unknown	Local	XmlReade…	VerifyDo
☐	schema	Unknown	Local	Stream	VerifyDo
☐	options	Unknown	Local	System.C…	Generate
☐	temp	Unknown	Local	global::Sy…	Resource
☐	contentR…	Unknown	Local	System.I…	Generate
☐	enc	Unknown	Local	System.T…	Generate
☐	field	Unknown	Local	System.C…	CreateEr
☐	compileUnit	Unknown	Local	System.C…	Generate
☐	classAccess	strClassAccess	Local	string	CreateCl
☐	provider	Unknown	Local	CodeDom…	Generate
☐	doc	Unknown	Local	XmlDocum…	Generate
☐	field	Unknown	Local	System.C…	CreateFi
☐	enumFlag…	blnEnumFlagsA…	Local	bool	CreateEr
☐	typeDecla…	Unknown	Local	System.C…	CreateEr

Figure 3-2. Naming tab

1. Select the **Naming** tab to identify the naming violations discovered during the session (see Figure 3-2). All naming violations found during the review are listed in this tab. When available, a suggestion for proper naming appears beside the violation.
2. Select the first naming violation on the **Naming** tab.

   The Details pane displays data about the selected naming violation. If selecting Hungarian naming, the Details pane is not available.
3. Examine the detailed explanation and/or suggestion for proper naming (when available). Follow any external links if you want more information about the violation.
4. Double-click on the naming violation in the **Naming** tab list to go to the source.
5. Repair the naming violation.
6. Return to the **Naming** tab in the code review Results window.
7. Select the **Fixed** check box to indicate you corrected the violation.
8. Select the next naming violation in the **Naming** tab, repeating steps 11 through 16 until you feel you addressed enough naming violations for this session.

You have now resolved naming violations in your solution using code review.

Saving Session Files

Saving your session file allows you to refer back to these results. You might want to open a saved session file for several reasons:

- To export the session data to XML at a later date (see "Exporting Data to XML" on page 88).
- To continue fixing the violations discovered in this session later.

Code review saves the session file by default when you exit, unless you clear the **Always save review results** setting under General options (see "Configuring General Options" on page 66).

1. With focus in the Results window, select **File > Save Code Review Session As**.
2. Enter a name for the session file and click **Save**.

   By default, code review saves the session file as **`SolutionName.dpmdb`** in the same location as your solution.

DevPartner saves session files as part of the active solution. They appear in the DevPartner Studio virtual folder in Solution Explorer. DevPartner code review session files take the **`.dpmdb`** extension.

By default, DevPartner physically saves the session files in your project's output folder. DevPartner automatically increments the file name based on the contents of the default folder (for example, **`MyApp.dpmdb`**, **`MyApp1.dpmdb`**, and so on). If you save session files to a location other than the default folder, you must manage the file naming.

For projects that do not have an output folder, such as a Visual Studio 2005 Web site project, DevPartner physically saves the files to the project folder.

Session files generated from the command line are not automatically added to the project's solution. Manually add externally generated session files to an open solution in Visual Studio.

This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a code review session, continue reading the rest of this chapter for more information.

Setting Options

Use the many available options to customize code review behavior. Your specific settings are preserved in a preferences database on your system. DevPartner provides three option pages to modify code review options:

- General Options
- Naming Guidelines Options
- Suppressed Rules

Configuring General Options

The General options page contains code review settings that you can modify prior to a code review. To access the General options, from the **DevPartner** menu select **Options**, then select **DevPartner > Code Review > General** from the tree view.

Selecting Projects To Review

You can choose some or all of the projects in a solution from the **Projects to be reviewed** list. The contents are empty if:

- You did not load a C# or Visual Basic solution in Visual Studio.
- You loaded a solution containing only C++ projects.

The **Projects to be reviewed** list contains the following information.

Table 3-1. Projects to be reviewed list

Item	Description
Check Box	The corresponding project is reviewed when checked
Project	The name of the project
Language	Visual Studio language associated with the project: • Visual Basic • C# • Web Site
Path	The path and name of the listed project

If you never made selections to the list of projects, DevPartner reviews all projects in the current solution by default. Once you edit the list of projects, DevPartner saves the state of included or excluded projects for the next time you work with this solution. The following caveats apply:

◆ You must select at least one project in order to review a solution. Otherwise, DevPartner does not let the code review proceed.

◆ You must have all selected projects set to output debug information. If a selected project is not set to output debug information for any available build configurations, build errors occur and that project is excluded from future sessions.

◆ If you select one or more projects that no longer exist in the solution, DevPartner reviews the remaining projects.

◆ If you inadvertently delete all the projects in a solution that you later attempt to review, DevPartner indicates that the selected projects no longer exist in the solution and suggests that you make appropriate changes on the General options page.

◆ You cannot select individual files, classes, or methods within a given project.

Selecting Rule Sets

You can select a rule set from the **Rule set** list to apply to a code review. The **Rule set** list contains all DevPartner-supplied and user-configured rule sets. The selected rule set is preserved and used each time you run a session on the current solution.

Note: Make sure that you select a valid rule set that contains rules and already exists in the rules database. Attempting to use a rule set that has been removed via the Rule Manager, or is empty could invalidate the results.

You can create and customize rules (along with their associated triggers, which cause the rules to fire when they are violated), and create and manage rule sets, using the Rule Manager (see “Using the Code Review Rule Manager” on page 94).

Table 3-2. Standard Rule Sets

Rule Set Name	Description
All Rules	Provides a master rule set which, out of the box: • Contains all DevPartner rules in the rules database • Contains any user-configured rules in the rules database • Ensures a comprehensive code review
Date Formatting	Checks for proper formatting and use of date values, in particular 2-digit year formatted dates
Default	Contains high and medium priority rules
Design Time Properties	Checks for design time properties and property values of forms and controls to assist with good user interface design
Internationalization	Assists with localization, string handling, and comparison for the international market

Table 3-2. Standard Rule Sets

Rule Set Name	Description
Logic	Checks for proper program logic, good .NET Framework programming practices, error handling, type checking, and garbage collection
Performance	Checks for code that negatively impacts performance
Naming Guidelines	Searches for .NET Framework naming discrepancies that involve two or more identifiers in the source code
Web Applications	Checks for good ASP.NET development, HTML tag use, validation, performance, caching, and state

Selecting a Type of Naming Analysis

Note: You should choose a naming analysis type. Otherwise, DevPartner bypasses critical analysis functions. Only choose **None** to temporarily ignore naming anomalies while concentrating on other programming problems.

Use the **Naming analysis to use** list to choose the type of naming analysis to apply to a review. Your choices include:

- **Naming Guidelines (default)**: Patterned after the Visual Studio .NET Framework naming guidelines.

  You must also set options on the Naming Guidelines options page to ensure a more precise review (see “Setting Naming Guidelines Options” on page 69).

- **Hungarian**: Patterned after the Hungarian Notation naming convention (see “Understanding the Hungarian Naming Analyzer” on page 93).

  You must also choose a valid Hungarian name set. The name set choice does not apply to the Naming Guidelines naming analysis.

- **None**: DevPartner bypasses naming analysis, and the **Naming** tab is empty following the code review.

Selecting Name Sets

If you perform a Hungarian naming analysis on your source code, make sure that you also choose a valid Hungarian name set (by default, the **Default** name set is associated with the default DevPartner-supplied rule sets).

You can create and manage name sets using the Rule Manager (see “Using the Code Review Rule Manager” on page 94).

Collecting McCabe Metrics

When you select **Collect metrics**, code review collects data that displays code complexity statistics, including complexity, bad fix probability, and understanding level. These metrics follow the industry-standard McCabe Metrics (see “Understanding McCabe Metrics” on page 78). The Metrics tab displays an aggregate of all items pertaining to the node selected on the Code Review solution tree.

Collecting Call Graph Data

When you select the **Collect call graph data** check box, code review collects information about all potential inbound and outbound calls to methods or properties, and displays a graphical representation of the results on the **Call Graph** tab. Individual nodes in the call graph represent the inbound and/or outbound call path for the selected method or property. The call graph shows a static representation of the potential method calls in the call path, rather than the dynamic calls made during program execution.

Generating Batch Files

When you select **Always generate a batch file**, DevPartner generates a batch file during the next interactive code review performed in Visual Studio. You can use this batch file to run a batch review from the command line on the same solution.

Note: If you use the `/r` option when running reviews in a batch file or from the command line, you should turn off **Always generate a batch file**, or backup and rename your batch file. Otherwise, your batch file is overwritten. See "Using the Command Line Interface" on page 85

Saving Review Results

When you select the **Always save review results** check box, DevPartner saves the session file as **`SolutionName.dpmdb`** in the same location as your solution following a code review. DevPartner displays the saved session file the Visual Studio Solution Explorer.

Prompting for Session File Name

When you select the **Prompt for session file name** check box, DevPartner prompts you to specify a location and name for the session file before it begins the review.

Setting Naming Guidelines Options

The Naming Guidelines options page includes choices that ensure a more precise review. To access the Naming Guidelines options, from the DevPartner menu select **Options**, then select **DevPartner** > **Code Review** > **Naming Guidelines** from the tree view.

Selections on this page are disabled until you select the Naming Guidelines naming analyzer from the **Naming analysis to use** list on the General options page.

Choosing Identifiers to Analyze

In the **What to analyze** section, select the type of identifiers to include in the analysis:

- **All public or protected identifiers (default)**: DevPartner code review examines public or protected identifiers and internal protected identifiers. However, this option excludes local and private identifiers.

- **All identifiers regardless of access or scope**: DevPartner code review examines all identifiers, regardless of access or scope.

Choosing a Dictionary

From the **Choose dictionary** list, select the dictionary database to apply to the naming analysis. DevPartner searches for naming violations based on the selected dictionary. **American English** is the default dictionary.

Choosing the Scope of Naming Analysis

In the **Include naming analysis for** list, select the corresponding check boxes for one or more identifiers for DevPartner to analyze. By default, all identifiers are selected.

Checking the **Variables** check box might affect other variable-specific selections on this options page (such as **What to analyze**).

Selecting Camel Case or Pascal Case

Select your capitalization preference for DevPartner to use when validating named Variables. The two options are Camel case and Pascal case. Camel case refers to a variable where the initial word is lower-case but the secondary word is capitalized, such as `integerBonus`. Pascal case refers to a variable where each word in the name is capitalized, such as `IntegerBonus`.

This option is disabled if you did not already select **Variables** from the **Include naming analysis for** list. It is also unavailable if you have selected **All public or protected identifiers**.

Selecting Namespace Options

If you checked the **Namespaces** check box in the **Include naming analysis for** field, you can specify additional namespace options.

- **Company name**: Enter a string for your company's name.
- **Technology name**: Enter a string for your company's technology.

DevPartner code review verifies namespaces for appropriate use of capitalization, complete words, presence of reserved words, use of numbers, etc. DevPartner code review also verifies that each namespace follows the recommended namespace syntax: `CompanyName.TechnologyName[.Feature][.Design]`. When you provide a company name and/or technology name, code review checks that the namespace is constructed using these entries.

Managing Suppressed Rules

The Suppressed Rules options page contains a list of rules that have been suppressed in the **Problems** tab (see Figure 3-4). Suppressed rules may be temporarily unsuppressed by selecting the check box next to the suppressed rule in the Suppressed Rules options page. To access the Suppressed Rules options, from the DevPartner menu select **Options**, then select **DevPartner > Code Review > Suppressed Rules** from the tree view.

Suppressing Rules

Suppressing a rule tells code review not to fire that rule in future sessions. Suppressing a rule is very different from filtering a code violation:

- When you suppress a rule, it is never fired, no data is collected, and nothing is preserved in the session file.
- When you filter code violations, the underlying rules still fire, the data is collected and saved in the session file, but is not displayed.

You can save rule suppressions locally in an individual solution or universally across all solutions. You must first perform a code review before you can select a rule to suppress.

1. Click the **Problems** tab on the Results window.

   The **Problems** tab lists the code violations.

2. Select a code violation to suppress its underlying rule.

3. Access the Suppress Rule dialog box in one of two ways:

   - Click the **Suppress Rule** toolbar button.
   - Right-click on the highlighted rule line, and select **Suppress Rule** from the context menu.

4. Choose the scope where you want to suppress the selected rule:

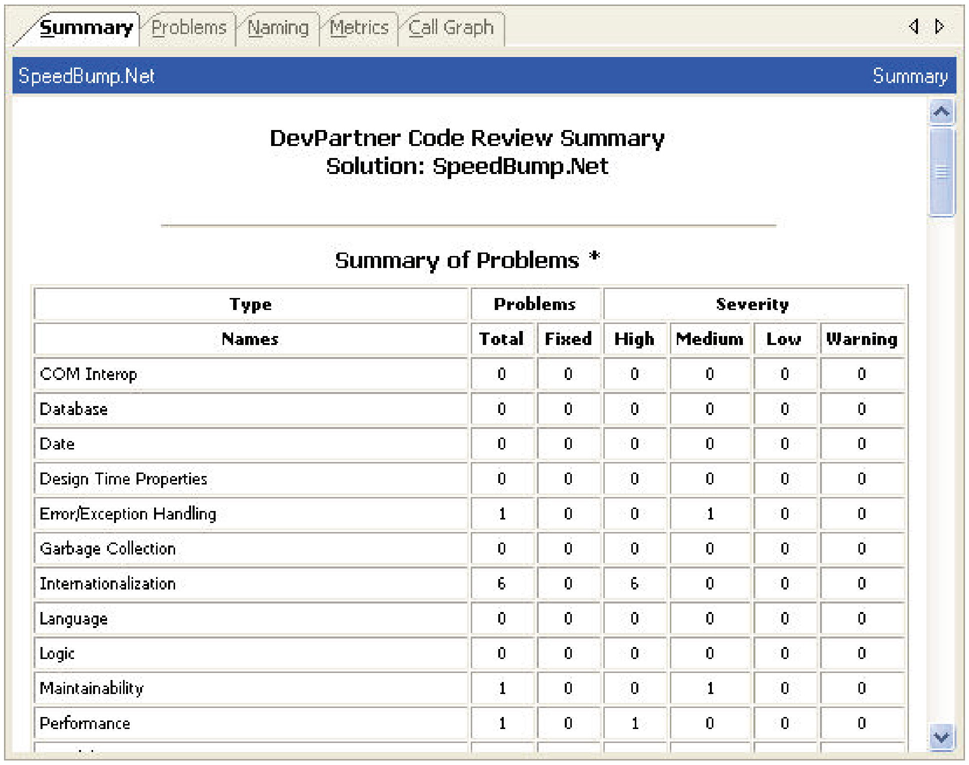
   - **Suppress this rule everywhere in this solution**: Affects future reviews of the current solution
   - **Universally across all solutions**: Universal effect on future reviews of all solutions

   When you select universal suppression, the preference database clears any solution-based suppressions for the rule and applies this universal setting across all solutions.

If you attempt to suppress a rule in the current solution and code review determines that the rule has already been suppressed in other solutions, the Suppress Rule dialog box prompts you to apply universal suppression instead. You still can choose to suppress it only in the current solution.

Viewing Summary Data

The **Summary** tab consolidates summarized results data in a single location, while details about each aspect of the session are displayed on the other corresponding tabs. Some items on the **Summary** tab are dynamic. As items on the **Problems** or **Naming** tabs are marked as *Fixed***,** the **Summary** tab dynamically reflects the update. If the review included unusual exceptions (such as running a review with an empty rule set), the **Summary** tab reflects that message in the header section. Scroll down the **Summary** tab to view each summary table.



DevPartner Code Review Summary
Solution: SpeedBump.Net

Summary of Problems *

Type	Problems		Severity			
Names	Total	Fixed	High	Medium	Low	Warning
COM Interop	0	0	0	0	0	0
Database	0	0	0	0	0	0
Date	0	0	0	0	0	0
Design Time Properties	0	0	0	0	0	0
Error/Exception Handling	1	0	0	1	0	0
Garbage Collection	0	0	0	0	0	0
Internationalization	6	0	6	0	0	0
Language	0	0	0	0	0	0
Logic	0	0	0	0	0	0
Maintainability	1	0	0	1	0	0
Performance	1	0	1	0	0	0

Figure 3-3. Summary tab

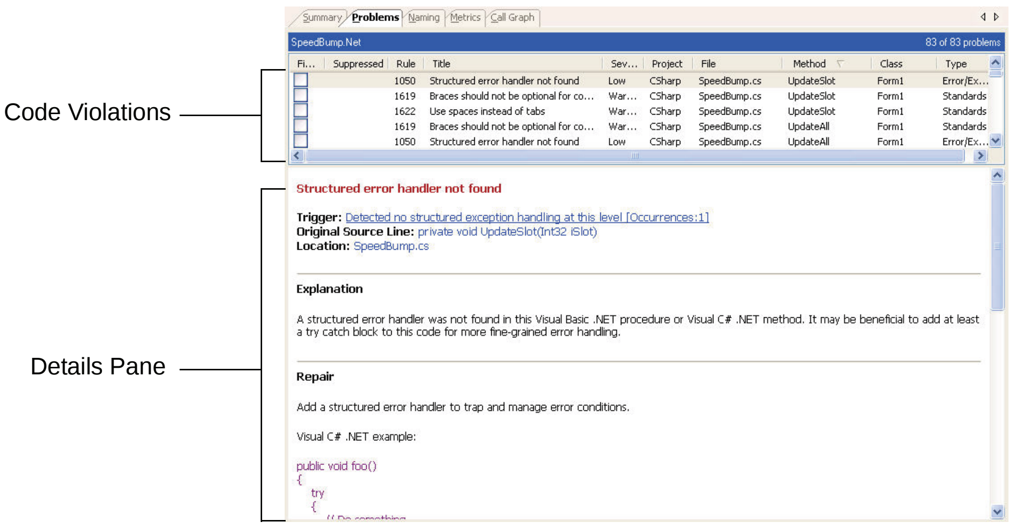
- **The Summary of Problems** table lists the categories of rules that were assessed in your review. It indicates the number of violations discovered and the number you have marked as fixed. It then breaks down the total number of violations by severity category.
- **Summary of Naming Guidelines** lists the categories that you originally selected on the Naming Guidelines options page to be included in the review. The table displays a summary of the naming identifiers selected on the Naming Guidelines options page, and indicates the number of violations found.

  This table only appears on the **Summary** tab if you selected **Naming Guidelines** on the General options page prior to the review. This table is not available for the Hungarian naming analyzer.
- **Summary of Call Graph Data** summarizes information about the call graph analysis captured during the review, including the total number of methods and properties analyzed, and the number that appear to be uncalled.

  This table appears on the **Summary** tab only if **Collect Call Graph Data** on the General options page is selected prior to review.
- **Summary of Counts** includes individual statistics gathered about the code review session itself, including how long it took to run, the number of lines in the solution, the number of comparisons made, etc.
- **Review Settings** lists configuration and review-related data. This information is useful for record keeping and troubleshooting.
- **Project List** provides information for each project in the solution, including whether each project had compile errors, or was successfully reviewed.

Viewing Code Violations

By default, the **Problems** tab has focus in the Results window following a code review. The **Problems** tab displays code violations found in the current solution. A Details pane below the list of code violations provides further explanation, examples, and possible repairs when you select a specific violation.



SpeedBump.Net 83 of 83 problems

Fi...	Suppressed	Rule	Title	Sev...	Project	File	Method	Class	Type
☐		1050	Structured error handler not found	Low	CSharp	SpeedBump.cs	UpdateSlot	Form1	Error/Ex...
☐		1619	Braces should not be optional for co...	War...	CSharp	SpeedBump.cs	UpdateSlot	Form1	Standards
☐		1622	Use spaces instead of tabs	War...	CSharp	SpeedBump.cs	UpdateSlot	Form1	Standards
☐		1619	Braces should not be optional for co...	War...	CSharp	SpeedBump.cs	UpdateAll	Form1	Standards
☐		1050	Structured error handler not found	Low	CSharp	SpeedBump.cs	UpdateAll	Form1	Error/Ex...

Structured error handler not found

Trigger: Detected no structured exception handling at this level [Occurrences:1]
Original Source Line: private void UpdateSlot(Int32 iSlot)
Location: SpeedBump.cs

Explanation

A structured error handler was not found in this Visual Basic .NET procedure or Visual C# .NET method. It may be beneficial to add at least a try catch block to this code for more fine-grained error handling.

Repair

Add a structured error handler to trap and manage error conditions.

Visual C# .NET example:

```
public void foo()
{
   try
   {
```

Figure 3-4. Problems tab and Details pane

Understanding the Problems Tab

The following table describes the information provided on the **Problems** tab.

Table 3-3. Contents of Problems tab

Column	Description
Fixed	Status of the code violation The checkbox is checked when fixed
Suppressed	Status of the rule suppression Suppressed, or blank for not suppressed
Rule	Number assigned to that code violation
Title	Title of the rule
Severity	Severity level (High, Medium, Low, Warning)
Project	Project where the violation exists
File	File where the violation exists
Method	Method where the violation exists
Class	Class of the fired rule
Type	Rule type

Details Pane

When you select a code violation on the **Problems** tab, more detailed information appears in the Details pane (see Figure 3-4 on page 73). The contents are generated from the rules stored in the code review rules database (system-supplied and user-configured). The following table lists the information provided in the Details pane.

Table 3-4. Contents of Details pane

Heading	Description
Rule title (shown in red)	Title of the rule
Trigger	Name of the trigger; appears as a hyperlink to the original source line (see “Configuring Triggers” on page 97)
Original Source Line	Line of code that caused the rule to fire
Location	Origin of the code violation
Explanation	Code violation description
Repair	Recommendation to fix the problem
Notes	Additional comments, such as external links to Microsoft MSDN knowledge base articles

Tip: Each code violation can include additional hyperlinks for Trigger, Original Source Line, and Location.

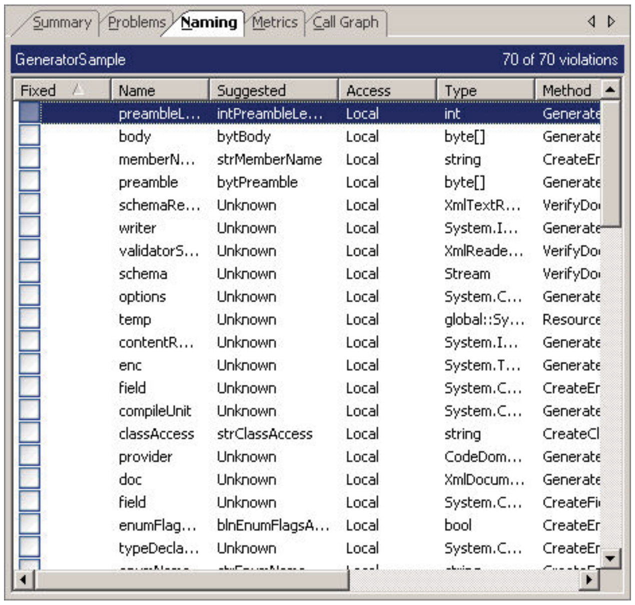
Viewing Naming Violations

The **Naming** tab lists naming violations that code review finds during a review. The appearance of the **Naming** tab varies depending on the type of naming analysis you selected on the General options page prior to the review. See “Understanding Naming Analysis” on page 90 for more information about each naming analyzer.

The **Naming** tab displays results from one or the other, but not from both naming analyzers. If **None** was selected, the **Naming** tab is empty following a code review.

Analyzing Hungarian Results

Figure 3-5 shows how the **Naming** tab appears when the Hungarian naming analyzer is selected on the General options page.



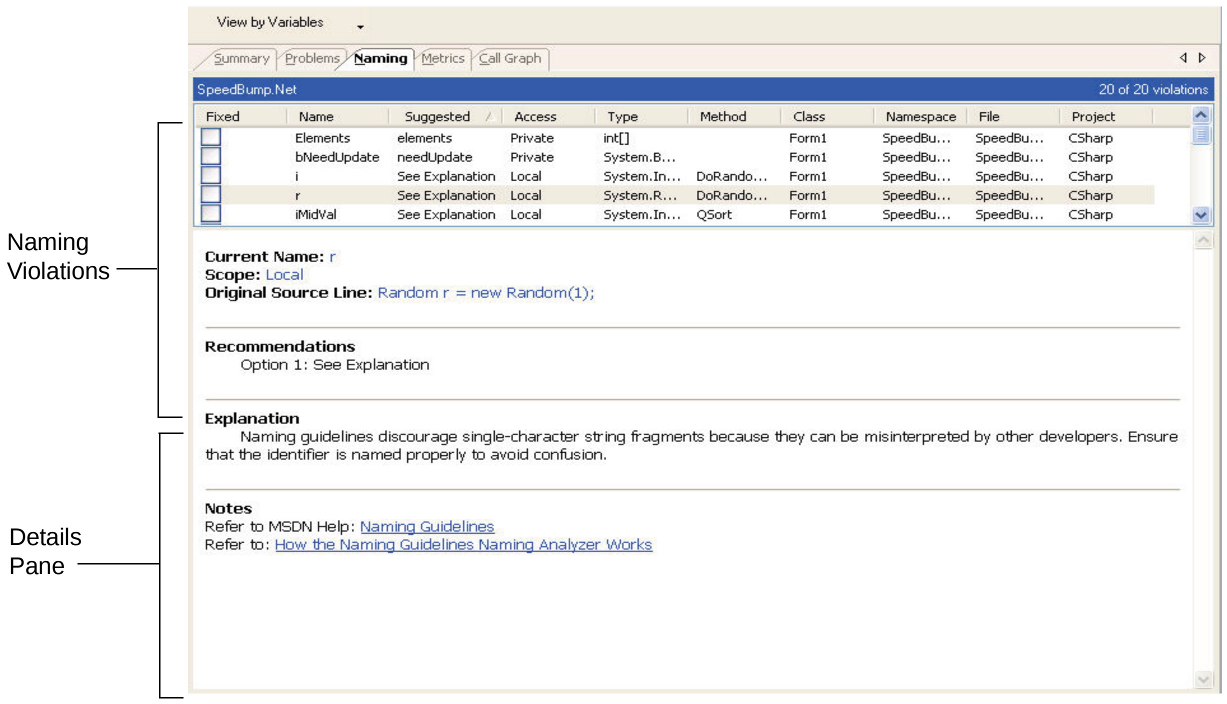
GeneratorSample — 70 of 70 violations

Fixed	Name	Suggested	Access	Type	Method
☐	preambleL...	intPreambleLe...	Local	int	Generate
☐	body	bytBody	Local	byte[]	Generate
☐	memberN...	strMemberName	Local	string	CreateEr
☐	preamble	bytPreamble	Local	byte[]	Generate
☐	schemaRe...	Unknown	Local	XmlTextR...	VerifyDo
☐	writer	Unknown	Local	System.I...	Generate
☐	validatorS...	Unknown	Local	XmlReade...	VerifyDo
☐	schema	Unknown	Local	Stream	VerifyDo
☐	options	Unknown	Local	System.C...	Generate
☐	temp	Unknown	Local	global::Sy...	Resource
☐	contentR...	Unknown	Local	System.I...	Generate
☐	enc	Unknown	Local	System.T...	Generate
☐	field	Unknown	Local	System.C...	CreateEr
☐	compileUnit	Unknown	Local	System.C...	Generate
☐	classAccess	strClassAccess	Local	string	CreateCl
☐	provider	Unknown	Local	CodeDom...	Generate
☐	doc	Unknown	Local	XmlDocum...	Generate
☐	field	Unknown	Local	System.C...	CreateFi
☐	enumFlag...	blnEnumFlagsA...	Local	bool	CreateEr
☐	typeDecla...	Unknown	Local	System.C...	CreateEr

Figure 3-5. Naming Tab for Hungarian Naming Analysis

Analyzing Naming Guidelines Results

Figure 3-6 shows a two-panel representation of the naming results when the Naming Guidelines naming analyzer is selected on the General options page. Notice the **Naming** tab in the upper panel and the Naming Details pane in the lower panel. Naming Guidelines analysis also enables the **View by** list above the **Naming** tab.



SpeedBump.Net — 20 of 20 violations

Fixed	Name	Suggested	Access	Type	Method	Class	Namespace	File	Project
☐	Elements	elements	Private	int[]		Form1	SpeedBu...	SpeedBu...	CSharp
☐	bNeedUpdate	needUpdate	Private	System.B...		Form1	SpeedBu...	SpeedBu...	CSharp
☐	i	See Explanation	Local	System.In...	DoRando...	Form1	SpeedBu...	SpeedBu...	CSharp
☐	r	See Explanation	Local	System.R...	DoRando...	Form1	SpeedBu...	SpeedBu...	CSharp
☐	iMidVal	See Explanation	Local	System.In...	QSort	Form1	SpeedBu...	SpeedBu...	CSharp

Current Name: r
Scope: Local
Original Source Line: Random r = new Random(1);

Recommendations
Option 1: See Explanation

Explanation
Naming guidelines discourage single-character string fragments because they can be misinterpreted by other developers. Ensure that the identifier is named properly to avoid confusion.

Notes
Refer to MSDN Help: Naming Guidelines
Refer to: How the Naming Guidelines Naming Analyzer Works

Figure 3-6. Naming tab for Naming Guidelines Naming Analysis

The following table lists the information provided on the **Naming** tab, regardless of the naming analyzer selected.

Table 3-5. Contents of Naming tab

Column	Description
Fixed	Status of the naming violation Select this check box when a violation is fixed.
Name	User-defined name for the data type
Suggested	Suggested names vary, depending on which naming analyzer is selected (see “Understanding Naming Analysis” on page 90) • If code review cannot suggest a name based on Hungarian naming conventions, *Unknown* appears in this column. • If code review cannot suggest a name based on Naming Guidelines, asterisks appear in this column. An explanation also appears on the Naming Details pane (Figure 3-6 on page 76).
Access	Category of access within the current solution
Type	Type of identifier
Method	Method where the data type is declared
Class	Class where the data type is declared
Namespace	Namespace where the data type is declared
File	File where the data type is declared
Project	Project where the data type is declared

Understanding the Naming Details Pane

If you selected Naming Guidelines and made additional choices on the Naming Guidelines options page, a Details pane appears below the **Naming** tab, providing additional details about the selected naming violation.

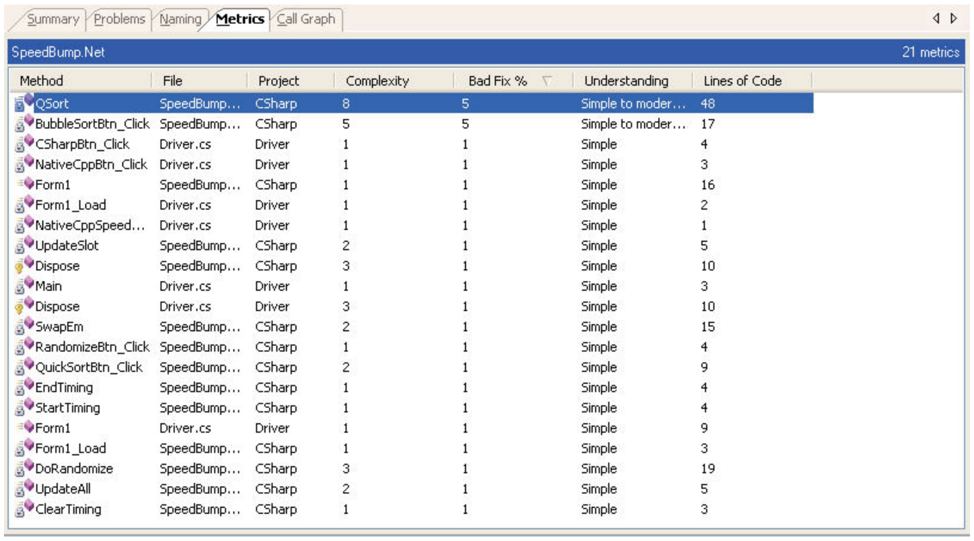
The Details pane is only available for the Naming Guidelines naming analyzer.

Table 3-6. Contents of Naming Details pane

Item	Description
Current name	Corresponds to the item selected in the upper panel
Scope	Indicates the scope of the identifier
Original Source Line	Displays the source line that pertains to the selected naming violation in the upper panel
Recommendations	Suggests one or more suitable names, based on the Naming Guidelines naming analyzer (see "Understanding the Naming Guidelines Naming Analyzer" on page 91)
Explanation	Provides an explanation for why this violation was flagged as a problem. If code review cannot suggest a better name, an explanation appears in this pane. DevPartner code review also shows a series of asterisks in the **Suggested** column of the upper panel of the **Naming** tab.
Notes	Optionally includes a hyperlink to the Naming Guidelines knowledge base in the .NET Framework General Reference

Viewing Collected Metrics

The **Metrics tab** (Figure 3-7) displays code complexity results (complexity, bad fix probability, and understanding level), based on McCabe Metrics (see "Understanding McCabe Metrics" on page 78).



SpeedBump.Net 21 metrics

Method	File	Project	Complexity	Bad Fix %	Understanding	Lines of Code
QSort	SpeedBump...	CSharp	8	5	Simple to moder...	48
BubbleSortBtn_Click	SpeedBump...	CSharp	5	5	Simple to moder...	17
CSharpBtn_Click	Driver.cs	Driver	1	1	Simple	4
NativeCppBtn_Click	Driver.cs	Driver	1	1	Simple	3
Form1	SpeedBump...	CSharp	1	1	Simple	16
Form1_Load	Driver.cs	Driver	1	1	Simple	2
NativeCppSpeed...	Driver.cs	Driver	1	1	Simple	1
UpdateSlot	SpeedBump...	CSharp	2	1	Simple	5
Dispose	SpeedBump...	CSharp	3	1	Simple	10
Main	Driver.cs	Driver	1	1	Simple	3
Dispose	Driver.cs	Driver	3	1	Simple	10
SwapEm	SpeedBump...	CSharp	2	1	Simple	15
RandomizeBtn_Click	SpeedBump...	CSharp	1	1	Simple	4
QuickSortBtn_Click	SpeedBump...	CSharp	2	1	Simple	9
EndTiming	SpeedBump...	CSharp	1	1	Simple	4
StartTiming	SpeedBump...	CSharp	1	1	Simple	4
Form1	Driver.cs	Driver	1	1	Simple	9
Form1_Load	SpeedBump...	CSharp	1	1	Simple	3
DoRandomize	SpeedBump...	CSharp	3	1	Simple	19
UpdateAll	SpeedBump...	CSharp	2	1	Simple	5
ClearTiming	SpeedBump...	CSharp	1	1	Simple	3

Figure 3-7. Metrics tab

The **Metrics** tab only displays data if you selected the **Collect metrics** check box on the **General** options page prior to the review. Table 3-7 lists the information provided on the **Metrics** tab.

Table 3-7. Contents of Metrics tab

Heading	Description
Method	Method name where the code complexity issue originated
File	File name where the issue originated
Project	Project where the issue originated
Complexity	Degree of complexity regarding a particular component; this metric is related to McCabe Cyclomatic Complexity
Bad Fix %	Likelihood that a new bug occurs in the code when trying to fix a known bug
Understanding	How straightforward the code logic is to decipher and maintain
Lines of Code	Total lines of code within the selected component; breakdown of individual line counts appear on the **Summary** tab

Understanding McCabe Metrics

When you collect McCabe Metrics, the **Metrics** tab displays code complexity statistics, including: complexity, bad fix probability, and understanding level. These metrics follow the industry-standard McCabe Metrics. The **Metrics** tab displays an aggregate of all items pertaining to the node selected on the Code Review Solution Tree.

Complexity

Complexity (also called Cyclomatic Complexity or McCabe's complexity) represents an industry standard established as part of McCabe Metrics. Complexity is a broad measure of soundness and confidence for a program. This measure provides a single ordinal number that can be compared to the complexity of other programs. It is often used in concert with other software metrics. As one of the more widely accepted software metrics, it is intended to be independent of language and language format. The complexity number denotes a stronger measure of a program's structural complexity than counting the number of lines of code.

Complexity measures the degree of complexity in a module's decision structures by measuring the number of linearly-independent paths through a program module. Each component is analyzed individually and then all possible decision points are calculated, e.g., If-Then-Else and Select Case statements. With Select Case, each case is a separate decision point.

McCabe Metrics defines Cyclomatic Complexity for each module as

```
e - n + 2
```

where:

e: is the number of edges in the control flow graph

n: is the number of nodes in the control flow graph

Cyclomatic complexity represents the minimum number of paths that should be tested. The more complex the code, the more intense the testing effort is for that component.

Bad Fix Probability

Bad fix probability represents the likelihood of inadvertently inserting a new bug while attempting to fix a known one. Bad fix probability looks at a procedure and assesses the odds of introducing a new bug. Well-written code would typically generate a lower bad fix probability percent. Derived from McCabe Metrics, bad fix probability correlates with the understanding level and complexity results.

Understanding Level

Similar to complexity and bad fix probability, understanding level evaluates how easily a developer can interpret and maintain the code. Understanding level evaluates code as follows:

Table 3-8. Understanding Level Metric

Range	Understanding Level
Less than 5	Simple
Between 5 and 10	Simple to moderate
Between 11 and 20	Moderate
Between 21 and 30	Moderate to high
Between 31 and 50	High
Between 51 and 94	High to untestable
Greater than 94	Untestable

Correlating All Metrics

The following table shows how all three metrics correlate with each other.

Table 3-9. Correlation of McCabe Metrics

Code Complexity Range	Bad Fix Probability Percent	Understanding Level Interpretation
Less than 5	1%	Simple
Between 5 and 10	5%	Simple to moderate
Between 11 and 20	10%	Moderate
Between 21 and 30	20%	Moderate to high
Between 31 and 50	30%	High
Between 51 and 94	40%	High to untestable
Greater than 94	60%	Untestable

Viewing Call Graph Data

The **Call Graph** tab displays a static view of inbound and outbound call path corresponding to the method or property selected from the Code Review solution tree (see Figure 3-8 on page 80).

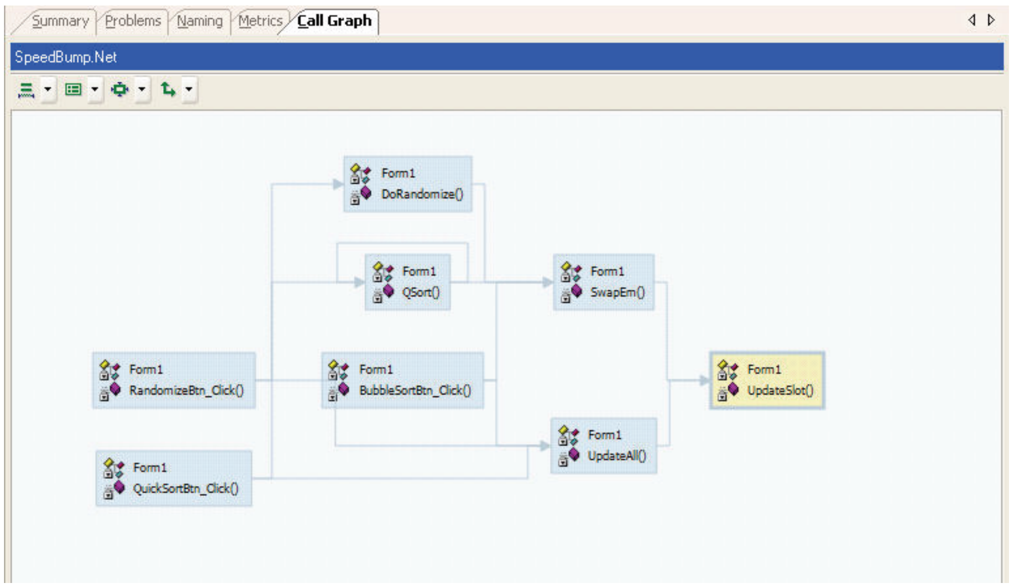


Figure 3-8. Call Graph tab showing example of Call Graph representation

Call paths are statically generated. This means that the graph shows the potential method calls in the call path, rather than the dynamic calls made during program execution.

The **Call Graph** tab is empty if:

- You did not select the **Collect call graph data** check box on the General options page prior to the code review. Call graph data was not collected during the review. To perform call graph analysis and collect call graph data, select this option and then perform another code review.
- You selected the check box, but did not select a method or property on the Code Review solution tree (see Figure 3-1 on page 62). Data was collected but no call graph appears until you select a method or property node on the Code Review solution tree.

Understanding Call Graph References

The **Call Graph** tab shows potential inbound/outbound call references in a call path by tracing the call hierarchy for the selected method or property. The display area shows the potential entry and exit points for each method or property. The call references start at the root node with all calls performed in reference to the root node. The call references continue until control returns to the root node, or the call is completed from the root node. The following types of call references appear in the display area:

Root Node

The root node refers to the method or property selected to be the starting point of the call graph. All other nodes either call into the root node or are called by it. The root node (Figure 3-9) appears as a light yellow rectangle with a wide blue border, which distinguishes it from all other nodes in the display area.

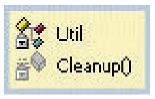


Figure 3-9. Example of Root Node

Inbound Calls

Inbound refers to methods or properties that directly or indirectly call into the root node. The inbound calls (Figure 3-10) are shown as light blue rectangular nodes, which differentiates them from the root node.

Outbound Calls

Outbound refers to methods or properties that are directly or indirectly called by the root node. As with the inbound calls, the outbound calls (Figure 3-10) appear as light blue rectangular nodes. They are connected by a series of arrows, pointing away from the root, to show the potential direction of the call path.

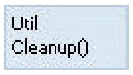


Figure 3-10. Example of Inbound or Outbound Call Node

Uncalled References

Uncalled refers to a method or property that is defined in the code but never referenced within the files that form an application component. The **Call Graph** tab identifies uncalled methods on a node using either the label **Uncalled** or the symbol (**!**).

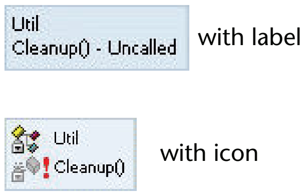


Figure 3-11. Two examples of Uncalled Identification

Recursive and Circular Call References

The **Call Graph** tab can graphically show instances of recursive or circular call references that exist in the selected path of execution.

- Recursive: Method or property that calls itself in the path of execution.

  A calls B;
  B calls B

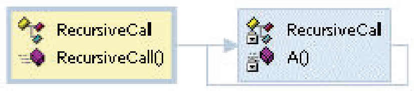


Figure 3-12. Example of Recursive Call graph

- Circular: Method or property that indirectly calls back into a previously called method or property in the path of execution.

  A calls B;
  B calls C;
  C calls back to A

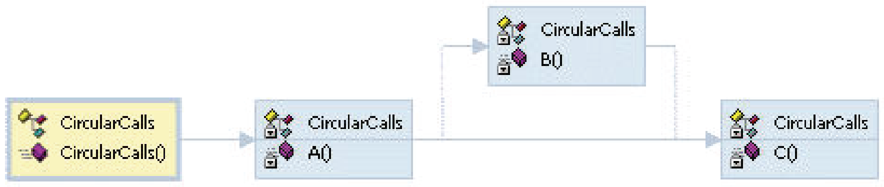


Figure 3-13. Example of Circular Call graph

Setting Call Graph Configuration Options

DevPartner code review provides four ways to configure how a call graph appears in the **Call Graph** tab. Access these options either from the Call Graph toolbar or by right-clicking on the background area of the **Call Graph** tab.

Number of Levels

Choose the number of levels to be displayed on the **Call Graph** tab. The call graph shows a specified number of levels of methods or properties that call into (inbound) and are called from (outbound) the root node. You can choose between one and six levels (six, default). The following example shows two levels selected. The plus signs (+) on the nodes to the right of the call graph indicate that more levels of call references are available for viewing.

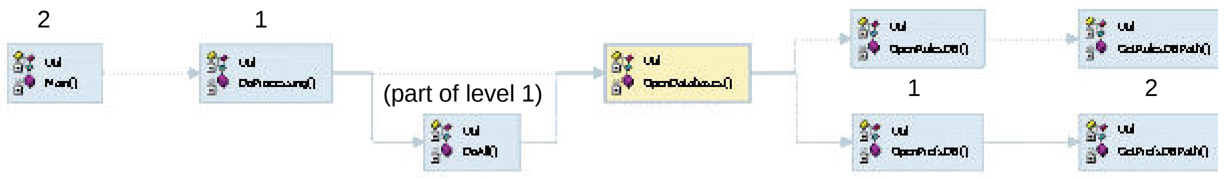


Figure 3-14. Two-level configuration

Node Style

You can choose the node style that is applied to the **Call Graph** tab. All call graph node styles show the class name, as well as the method or property name. Some node styles also include icons indicating the access type of the class, method, or property: public, private, internal, or protected. These are standard Solution Tree icons. Other icons, representing uncalled methods and properties, only appear in the call graph.

The following table shows examples of the various node styles.

Some examples show root node and others use standard node (inbound or outbound). See "Understanding Call Graph References" on page 81 for more information on how nodes are differentiated.

Table 3-10. Node Styles

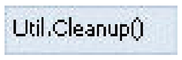
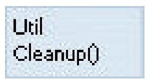
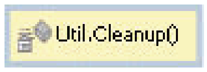
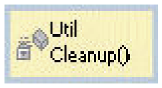
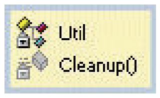
Node Style	Description	Uncalled Representation	Examples
Single label	Shows the class name, then a period, followed by the method or property name, but without icons	The designation - `Uncalled` appends the method or property name.	 Util.Cleanup() Inbound/outbound
Top and bottom labels	Shows the class name appearing on the first line and the method or property name appearing on the next, but without icons	The designation - `Uncalled` appends the method or property name on the second line.	 Util Cleanup() Inbound/outbound

Table 3-10. Node Styles

Node Style	Description	Uncalled Representation	Examples
One image and label	Shows a standard method or property icon, plus the class name, then a period, followed by the method or property name, all on the same line	The corresponding icon includes an exclamation point icon (**!**).	 Root
One image and two labels	Shows an icon for the method or property, along with the class name on the first line and the method or property name on the second line	The corresponding icon includes an exclamation point icon (**!**).	 Root
Two images and two labels	Shows an upper-level icon for the class followed by the class name, and a lower-level icon for the method or class, followed by its name	The explanation point icon (**!**) icon appears between the data type icon and the name.	 Root

Scaling

Choose the relative size of the call graph on the **Call Graph** tab. Two scaling options are available:

◆ **To fit in available space (default)**

This selection lets you scale the call graph so that all the nodes fit within the display area. By default, scroll bars are not available with this choice. If you reconfigure the call graph using the other options, the contents resize without the inclusion of scroll bars.

◆ **By percent of full size**

This selection lets you enlarge or shrink the contents in the display area by one of these fixed percentage values: 100%, 80%, 75%, 66%, or 50%. This choice allows you to zoom into sections of a large or complicated call sequence. Moreover, when the contents are redrawn, the selected method or property (root node) is clearly visible in the display area. Scroll bars are also available.

Layout

Choose how the call graph nodes appear on the **Call Graph** tab. Choices include:

◆ **Horizontal**

The nodes appear in a left-to-right orientation in the display area. The methods or properties calling into the selected node (also called the root node) are located to its left. The methods or properties that the selected node calls into branch to the right.

- **Vertical**

  The nodes appear in a top-to-bottom orientation in the display area. The methods or properties calling into the selected root node are located above the root node. The methods or properties that the selected node calls into are located below it.

Using the Command Line Interface

You can run a batch script from the command line interface (using **`CRBatch.exe`**) to review large solutions with many managed projects, or as an overnight or automated build process. The command line interface streamlines the code review process by bypassing user interaction.

Note: If the solution file is set to read-only, Visual Studio interrupts the batch review with an error message.

When you select **Always generate a batch file** on the General options page, code review generates a batch file during the next interactive code review performed in Visual Studio. You can use this batch file to run the batch review on the same solution.

Note: If you use the `/r` option, you should disable **Always generate a batch file**, or backup and rename your batch file. Otherwise, your batch file is overwritten.

The command line interface generates an HTML-formatted summary file (**`CR_solution-name.htm`**) in the solution folder after a review completes. This file is identical in content to the session file generated interactively.

You can script a batch procedure that reviews your solution, and then:

- Emails the generated summary and session files to another location
- Saves the summary file to a local intranet for later viewing from that location, or from an external Internet Web site
- Calls **`CRExport.exe`** to export the data to XML for even more formatting and display options (see "Exporting Data to XML" on page 88)

If code review cannot execute a batch review, it creates an error file, **`CR_solution-name.err`**. If the batch file fails on an attempted export to XML, it creates an error file **`CREXPORT_sessionfiledatabasename.err`**. Both error log files are created in the same path as the session file.

Syntax and Options

Run a code review session from the command line or batch file using the following command line syntax and options:

```
CRBatch.exe [/?] /f filename [/v] [/r] [/vs version] [/l XML filename]
```

Table 3-11. Command Line options

Option	Definition
/?	Displays the list of command line options for **`CRBatch.exe`**

Table 3-11. Command Line options

Option	Definition
/f *filename*	Identifies the configuration file to use in the review (mandatory)
/v or /verbose	Instructs the command line interface to report errors in a message box and to set the exit code used by the batch procedures (optional, although useful when physically debugging configuration files)
/r or /results	Instructs the command line to examine the results of the review for coding problems and naming violations, and return a specific error code if either or both error types were found (optional)
/vs "11.0" /vs "10.0" /vs "9.0" or /vs "8.0"	Indicates the Visual Studio .NET Framework version where the batch review executes: 11.0 (2012) 10.0 (2010), 9.0 (2008) or 8.0 (2005)
/l *XML path/file name*	Performs Code Review only on the projects that are listed in the specified XML file. You can select individual projects in a solution to be included in the code review. Use the Code Review options to select the projects. See "Using the Project List file in CRBatch" for more information. Selecting projects within a solution creates an XML file containing the list of selected projects. Specify that XML file using the /l parameter to only process the projects in the list for a code review. The /l parameter is not required, but if used, the XML path/file name must follow. If the /l parameter is used and the XML path/file name is not specified or can not be found, an error displays (or is logged) and the code review will not run. See "Understanding the Error File" for more information on logged errors.

Using the Project List File in CRBatch

When running code review from the command line, you can choose some or all of the projects in a solution to process. Projects to be processed are read from a project list XML file.

Code Review generates this file if a code review is first run against the solution in Visual Studio. The list is based on projects selected for the solution in the Projects to be reviewed table of the Code Review General Options page.

When the project list file is generated, it is named **`CR_[`*`solutionname`*`].xml`** (where **`[`*`solution name`*`]`** is the name of the solution). This file is stored in the solution folder. Once created, a project list XML file can be used in the Code Review command line interface.

If you do not wish to run a code review in Visual Studio to create the project list XML file for command line use, you can create your own project list XML file. The following shows project list XML file structure.

```
<CRProjectList CRVersion="11.0.0">

    <!-- DevPartner CodeReview Project file-->

    <Projects>

        <Include IncludeType="Include">

            <Project>Project A</Project>

            <Project>Project B</Project>

        </Include>

    </Projects>

</CRProjectList>
```

If creating your own project list XML file, the file name must have a .XML extension to be used in CRBatch.

The project list can be inclusive or exclusive. By default, the project list XML file created by running a code review in Visual Studio is inclusive, as shown above. To make the list exclusive, replace the `Include` specifier value with `Exclude` as shown in the following:

```
<CRProjectList CRVersion="11.0.0">

    <!-- DevPartner CodeReview Project file-->

    <Projects>

        <Include IncludeType="Exclude">

            <Project>Project C</Project>

            <Project>Project D</Project>

            <Project>Project E</Project>

        </Include>

    </Projects>

</CRProjectList>
```

Using the Project List XML file When Running Code Review From the Command Line

The following shows an example of the `CRBatch` command that uses a project list XML file. The file name must proceed the `/l` parameter in the command syntax.

```
CRBatch /F
"c:\VS.NET2008\WindowsApplication2\CR_WindowsApplication2.CRB" /vs
"9.0" /l
"c:\VS.NET2008\WindowsApplication2\CR_WindowsApplication2.xml"
```

Any CRBatch command that includes the `/l` parameter can also be used in a batch file.

Understanding the Error File

The following error codes are returned to a calling batch process when the command line interface exits.

Table 3-12. Command line error codes

Error Number	Message
0	Successful
1	No configuration file specified
2	Configuration file does not exist
3	No solution file was specified
4	Solution file does not exist
5	CRBatch initialization error
6	Invalid command line argument
7	Create Visual Studio process failed
8	License check failed
9	Visual Studio exited with an error
10	Visual Studio version number incorrect
11	Unexpected error
12	Coding problems found
13	Naming violations found
14	Coding problems and naming violations found
70	Attempt to create error file (`.ERR`) failed

If a batch-generated review encounters a build error or compile errors exist in the solution being reviewed, the batch review stops without generating a session or summary file. An error message is appended to the error file.

Error 11 is returned for unexpected runtime errors. The error details (error message and stack-trace) are written to the `.ERR` file.

Exporting Data to XML

DevPartner code review allows you to export session results data to XML, providing you with a simple way to port your results data into report formats, e-mail, an internal Web page, etc. You can export your data to XML:

- From code review, after running a code review session
- From the command line, using a saved session file
- In an automated batch process, using a saved session file

The **`DPCRExport.xsd`** schema file, located in the CodeReview installation folder, describes the contents and XML format for exported data.

Exporting Session Data from within DevPartner

After completing a code review, you can export all data from the current session file to an XML file. Select **Export DevPartner Data** from the **File** menu and provide a name for the export file. By default the file is saved in the same location as the solution, but the file does not appear in the solution explorer.

You must maintain focus in the code review session window to export code review data.

This process always exports all session data, including inbound methods from the call graph data. To be more selective about what categories of data you export to XML, use the command line.

If code review cannot export your session data to XML, it generates an error message describing the problem it encountered.

Exporting Session Data from the Command Line

DevPartner code review includes a command line utility, **`CRExport.exe`**, that exports the results of a code review session to an XML file. To export session data you must specify the session file and the output file using the mandatory command line arguments. For example:

```
CRExport.exe /f C:\MyResults\WebApp1.DPMDB /e C:\MyXML\WebAppData
```

Optional command line arguments also let you specify the categories of data to export from the session database file.

If you call **`CRExport.exe`** without passing it any of the optional arguments, it exports all session data, including inbound methods. This behavior is equivalent to passing **`CRExport.exe`** the `/a i` arguments, or initiating the data export from within DevPartner.

If the export utility cannot create the export file, it generates an error log file, **`CREXPORT_sessionfiledatabasename.err`**, in the same path as the session file.

Syntax and Options

Export your session data to XML via the command line or batch file using the following command line syntax and options:

```
CRExport.exe [/?] /f sessionfile /e xml_exportfile [/a | /a i | /p |
/m | /n | /s | /c | /c i]
```

Table 3-13. Command line options

Option	Definition
/?	Displays the list of command line options for **`CRExport.exe`**
/f *sessionfile*	Identifies the session database to use for this export (mandatory)
/e *xml_exportfile*	Identifies the XML file to receive the exported data (mandatory)
/a	Exports all data for the specified session, including the outbound methods for call graph data, but Inbound methods are not exported
/a i	Exports all data for the specified session, including inbound and outbound methods for call graph data

Table 3-13. Command line options

Option	Definition
/p	Exports the problems data for the specified session
/m	Exports the metrics data for the specified session
/n	Exports the naming analysis data for the specified session
/s	Exports the code size data for the specified session
/c	Exports the outbound, or called, methods in the call graph data for the specified session
/c i	Exports the call graph data, including inbound and outbound methods, for the specified session

Exporting Session Data from a Batch Process

You can use **`CRExport.exe`** along with **`CRBatch.exe`** as a single batch process to conduct a code review, and then export the session data to XML. This feature is especially useful when you already run a code review via batch process:

- As part of a nightly build process
- On very large applications
- To automate your quality control testing

Understanding Naming Analysis

The code review feature incorporates two kinds of naming analysis capabilities:

- **Naming Guidelines**

  The naming analyzer supports the .NET Framework. See "Understanding the Naming Guidelines Naming Analyzer" on page 91.

- **Hungarian**

  The Hungarian naming analyzer is a legacy naming analyzer in code review. See "Understanding the Hungarian Naming Analyzer" on page 93.

  You can also choose **None** from the **Naming analysis to use** list on the General options page to bypass naming analysis altogether.

Understanding the Naming Guidelines Naming Analyzer

The Naming Guidelines naming analyzer is patterned after the Visual Studio .NET Framework naming guidelines. These naming guidelines ensure that consistent, predictable, and manageable naming practices are applied to .NET Framework types in a managed class library.

Choose **Naming Guidelines** from the **Naming analysis to use** list on the General options page, plus make additional selections on the Naming Guidelines options page to ensure a more precise review.

The Naming Guidelines naming analyzer examines the following:

- Parameters
- Classes
- Namespaces
- Methods
- Delegates
- Enums
- Structs
- Interfaces
- Variables

The naming analyzer looks for naming violations in the source code related to capitalization, case sensitivity, abbreviations and acronyms, and syntax for namespaces and other .NET Framework identifiers.

The following sections describe guidelines that the Naming Guidelines naming analyzer follows.

Capitalization

When it finds a naming violation, code review attempts to suggest a more appropriate name on the **Naming** tab using the capitalization style that you selected on the Naming Guidelines options page — Camel or Pascal.

Table 3-14. Capitalization styles used in Naming Guidelines Naming Analyzer

Capitalization Style	First Concatenated Word	Subsequent Concatenated Words	Examples of Suggested Names
Camel case	Not initial-capped	Initial-capped	**r**ed**C**olor
Pascal case	Initial-capped	Initial-capped	**R**ed**C**olor

Case Sensitivity

DevPartner code review discourages using case sensitivity to differentiate identifiers in the source code. Case *insensitivity* is strongly encouraged because it supports interoperation between case-sensitive and case-insensitive programming languages and also reduces confusion between two similarly-named identifiers. Developers should avoid names that vary only by case. Rather, they should use names that are functional in either case-sensitive or case-insensitive programming languages.

Abbreviations and Acronyms

DevPartner code review supports the use of generally accepted abbreviations and acronyms. DevPartner code review determines proper naming based on:

- The number of letters for the abbreviation or acronym
- The position of the abbreviation or acronym in the identifier name

Namespace Syntax

DevPartner code review supports the .NET Framework naming convention for namespaces. The namespace name starts with the company name, followed by the technology name, and optionally ends with the feature and/or design name. Here is an example of the syntax:

```
CompanyName.TechnologyName[.Feature][.Design]
```

By default, code review recommends Pascal case for namespaces (see "Selecting Camel Case or Pascal Case" on page 70). The period character (.) separates each logical concatenated word. Enter the namespace information in the **Namespace options** field on the Naming Guidelines options page prior to the review.

Syntax for Other .NET Framework Identifiers

DevPartner code review checks for properly named .NET Framework identifiers in the source code. Here are some examples of what code review looks for:

- **Numeric characters**

  DevPartner code review checks whether numbers are part of the identifier name. While code review does not remove the numeric characters, it does flag the name as a violation.

- **Underscore characters**

  DevPartner code review looks for instances of the underscore character (_) in the identifier name. The underscore character is discouraged in the Naming Guidelines naming analyzer. DevPartner code review removes the underscore character except in the following cases:

  - If the underscore is a leading character (i.e., _redColor)
  - If it is used in a method name
  - If its removal introduces another naming violation

- **Casing for constants**

  DevPartner code review follows Pascal or Camel casing for constants (depending on the case selection you made on the Naming Guidelines options page), rather than all uppercase. For example, code review would change the constant `HTTP_PORT` in:

  ```
  private const int HTTP_PORT = 80
  ```

  - To httpPort based on Pascal

  - To httpPort based on Camel

- **Delegate**

  If a delegate identifier name includes the word delegate (regardless of case) along with one or more identifiable words, code review removes the word delegate as long as it does not introduce another violation. For example, the name, `MyDelegateWord`, would be renamed as `MyWord`.

Understanding the Hungarian Naming Analyzer

DevPartner code review includes the Hungarian naming analyzer, patterned after the Hungarian Notation naming convention.

With Hungarian naming, variable names include specific character(s) that identify a particular scope-level or data-type prefix for the variable in question. For example, the data-type prefix `int` signifies an integer, such as integer variable `Port`; and the scope-level prefix `g_` signifies global, as in `g_intPort`.

DevPartner code review uses the Hungarian naming analyzer in a code review when the **Hungarian** option is selected from the **Naming analysis to use** list on the General options page. DevPartner code review also uses the currently selected name set. When you start a code review, the naming analyzer evaluates scope-level prefixes and data-type prefixes for every variable in the code. If applicable, code review makes recommendations consistent with the name set (*Default* preferred) and displays the naming results on the **Naming** tab (Figure 3-5 on page 75) following the code review.

The Hungarian naming analyzer does not evaluate parameter names.

The following tables list examples of scope-level and data-type prefix combinations that are evaluated in the Hungarian naming analyzer, as specified in the current name set.

Table 3-15. Scope prefix

Scope	Prefix
Global	g_
Member	m_
Local	““

Table 3-16. Data Type prefix

Data Type	Prefix
string	str
int	int
int	i
boolean	bool
bool	bln

The qualifiers on a variable declaration determine the scope, such as the boundaries where the variable exists. For example, code review considers a variable with public status as having a global scope because it is accessible outside the class.

The default name set contains scope prefixes that you can edit using the Rule Manager. You can also customize variable and object names, based on Hungarian Notation, using the Rule Manager.

Constructing a Hungarian Naming Suggestion

The Hungarian naming analyzer makes a more appropriate suggestion when it encounters one or more of the following anomalies in your source code:

- It finds an incorrect or missing scope prefix (e.g., `m_` for global, instead of `g_`)
- It finds an incorrect or missing data type prefix (e.g., `Short`, instead of `intShort` for integer type)
- You selected the **Warn if the first letter after the prefix is not capitalized** check box (on the New Rule Set or Edit Rule Set dialog box in the Rule Manager) to apply to the Hungarian name set, but the first letter of the variable name after the prefix is not capitalized.

The naming analyzer combines the following: Scope Level Prefix + Data Type Prefix. If you have not specified a scope-level prefix in the Hungarian name set, the suggested name would begin with the data type prefix.

DevPartner code review displays **Unknown** on the **Naming** tab, rather than attempt to suggest a name if code review cannot recognize the data type for a variable because:

- The data type does not exist in the current Hungarian name set
- You have selected the **Warn if unknown objects are found** check box on the New Rule Set or Edit Rule Set dialog box in the Rule Manager to apply to the name set

See “Using the Code Review Rule Manager” on page 94 for more information on managing name sets.

Using the Code Review Rule Manager

DevPartner code review contains an extensible rules database that is based on the Microsoft Visual Studio programming standards. The rules database is maintained and stored in the Rule Manager standalone application. With Rule Manager, you can configure rules, triggers, and rule sets. You can also configure Hungarian name sets that the Hungarian naming analyzer uses during a code review. The Rule Manager stores any modifications you make to the code review rules database. These modifications become immediately available when you configure and perform your next code review.

Access Rule Manager by selecting **Micro Focus > DevPartner Studio > Utilities > Code Review Rule Manager** from the **Start** menu.

Configuring Rules

Use the Rule Manager to create, edit, and delete rules. You can also add HTML links to the rule descriptions to provide more information for developers trying to resolve violations.

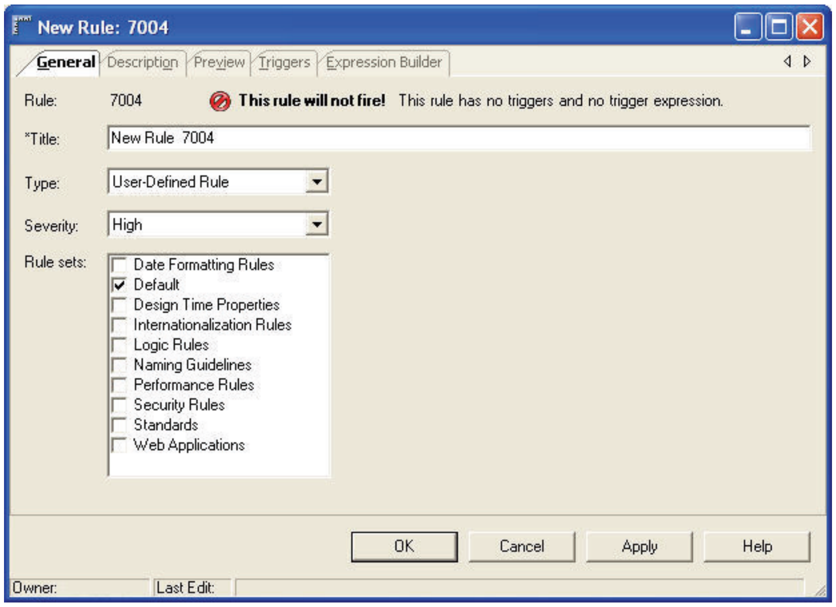
Creating Rules

Use the New Rule dialog box to create and configure a new rule. To create a new rule, complete the following steps:

1 Select **Rule** > **New Rule**.

The New Rule dialog box opens with the **General** tab displayed by default. The title bar shows the pre-assigned rule number. The status bar shows the current Owner and Last Edit details (see Figure 3-15).

Until you create a Trigger and Expression for the rule, it does not fire.



Rule: 7004 **This rule will not fire!** This rule has no triggers and no trigger expression.

*Title: New Rule 7004

Type: User-Defined Rule

Severity: High

Rule sets:
- [ ] Date Formatting Rules
- [x] Default
- [ ] Design Time Properties
- [ ] Internationalization Rules
- [ ] Logic Rules
- [ ] Naming Guidelines
- [ ] Performance Rules
- [ ] Security Rules
- [ ] Standards
- [ ] Web Applications

Owner: Last Edit:

Figure 3-15. New Rule dialog box

2 Set up the new rule using the following tabs (in this order):

a **General** — to enter general rule properties

b **Description** — to enter details about the rule

c **Preview** — to review current entries

d **Triggers** — to configure up to five triggers for the rule

e **Expression Builder** — to build trigger expression(s) for each trigger

3 Click the **Description** tab and add a description for the rule.

Use the Description tab to provide HTML links that direct developers to external resources to help resolve coding issues. These links appear in the lower panel (Description pane) of the **Problems** tab following a code review session.

4 Click the **Triggers** tab, and add a trigger for your rule.

   See “Configuring Triggers” on page 97 for more information on creating a trigger.

5 Select the **Expression Builder** tab to build a trigger expression.

   You can build an expression for each trigger you just configured on the **Triggers** tab. For more information on building trigger expressions, refer to the Rule Manager online help.

Editing Rules

Use the Edit Rule dialog box to modify existing rule properties. The Edit Rule dialog box contains all the same fields as the New Rule dialog box (see Figure 3-15 on page 95). To edit an existing rule, complete the following steps:

1 Select **Rule > Edit Rule**.

   The Edit Rule dialog box opens with the **General** tab displayed by default. The title bar shows the rule number and title. The status bar shows the current Owner and Last Edit details.

2 Modify the existing rule using the following tabs (in this order):

   a **General** — to modify existing rule properties

   b **Description** — to modify details about the rule

   c **Preview** — to review current entries

   d **Triggers** — to modify settings for the existing triggers

   e **Expression Builder** — to modify trigger expression(s) for each trigger

Deleting Rules

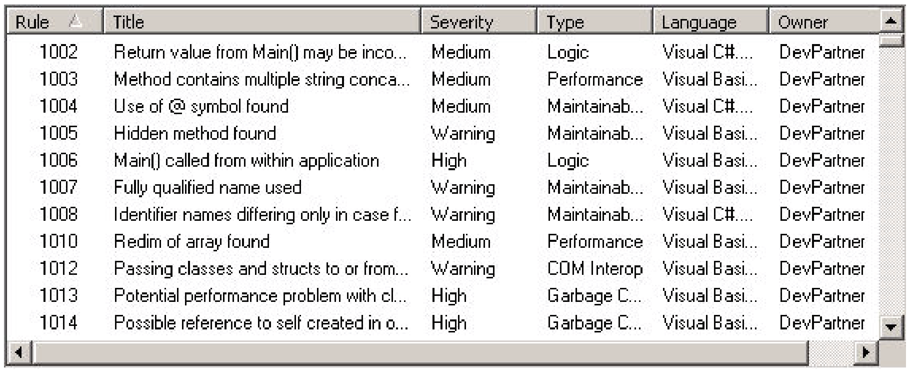
You can only delete user-configured rules that reside in **All Rules**, not any DevPartner-supplied rules. When you delete a user-defined rule from **All Rules**, the Rule Manager automatically deletes it from any other rule sets where it also resides.

Editing a DevPartner-supplied rule removes it from system ownership but does not change it to user-defined status. You cannot delete it from **All Rules**.

To delete a rule, complete the following steps:

1 Select **All Rules** from the **Rule Set** list.

   The rules in **All Rules** appear in the **Rule List** pane.



Rule	Title	Severity	Type	Language	Owner
1002	Return value from Main() may be inco...	Medium	Logic	Visual C#....	DevPartner
1003	Method contains multiple string conca...	Medium	Performance	Visual Basi...	DevPartner
1004	Use of @ symbol found	Medium	Maintainab...	Visual C#....	DevPartner
1005	Hidden method found	Warning	Maintainab...	Visual Basi...	DevPartner
1006	Main() called from within application	High	Logic	Visual Basi...	DevPartner
1007	Fully qualified name used	Warning	Maintainab...	Visual Basi...	DevPartner
1008	Identifier names differing only in case f...	Warning	Maintainab...	Visual C#....	DevPartner
1010	Redim of array found	Medium	Performance	Visual Basi...	DevPartner
1012	Passing classes and structs to or from...	Warning	COM Interop	Visual Basi...	DevPartner
1013	Potential performance problem with cl...	High	Garbage C...	Visual Basi...	DevPartner
1014	Possible reference to self created in o...	High	Garbage C...	Visual Basi...	DevPartner

Figure 3-16. Rules List pane

2 Select one or more user-defined rules in the Rule List pane.

3 Select **Rule** > **Delete Selected Rules from Rules Database**.

The Rule Manager deletes the selected rule(s) from the **All Rules** rule set. This action cannot be undone.

Delete Selected Rules from Rule Database is only enabled in the **Rule** menu once you have selected **All Rules** from the **Rule Set** list.

Configuring Triggers

Select the **Triggers** tab to configure up to five triggers that fire a rule.

Although some DevPartner-supplied rules use macros, you cannot edit or configure a trigger for any rules that are macro-based.

If no triggers are associated with the rule you are configuring, the **Existing Triggers** list box is the only visible field and appears empty. In addition, the **Add** button becomes available so that you can add a new trigger to the setup.

If one or more triggers already appear in **Existing Triggers**, the other fields applicable to the trigger's Type are displayed, and required fields appear with an asterisk. The **Add** and **Delete** buttons are available for setup.

Adding a Trigger

Complete the following steps to add a trigger:

1 Click the **Add** button to add a new trigger.

The Rule Manager displays a default name, *New Trigger n*, in the **Trigger Name** field. For example, if this is your first trigger, the name is **New Trigger 1**.

When you reach the five trigger limit, the **Add** button on this pane automatically disables.

2 Enter or change the trigger name.

Do not use left `[` or right `]` brackets, such as in `[CheckString]`. If you type in bracket characters, the Rule Manager ignores these keystrokes. Brackets delimit multiple triggers in a trigger expression within the **Expression** field on the Expression Builder pane. An attempt to manually insert brackets invalidates a trigger expression.

3 Select a trigger type from the **Type** list.

The trigger type selection determines the remaining parameters for the trigger being configured (see Table 3-17).

Table 3-17. Trigger types

Type	Function
Code	Detects problems in the actual source code
Web Form Page	Ensures compliance with HTML and/or ASP.NET tag construction
Design Time Property	Isolates the trigger firings to specific Visual Studio .NET properties
Web.config	Ensures compliance with elements in ASP.NET Web.config files

4 Configure all the required fields for the selected trigger type.

Depending on the Type selected, supply different parameters for your trigger. For more information on configuring your specific trigger type, see the Rule Manager online help.

5 Add a regular expression for your rule to the Regular Expression text box (see "Creating New Rules Using Regular Expressions" on page 102).

Deleting a Trigger

To delete a trigger listed in **Existing Triggers**, complete the following steps:

1 Select the trigger and click **Delete**.

A confirmation message appears.

2 Click **Yes** to confirm or **No** to abort this action.

You cannot delete a trigger if it is already being used in a trigger expression.

Configuring Rule Sets

Rule sets are collections of rules you can use in a code review session. DevPartner code review includes a selection of pre-configured rules sets. You might find you want to work with custom rule sets, and you can use the Rule Manager to create, edit, or delete rule sets.

Creating Rule Sets

The Rule Manager includes a master rule set called **All Rules**. However, you can create additional rule sets tailored to your project-specific requirements. To create a new rule set, complete the following steps:

1 Select **File** > **New Rule Set**.

The New Rule Set dialog box appears.

2 Enter a rule set name in the **Rule Set Name** field (up to thirty characters).

3 Enter a brief description for the new rule set in the **Description** field (optional).

4 Select a Hungarian name set from the **Use Set** list in the **Hungarian Name Sets** section of the dialog box.

   Name sets in the Rule Manager only support the Hungarian naming analyzer, patterned after the Hungarian naming convention. They do not support the Naming Guidelines naming analyzer, patterned after the Visual Studio .NET naming guidelines.

5 Choose how you prefer Hungarian naming violations to appear on the **Naming** tab:

   - ◇ If you select **Warn if unknown objects are found**, code review specifies a naming violation as **Unknown** if it cannot make a suggestion.
   - ◇ If you select **Warn if the first letter after the prefix is not capitalized**, code review makes a suggestion.

6 Click **OK**.

   The Rule Manager validates the new rule set.

7 Populate the rule set with rules by:

   - ◇ Creating new rules (see "Creating Rules" on page 95).
   - ◇ Opening an existing rule set in order to select, copy, and paste rules into the new rule set.

Editing Rule Sets

To edit the properties of a rule set, complete the following steps:

1 Select an existing rule set from the **Rule Set** list.

2 Select **File > Rule Set Properties**.

   The Edit Rule Set dialog box appears. The Edit Rule Set dialog box has the same available fields as the New Rule Set dialog box.

3 Enter a rule set name in the **Rule Set Name** field (up to thirty characters).

4 Enter a brief description for the new rule set in the **Description** field (optional).

5 Select a Hungarian name set from the **Use Set** list in the **Hungarian Name Sets** section of the dialog.

   Name sets in the Rule Manager only support the Hungarian naming analyzer, patterned after the Hungarian naming convention. They do not support the Naming Guidelines naming analyzer, patterned after the Visual Studio .NET naming guidelines.

6 Choose how you prefer Hungarian naming violations to appear on the **Naming** tab:

   - ◇ If you select **Warn if unknown objects are found**, code review specifies a naming violation as **Unknown** if it cannot make a suggestion.
   - ◇ If you select **Warn if the first letter after the prefix is not capitalized**, code review makes a suggestion.

7 Click **OK**. The Rule Manager validates the changes to the rule set properties.

Deleting Rule Sets

To delete an existing rule set, complete the following steps:

1. Select a rule set from the **Rule Set** list. You can delete user-defined rule sets, but not Dev-Partner-supplied rule sets.
2. Select **File > Delete Rule Set**. The Delete Rule Set dialog box appears.
3. Click **Delete**. This action cannot be undone.

Configure Hungarian Name Sets

Use the Rule Manager to create, edit, duplicate, or delete Hungarian Name Sets used by the Hungarian Naming Analyzer during a code review session. To access the Hungarian Name Sets dialog box, select **File > Hungarian Name Sets**.

Creating a Hungarian Name Set

Complete the following steps to create a new Hungarian Name Set:

1. Click **New**. The New Hungarian Name Set dialog box appears.
2. Replace **Untitled** in the uppermost field with a unique name for the name set.
3. Click **Create**. After you click **Create**, the **Add**, **Edit**, and **Delete** buttons are enabled.
4. Select the applicable language to apply to this new name set. Once you select the language, the Rule Manager verifies the new name.

Editing a Hungarian Name Set

Complete the following steps to edit an existing Hungarian name set:

1. Select a name set on the Hungarian Name Sets dialog box.
2. Click **Edit**. The Edit Hungarian Name Set dialog box appears.
3. Edit the language, variables, and objects associated with the name set as you see fit. You cannot edit the name of a Hungarian name set.

Duplicating a Hungarian Name Set

You can duplicate a Hungarian name set. This lets you create a new name set using an existing name set as a template. To duplicate a Hungarian name set, complete the following steps:

1. Select a name set on the Hungarian Name Sets dialog box.
2. Click **Duplicate**. The Duplicate Hungarian Name Set dialog box appears.
3. Replace **Copy of <*name*>** in the uppermost field with a unique name.
4. Click **Create**.

   After you click **Create**, the **Add**, **Edit**, and **Delete** buttons are enabled. Rule Manager verifies the name set.

Deleting a Name Set

You can only delete a user-defined Hungarian Name Set that is not currently in use by a rule set.

To delete a Hungarian Name Set, complete the following steps:

1 Select **File > Hungarian Name Sets**.

The Hungarian Name Set dialog box appears.

2 Select the name set to delete.

The Rule Manager highlights all the variables and objects associated with that Hungarian name set within each tabbed pane, and disables the **Add**, **Edit**, and **Duplicate** buttons.

3 Click **Delete**.

The Delete Hungarian Name Set dialog box appears.

4 Click **OK** to delete the selected name set.

The Hungarian Name Sets dialog box appears. This action cannot be undone.

5 Click **OK**.

Manipulating the Rule List

There are two ways you can manipulate the rules displayed in the Rule List:

Filter the Rule List View

Use the Filter pane, located on the left side of the Rule Manager window below the **Rule Set** list, to filter contents appearing in the Rule List pane (see Figure 3-16 on page 97).

1 Select a rule set from the **Rule Set** list.

The Rule Manager automatically lists all rules in the database when you select **All Rules in Set**. Select an individual rule set to filter selections.

2 Click the **Filter** tab.

3 Select (or clear) at least one item from each group at **Filter** options.

You can select the group check box or click + to expand and make individual choices from within the group.

You must pick at least one item from each group. If you do not, then a mouse pointer directs you to the area needing attention. The groups include:

◇ **Type** — Rules coincide with programming technologies.

◇ **Severity** — Choices include High, Medium, or Low, and Warning. Use Warning to call attention to a particular coding problem.

◇ **Language** — Only languages that apply to the selected rule set appears.

  - ◇ **Owner** — DevPartner-supplied rules reference the identifier DevPartner. All other rules become the ownership of the individual who created the rule. The Rule Manager only displays owners that apply to the selected rule set.

4. Click **Apply** to filter the current view.

Find a Specific Rule

Use the **Find** tab, located on the left side of the Rule Manager window below the **Rule Set** list, to search for one or more rules.

1. Select a rule set to search in from the **Rule Set** list.

   If you choose **All Rules in Set**, all rules in the rules database appear.

2. Click the **Find** tab to display search options.
3. Select a criteria from the **Search Rule Set For** list. You can also select recent search strings from the **Contains** list.
4. Enter a string at **Contains** to define the specific search condition.
5. Click **Find**.

To perform a subsequent search, choose either of the **Search In** options:

- ◆ **All rules in set** — To initiate a new search
- ◆ **Current results** — To continue searching within the current results

You can optionally change search criteria, noted above, and then click **Find** again.

Creating New Rules Using Regular Expressions

You can create your own rules in code review and use them to identify many suspect coding practices. DevPartner code review rules make extensive use of regular expressions, which provides a robust and versatile method for searching text.

Regular expressions are widely used, well-documented, and can be written to match patterns in HTML, Visual Basic, and Visual C# syntax. DevPartner code review uses the same regular expression engine as Microsoft Visual Studio and supports the same syntax.

DevPartner code review makes it easier to use regular expressions in its rules by limiting the scope of any given rule to certain parts of the code. For example, a rule can apply to the entire file, just methods, or only `While` blocks. Since rules can specify a scope, the regular expressions can focus on a targeted part of the code.

DevPartner code review also assists the regular expression search by removing comments from a code block. Removing comments before executing the review reduces false positives.

The following sections provide examples of actual code review rules and explain the regular expressions that drive them.

To learn more about how to write regular expressions for your code review rules, refer to the following resources:

◇ Forta, Ben. *Teach Yourself Regular Expressions in 10 Minutes*. Indiana: Sams Publishing, 2004.

◇ Friedl, Jeffrey E.F. *Mastering Regular Expressions.* 2nd ed. California: O'Reilly, 2002.

◇ Goyvaerts, Jan. *Regex Tutorial, Examples and Reference*.
1 Feb. 2006 <http://www.regular-expressions.info>.

◇ Microsoft Corporation. *.NET Framework Regular Expressions*. 2006. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcomregularexpressions.asp>

Matching Lines Exceeding 90 Characters

Best practice coding standards recommend that a line of code should not exceed 90 characters. A code review rule enforces this standard by searching for lines that exceed 90 characters. The following regular expression ensures lines do not exceed 90 characters in length:

```
(?-s).{91,}
```

This regular expression first sets the Single Line option to **False**, causing the expression to evaluate all characters up to, but not including, the newline character (`\n`) as a single line. This evaluation treats each line of code, from its beginning to the newline (`\n`) character, as a distinct and different line.

Next, the rule incorporates the most elementary aspect of regular expressions — matching single characters. This rule uses the period (`.`) metacharacter to match any single character on the line.

The rule follows the period (`.`) with a repeating match metacharacter `{91,}`. Repeating match metacharacters specify that a match must repeat a certain number of times, or within a certain range of instances. In this rule, it specifies the expression is true only if any single character is matched 91 or more times; the second value in the range is left empty, because the rule only cares if the number of matches exceeds 90 characters. Table 3-18 describes the basic repeating match metacharacters.

Table 3-18. Repeating match metacharacters

Character	Meaning
+	Matches one or more instances of the preceding character
*	Matches zero or more instances of the preceding character
?	Matches zero or one instance of the preceding character

{*n*}
{2,6}
{*n*,}
	Matches an exact number of instances of the preceding character where *n* represents the number of required repetitions
	These braces are also used to designate a range of repetition, such as from two to six times, by including the upper and lower limit separated by a comma.
	Omitting the upper limit matches against a minimum number of instances without an upper bound

Matching Tabs Used Instead Of Spaces

Best practice coding standards recommend that spaces be used instead of tabs. The number of spaces represented by a tab can differ between editors, and this difference can cause the source code to have a different appearance in each editor. To enforce a consistent appearance of the source, spaces should be used. The following regular expression is used in a code review rule that searches for the use of tabs inside of methods:

```
(?s)\t.*
```

This regular expression sets the **Single Line** option to **True**, causing the expression to evaluate every character on every line up to and including newline characters (`\n`), as part of a single line.

Next, the rule specifies a match against the tab character by using a metacharacter (`\t`). Without further change, the regular expression would find every occurrence of a tab character in the method. Instances where multiple tabs are used in a method, such as for indenting lines, would fire the rule for each tab in that method. That is not the intended behavior of the rule.

This rule should evaluate to true if at least one tab is used in the method, but not every time it encounters a tab in the method. To accomplish this result, the rule needs the period (`.`) metacharacter followed by a repeating match metacharacter specifying zero or more instances. Table 3-18 on page 103 shows the repeating match metacharacter to use is the asterisk (`*`). Adding these last two metacharacters specifies that the rule must evaluate to true the first time it encounters a tab character, and then capture every following character in the method.

Matching Instances Where Code Catches System.Exception

Avoid catching `System.Exception` to handle your errors because it does not catch errors at a fine enough level of detail to allow for the proper differentiation of error types. Error handling code blocks should intercept and handle errors at the finest granularity as possible, since doing so can make a program more robust and less likely to crash. The following regular expression is used in a code review rule designed to find instances in the code where Visual Basic syntax is used to catch `System.Exception`:

```
Catch\s\w+\sAs\s(System\.)?Exception
```

The first part of this expression locates any instance of the literal word `Catch` in the code. Since the rule should not match instances where `Catch` is the first part of a longer word, the literal text is followed by the metacharacter for whitespace (`\s`).

Visual Basic syntax uses the word `Catch` followed by a variable name (used to hold the exception object). The variable is followed by more whitespace, and the literal word `As`.

The rule needs regular expression functionality to locate a legal variable name, followed by more whitespace, and the word As. The metacharacter `\w`, paired with the repeating match metacharacter `+`, locates one or more instance of any alphanumeric character (upper or lower case) or the underscore character. Adding `\sAs\s` finishes the search for a legal variable name followed by whitespace and the word As.

So far, the regular expression locates the following code:

```
Catch MyExceptionObject As
```

This regular expression would successfully locate all code that is catching exceptions. However, the rule should only match against code that catches `System.Exception`. The regular expression requires further refinement.

To ensure that the regular expression only matches instances where the code catches `System.Exception`, it searches for the literal words `System` and `Exception` separated by a period. Since the period is a metacharacter, the rule needs to specify a match on a literal period by preceding it with the backslash, removing its special character status.

If the rule now has `System\.Exception` as part of the regular expression, there is still a problem. It is acceptable syntax for the catch of `System.Exception` to leave off the `System.` and only use the term `Exception`. One last modification to the regular expression makes the matching of `System.` optional. Wrapping `System\.` in parenthesis makes it a subexpression, which can be followed by the `?` metacharacter to specify zero or one match.

Matching Methods Having More Than One Return Point

Best practice coding standards recommend that methods have only one return point. Having more then one return point could cause code to be hard to understand. The following regular expression is used in a code review rule that locates instances where a method has more than one return point. Most of the pieces making up this expression have been used in previous rules, but there are a couple of new things to examine.

```
(?s)(\breturn\b.*){2,}
```

First the rule sets the **Single Line** option to true, using `(?s)`, to focus on the entire method. To consider the method as a whole, the rule needs to evaluate every character on every line, up to and including newline characters (`\n`), as part of a single line.

Another part of the expression used in earlier rules is the repeating match metacharacter at the end. This expression uses `{2,}` to modify the preceding subexpression (contained in parenthesis), requiring that there must be two or more matches within the method.

The subexpression, `(\breturn\b.*)`, is the part of the regular expression doing most of the work. It is written as a subexpression to allow the whole block to be modified by the repeating match metacharacter. The metacharacter `\b` is a word boundary. By surrounding the literal text `return` with the word boundary metacharacters, the regular expression looks for instances of the word standing alone, not as part of a larger word.

Note: The previous example followed the literal text `Catch` by the whitespace metacharacter **`\s`** to ensure it would only find instances where `Catch` was a whole word. This is a good example of flexible regular expressions. That rule could have used the word boundary metacharacter `\b`, but did not.

The final `.*` within the subexpression searches for a match of zero or more instances of any character. The rule is now complete, and searches entire methods for two or more instances of the word `return`, followed by zero or more characters.

Enforcing Initialization Of Variables When They Are Defined

As a best practice, to keep code concise and easy to understand, variables should always be initialized when they are defined. The following regular expression is from a code review rule that locates instances where a variable is defined, but not initialized:

```
(?-s)\bDim\b(?!.*=)(?!.*\bnew\b)
```

Since the rule needs to evaluate each line of code by itself, the first thing it does is set the **Single Line** option to false.

Next, the regular expression is going to look for the word `Dim`. It wraps the literal text `Dim` with word boundary metacharacters `\b` to ensure it only considers whole words.

The subexpression implements the concept of looking ahead or behind. The ability for regular expressions to look ahead or behind gives them additional flexibility.

Looking ahead or behind means that a subexpression in the regular expression is searching for a match. Instead of matching and returning the specified text itself, like a simple string match would, the subexpression only verifies that the match exists. Finding a match causes the subexpression to evaluate to true, which then allows the rest of the regular expression to succeed or fail according to its other qualifiers. This kind of looking ahead and behind is referred to as a *positive look ahead* or *positive look behind*, because the subexpression evaluates to true when it finds text that matches.

The syntax for the positive look ahead and positive look behind is:

- Positive look ahead `(?=subexpression)`
- Positive look behind `(?<=subexpression)`

Similarly, *negative look ahead* and *negative look behind* work by searching for text that does not match the subexpression specified in the statement.

The syntax for the negative look ahead and negative look behind is:

- Negative look ahead `(?!subexpression)`
- Negative look behind `(?<!subexpression)`

The regular expression for this rule needs to use the negative look ahead construct to detect when there is *not* an equal sign (`=`) to the right of the `Dim` keyword, with or without a space preceding it. The subexpression `(?!.*=)` handles that negative look ahead.

The last part of the expression, `(?!.*\bnew\b)`, uses another negative look ahead to evaluate to true if the word `new` does not exist to the right of the `Dim` keyword, with or without a preceding space.

The complete rule now has a regular expression that evaluates to true whenever it encounters a line of code where the word `Dim` is not followed by an equal sign (`=`) or the word `new`.

Matching Instances Of More Than One Statement Per Line

In order to increase readability and maintainability of code, only one statement should ever be placed on a single line (with the exception of loop syntax). The following regular expression is from a code review rule that locates instances where a line contains more than one statement:

```
(?<!for.*);.*;
```

It might appear that the easiest way to detect more than one statement on a given line would be to determine if any line contains more than one semicolon. In fact, this search is the essence of the regular expression in this rule, but it needs to also take into account the possibility of a semicolon being associated with the `for` keyword.

To exclude any instances where the keyword `for` is associated with a semicolon, the rule uses the negative look behind construct `(?<!for.*)` to look back on any line where it encounters a semicolon, making sure the word `for` is not there. The remaining part of this regular expression (`;.*;`) searches for a semicolon followed by any number of other characters, and then another semicolon.

Ensuring Open Braces Are Placed On A Separate Line

Best practice coding standards recommend that open braces should be placed at the beginning of their own separate line following the statement that begins the block. The following regular expression is from a code review rule that locates instances where open braces are not placed on their own separate line:

```
(?m)^\s*\w+(?=.*?\{).*?$
```

There are several new concepts at work in this expression. The rule first sets the **Multi Line** option to true with `(?m)`. This setting changes the behavior of two other metacharacters — **Line Beginning** (`^`) and **Line End** (`$`). By enabling the multi-line option, `^` and `$` capture the beginning and end of each line rather than the entire string being searched.

Once multi-line mode is enabled, the regular expression searches for the beginning of the line (`^`), followed by one or more whitespace characters (`\s*`), and one or more word characters (`\w+`). This sets up the basis that the regular expression uses. If it finds one or more word characters, there should be no open braces on the line.

The positive look ahead subexpression `(?=.*?\{)` searches through each line looking for any character followed by an open brace. The backslash before the open brace removes its metacharacter status. Once the rule determines that a line contains a character followed by an open brace, and the open brace is not on a line by itself, `.*?` at the end of the regular expression allows it to capture the remaining text right to the end of the line (matched by the `$` metacharacter).

Ensuring Loop Counters Are Not Modified Inside the Loop Bodies

Changing the loop counter inside the body of the loop could cause unpredictable results, and makes code harder to understand. The following regular expression is from a code review rule that locates any instance where a loop counter is modified inside of the loop body:

```
(?s)\bfor\b\s*\(\s*\w+\s+(?<VARNAME>\w+).*\).*\b\k<VARNAME>\b\s*=
```

This regular expression is extremely long because it has to do a lot of work to enforce the rule. To identify and store the loop counter variable name, the regular expression must first capture the `for` keyword, left parenthesis, and loop counter type.

The first half of the regular expression is gathering required information. It locates a line with the `for` keyword, followed by any number of whitespace characters, a left parenthesis, more whitespace characters, one or more word characters, more whitespace, and finally uses a subexpression to capture the variable name.

The subexpression `(?<VARNAME>\w+)` captures the name of the loop counter and store it in the variable, `VARNAME`.

This construct can use any variable name, as long as the name does not contain any punctuation and does not begin with a number.

Once it has captured the name of the loop counter, the last part of the regular expression captures any remaining characters and the right parenthesis. It then begins searching through the loop body for an instance of the loop counter, followed by an equal sign, using the following construct:

```
.*\b\k<VARNAME>\b\s*=
```

This part of the expression is the essence of the rule. Prior to reaching this point, the regular expression has determined the name of the loop counter and has placed the scope of its search within the loop body. This final part of the expression now matches all characters up to the value of `VARNAME` (the loop counter), and then looks for an equal sign following the counter.

The fact that the loop counter is followed by an equal sign indicates it is being set to some value, or modified. Since the counter should never be modified inside of the loop body, the rule has found a violation.

Submitting Data to Visual Studio Team System\Team Foundation Server

DevPartner Studio supports Microsoft Visual Studio Team System if the Team Explorer client is installed and a Team Foundation Server connection is available.

Visual Studio Team System and Team Foundation Server Support in DevPartner Code Review

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected item. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected item. Access Work Item submission in any of the following tabs in a code review session file:

- Problems tab (see “Viewing Code Violations” on page 73)
- Naming tab (see “Viewing Naming Violations” on page 74)

When you submit a bug, DevPartner populates the **Work Item** form with data from the tab. For more information about DevPartner Studio integration with Visual Studio Team System, see “Visual Studio Team System Support” on page 20.

Chapter 4

Automatic Code Coverage Analysis

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with coverage analysis. The second section provides reference information for an in-depth understanding of DevPartner Studio's coverage analysis feature.

Refer to the DevPartner Studio online help for additional task-oriented information about coverage analysis.

What is Coverage Analysis?

DevPartner Studio's coverage analysis feature allows developers and test engineers to be sure that they are testing all of an application's code. When you run your tests with coverage analysis, DevPartner tracks all components, images, methods, functions, modules, and individual lines of code covered by your tests. When your tests end, DevPartner displays information showing you what code was exercised and what code was not exercised.

DevPartner collects coverage data for 32-bit and 64- bit managed applications, including Web and ASP.NET applications, as well as 32-bit and 64- bit unmanaged (native) C++ applications.

Using Coverage Analysis Out of the Box

The following Ready, Set, Go procedure introduces you to using DevPartner to analyze code coverage.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject being described in the shaded box, read the additional text following the box.

Analyzing an application with DevPartner Studio does not require elevated system privileges. The system privileges used to create and debug an application are sufficient for DevPartner to analyze the application.

Ready: Consider What You Want to Analyze

Before using code coverage, consider what you want to analyze.

The following procedure assumes:

- You are working in a supported release of Visual Studio.
- You are testing a single-process, managed application.
- You can build and run the application.
- Your solution includes a startup project.

See“DevPartner Studio Supported Project Types” on page 267 for a comprehensive list of supported project types for DevPartner coverage analysis.

When analyzing applications, decide what data you are interested in collecting before beginning your coverage session. In some cases, there are steps you need to take before beginning a session. For example, some set-up would be required if:

- there are modules you want to omit from the coverage analysis
- if there are unmanaged modules that you would like analyzed
- if you want to include code run on a remote server

For this procedure, all managed, local code in the application is analyzed.

Set: Properties and Options

Once you have decided what code you want included in the coverage analysis, you can set several properties and options to focus your data collection.

For this procedure, you can use the default DevPartner properties and options. No additional set-up is required.

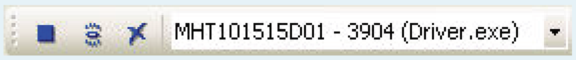
Use Solution Properties and Project Properties to choose whether your analysis session data should include information for .NET assemblies and COM that runs outside of the application. Use DevPartner options to change data display options, exclude parts of applications from analysis, or create a session control file to manage data collection. See “Setting Properties and Options” on page 115 for more information.

Go: Collect Coverage Data

After considering what you want to analyze and setting the appropriate properties and options, you are ready to collect coverage data.

1. From Visual Studio, open the solution associated with the application.
2. Select **DevPartner > Start with Coverage Analysis** to begin a coverage analysis session.

   During a session, the Session Control Toolbar options are active.

   MHT101515D01 - 3904 (Driver.exe)

   DevPartner session controls let you focus coverage analysis on any phase of the application. Use the session controls to stop data collection, take a snapshot of the data currently collected and then continue recording, or clear data collected but not yet saved in a snapshot.
3. Run the code you want to analyze.
4. Click the **Snapshot** icon . (Click twice if necessary to bring focus to the session window.) When you take a snapshot, DevPartner creates a file containing the collected data, called a session file, and displays the session file data.
5. Return to the application and continue running tests.
6. When you are finished running your tests, exit the application. The final session file displays in Visual Studio.

Note: If a security exception message displays when attempting to collect data for a managed application, see page 118 for information about changing your security policy.

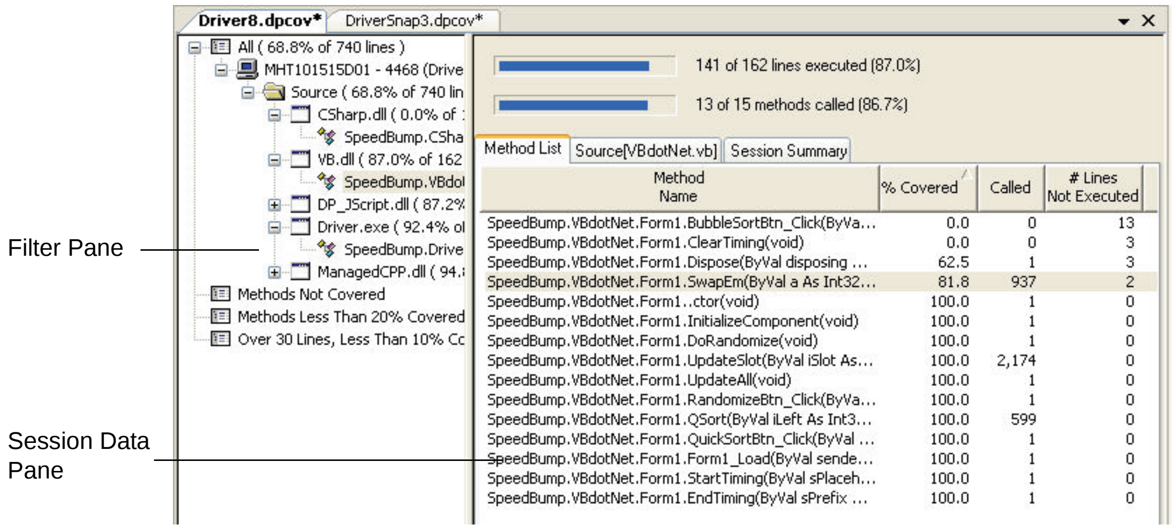
You can analyze coverage in conjunction with the DevPartner error detection feature. Knowing how much of your code was covered by your tests helps you gauge the comprehensiveness of your error detection data. See "Integration with DevPartner Error Detection" on page 130 for more information about running a session with both error detection and coverage analysis.

Analyze the Data

When taking a snapshot or exiting an application, DevPartner displays the session file in Visual Studio, as shown in Figure 4-1 on page 112. The session window consists of:

- The filter pane, which lists the source files and images in the application and shows the lines covered in each as a percentage of the total lines in the file.
- The session data pane, which contains three tabs and two coverage meters that display data for the item selected in the filter pane.

- The coverage meters, displayed above the tabs in the session data pane, summarize the line and function coverage for the item selected in the filter pane.



All (68.8% of 740 lines)
MHT101515D01 - 4468 (Drive
Source (68.8% of 740 lin
CSharp.dll (0.0% of
SpeedBump.CSha
VB.dll (87.0% of 162
SpeedBump.VBdo
DP_JScript.dll (87.2%
Driver.exe (92.4% o
SpeedBump.Drive
ManagedCPP.dll (94.
Methods Not Covered
Methods Less Than 20% Covered
Over 30 Lines, Less Than 10% Co

141 of 162 lines executed (87.0%)

13 of 15 methods called (86.7%)

Method Name	% Covered	Called	# Lines Not Executed
SpeedBump.VBdotNet.Form1.BubbleSortBtn_Click(ByVa...	0.0	0	13
SpeedBump.VBdotNet.Form1.ClearTiming(void)	0.0	0	3
SpeedBump.VBdotNet.Form1.Dispose(ByVal disposing ...	62.5	1	3
SpeedBump.VBdotNet.Form1.SwapEm(ByVal a As Int32...	81.8	937	2
SpeedBump.VBdotNet.Form1..ctor(void)	100.0	1	0
SpeedBump.VBdotNet.Form1.InitializeComponent(void)	100.0	1	0
SpeedBump.VBdotNet.Form1.DoRandomize(void)	100.0	1	0
SpeedBump.VBdotNet.Form1.UpdateSlot(ByVal iSlot As...	100.0	2,174	0
SpeedBump.VBdotNet.Form1.UpdateAll(void)	100.0	1	0
SpeedBump.VBdotNet.Form1.RandomizeBtn_Click(ByVa...	100.0	1	0
SpeedBump.VBdotNet.Form1.QSort(ByVal iLeft As Int3...	100.0	599	0
SpeedBump.VBdotNet.Form1.QuickSortBtn_Click(ByVal ...	100.0	1	0
SpeedBump.VBdotNet.Form1.Form1_Load(ByVal sende...	100.0	1	0
SpeedBump.VBdotNet.Form1.StartTiming(ByVal sPlaceh...	100.0	1	0
SpeedBump.VBdotNet.Form1.EndTiming(ByVal sPrefix ...	100.0	1	0

Figure 4-1. Coverage Analysis Session window

Using the Filter Pane and the Session Data Pane

In addition to listing files and images in the application, the filter pane also includes a set of filters to help focus on the most significant data.

To begin evaluating your data, start by using a filter to reduce the amount of data displayed, and then examine the **Method List** to find methods that were least covered by your tests.

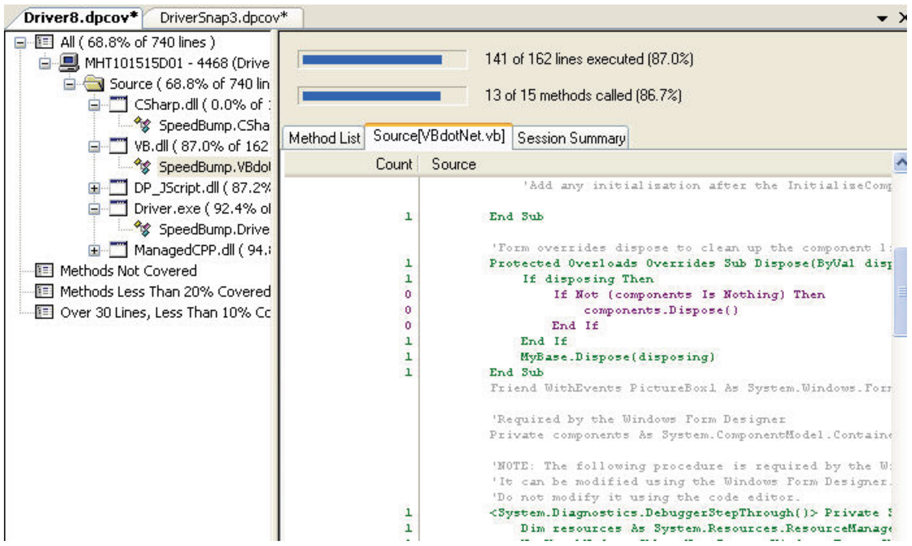
1 In the Filter pane, click on the **Methods Less Than 20% Covered** filter. This reduces the displayed data and help you focus on methods that were least exercised.

2 Examine the data on the **Method List** tab to discover how much of each method was adequately covered by your tests.

If there is application code that is inadequately covered, revise tests to cover more application functionality.

Viewing Source Code

The **Source** tab displays the source code for the item selected in the filter pane. Use the **Source** tab to help you identify the functionality that requires more test coverage.



```
141 of 162 lines executed (87.0%)
13 of 15 methods called (86.7%)
```

```
Count  Source
                'Add any initialization after the InitializeComp

  1         End Sub

            'Form overrides dispose to clean up the component l
  1         Protected Overloads Overrides Sub Dispose(ByVal disp
  1             If disposing Then
  0                 If Not (components Is Nothing) Then
  0                     components.Dispose()
  0                 End If
  1             End If
  1             MyBase.Dispose(disposing)
  1         End Sub
            Friend WithEvents PictureBox1 As System.Windows.Forn

            'Required by the Windows Form Designer
            Private components As System.ComponentModel.Containe

            'NOTE: The following procedure is required by the W
            'It can be modified using the Windows Form Designer.
            'Do not modify it using the code editor.
  1         <System.Diagnostics.DebuggerStepThrough()> Private S
  1             Dim resources As System.Resources.ResourceManage
```

Figure 4-2. The Source Code tab

You can display the code for a specific method in a source file by double-clicking on the method in the **Method List**.

3 On the **Method List** tab, double-click a method with a low value in the **% Covered** column. The source code for that method is displayed on the **Source** tab, as shown in Figure 4-2

The **Source** tab indicates coverage data for each line of code. DevPartner highlights the lines that were executed (green by default), not executed (purple by default), and lines that cannot be executed, such as comments (gray by default).

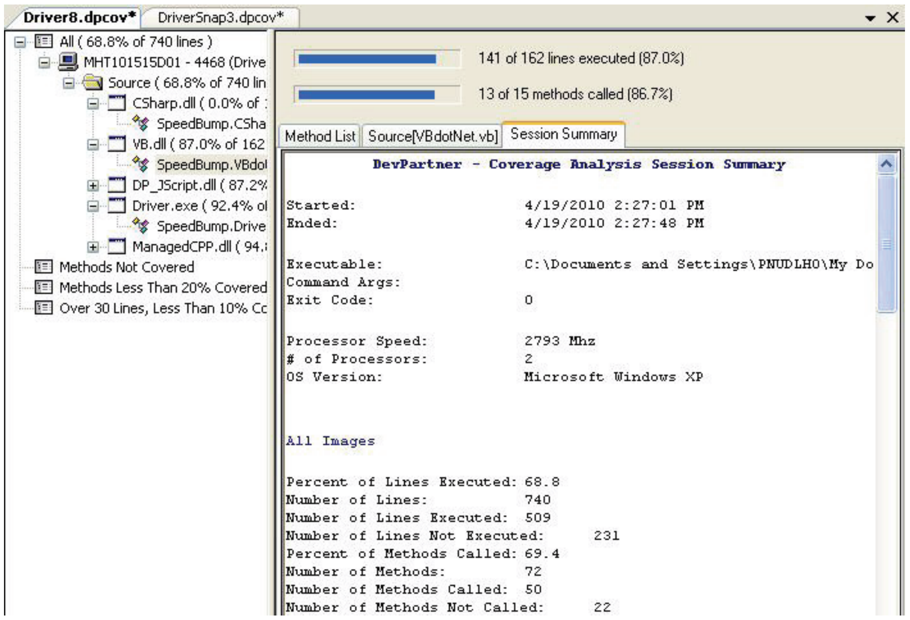
The **Count** column displays the number of times the line was executed.

Note: To present source code data for managed applications, DevPartner requires program database file (PDB) information.

On the **Source** tab, you can right-click on a line to view the context menu, from which you can go to the previous unexecuted line, the next unexecuted line, choose the columns to display, or choose another source file to view.

Viewing Session Summary Data

The **Session Summary** tab displays a synopsis of the coverage analysis session.



```
          DevPartner - Coverage Analysis Session Summary


Started:                    4/19/2010 2:27:01 PM
Ended:                      4/19/2010 2:27:48 PM


Executable:                 C:\Documents and Settings\PNUDLHO\My Do
Command Args:
Exit Code:                  0


Processor Speed:            2793 Mhz
# of Processors:            2
OS Version:                 Microsoft Windows XP



All Images


Percent of Lines Executed: 68.8
Number of Lines:            740
Number of Lines Executed:   509
Number of Lines Not Executed:       231
Percent of Methods Called: 69.4
Number of Methods:          72
Number of Methods Called:   50
Number of Methods Not Called:       22
```

Figure 4-3. The Session Summary tab

4 Click on the **Session Summary** tab.

The Session Summary includes contextual information about the session, such as the date and time of the session, the processor speed and operating system, and so on. This information can be useful when viewing an older session file, particularly one that was created by someone else.

The summary also includes coverage data from the filter pane and the Method List tab, showing data for both the files and the methods that were analyzed.

5 Scroll through the tab to view the session summary data.

Saving Session Files

When you have finished reviewing coverage data you can save the session file. If you have created more than one session file, you can merge the coverage data from multiple session files.

1 Close the session file window in Visual Studio to save the session file. When prompted, accept the default file name and location. By default, the file is saved in the project's output folder.

2 If there are multiple coverage session files for this solution, you might be prompted to merge the files, or the merge might occur automatically, depending on the **Merge** setting in your solution properties. Refer to "Solution Properties" on page 115 for information about the **Automatically Merge Session Files** property.

DevPartner saves session files as part of the active solution. They appear in the DevPartner Studio virtual folder in Solution Explorer. Coverage session files take the **`.dpcov`** extension.

By default, DevPartner physically saves the session files in your project's output folder. DevPartner automatically increments the file name based on the contents of the default folder (for example, **`MyApp.dpcov`**, **`MyApp1.dpcov`**, and so on). If you save session files to a location other than the default folder, you must manage the file naming and numbering.

For projects that do not have an output folder, such as a Visual Studio 2005 Web site project, DevPartner physically saves the files to the project folder.

Session files generated from the command line are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

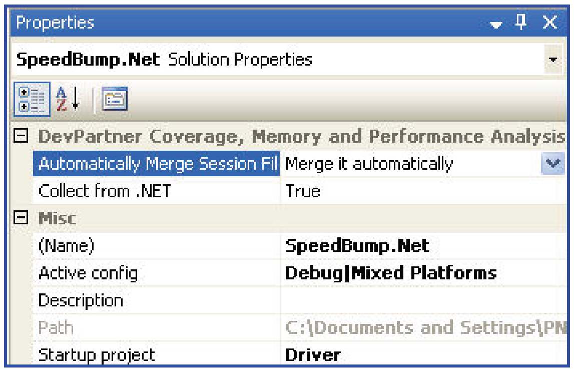
This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a coverage analysis session, continue reading the rest of this chapter for additional information, or refer to the DevPartner online help for task-based information.

Setting Properties and Options

Before beginning a coverage analysis session, it is often useful to fine-tune data collection to include or omit certain types of information. Use Solution Properties, Project Properties, and DevPartner options to better focus your analysis session.

Solution Properties

To view coverage properties available at the solution level, select the solution in the Solution Explorer and press F4 to view the Properties window.



DevPartner Coverage, Memory and Performance Analysis

Automatically Merge Session Fil	Merge it automatically
Collect from .NET	True

Misc

(Name)	SpeedBump.Net
Active config	Debug|Mixed Platforms
Description	
Path	C:\Documents and Settings\PN
Startup project	Driver

Figure 4-4. Solution properties

The following solution properties affect coverage analysis:

- **Automatically Merge Session Files -** Controls merge behavior for coverage analysis sessions (described in “Merging Session Data” on page 125).
- **Collect from .NET -** Visible only for managed code applications. Set this property to false if you do not want DevPartner to collect information for .NET assemblies.

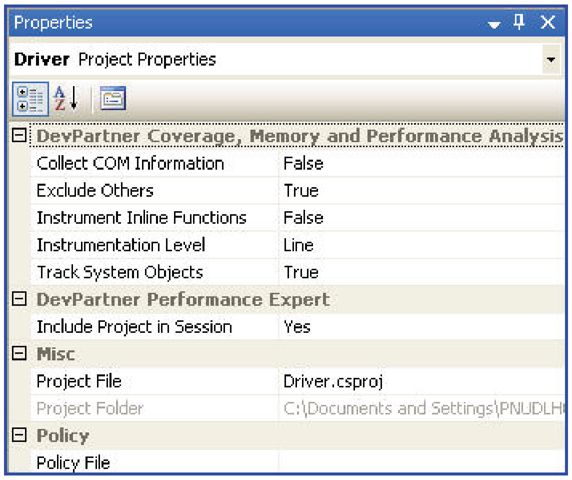
This property affects only coverage analysis and performance analysis sessions. Memory analysis and Performance Expert always collect data from managed applications, even when this value is set to false.

The **Collect from .NET** property is not available with DevPartner for Visual C++ Boundschecker Suite.

- **Startup project -** The solution must include a startup project. If the solution contains multiple startup projects, DevPartner prompts you to choose a startup project for the session.

Project Properties

To review project level properties, select a project in the Solution Explorer and review the properties that can be set for projects within the solution.



Properties

Driver Project Properties

DevPartner Coverage, Memory and Performance Analysis	
Collect COM Information	False
Exclude Others	True
Instrument Inline Functions	False
Instrumentation Level	Line
Track System Objects	True
DevPartner Performance Expert	
Include Project in Session	Yes
Misc	
Project File	Driver.csproj
Project Folder	C:\Documents and Settings\PNUDLH
Policy	
Policy File	

Figure 4-5. Project properties

The following project properties affect coverage analysis:

- **Collect COM Information -** DevPartner collects method level data based on DLL exports and COM interfaces. Select False if you do not want DevPartner to collect information for COM that runs outside the application.
- **Instrument Inline Functions -** DevPartner always collects coverage data for inline functions in managed applications.
- For unmanaged code, set this property to **True** to instrument inline functions. Inline functions are not instrumented by default if inline optimizations are enabled.

All properties persist unless you explicitly change them.

Options

To review DevPartner option settings for coverage analysis sessions, choose **DevPartner > Options > Analysis.**

- The **Display** option lets you set the precision, scale, and units used when displaying your data.

- The **Exclusions** option lets you omit one or more images from data collection. Refer to "Excluding Images" for more information on excluding images.
- The **Session Control File** option lets you create a set of rules and actions to control the data that DevPartner collects as the application or module runs. Refer to "Analysis Session Controls" on page 303 for more information about session control files.

Other Visual Studio options, such as the **Environment > Fonts and Colors** options, also affect DevPartner features.

Excluding Images

When you run an application under coverage analysis, DevPartner collects data for all source and system images. However, you can use Exclusions to omit one or more images from analysis.

While viewing Analysis Options (**DevPartner** > **Options** > **Analysis)** select **Exclusions - Coverage**.

From the **Show** list at the top of the page, select one of the following:

- Global exclusions
- Local exclusions in current user folder
- Local exclusions in executable folder

The **Local exclusions in current user directory and Local exclusions in executable directory** options are available only when a solution is open and the executable folder differs from the current working folder.

Click **Insert** to add an image to the exclusion list. Type a name, or browse to the image you want to exclude. Allowable file types for exclusion are **`.exe`**, **`.dll`**, **`.ocx`**, and .netmodule. Use the **Files of type** list to limit the types of files displayed.

If you choose a .NET module (**`.netmodule`**), only the unmanaged parts of the module are excluded.

To remove an image from the list of exclusions, select the item and click **Delete**.

To save a copy of the exclusion list (**`nmexclud.txt`**) to another location, click **Save To**. Global exclusions are saved in **`nmexclud.txt`** in the **`\Analysis`** sub-folder in the DevPartner installation folder. Local exclusions are saved in **`nmexclud.txt`** for the application in the current working folder or in the application executable folder.

Exclusions do not apply to files compiled with Native C/C++ Instrumentation. For example, if you attempt to exclude an instrumented unmanaged C/C++ image, DevPartner still collects information for that file, although no system call information is collected. If you wish to exclude an unmanaged C/C++ image from data collection, do not instrument that image.

About Instrumentation

When you run a managed application, DevPartner inserts hooks into the byte code for each assembly as it is loaded by the compiler, a process called instrumentation. This code contains instructions that DevPartner uses to collect coverage data while your application is running.

DevPartner instrumentation does not change the actual files on disk; it only modifies the in-memory representation of files as they execute.

Unlike managed code, which DevPartner instruments at runtime, you must instrument unmanaged C/C++ code when you compile it. To instrument unmanaged code, DevPartner inserts hooks directly into your source code. DevPartner provides an Instrumentation Manager in which you specify the type of instrumentation to be used and specify any projects in the solution to exclude from instrumentation. (Refer to “Collecting Data for Unmanaged Code” on page 119 for more information about the Instrumentation Manager.) When you rebuild the unmanaged project, the hooks are inserted. To remove the hooks, turn off instrumentation by deselecting the Native C/C++ Instrumentation option from the DevPartner menu, and rebuild the project.

Collecting Data from Various Types of Applications

This section provides information about using DevPartner coverage analysis to collect data from different types of applications.

DevPartner supports all Visual Studio managed code languages, as well as unmanaged C/C++. DevPartner can also collect coverage data for JScript and VBScript Web applications when using Internet Explorer (IE) or Internet Information Services (IIS).

Refer to “DevPartner Studio Supported Project Types” on page 267 for a complete list of languages and project types supported in each version of Visual Studio.

Collecting Data From Managed Code

Many applications developed in Visual Studio are managed applications, such as C#, Visual Basic, and managed C++ applications.

When attempting to collect data for a managed application, a security exception message displays if your security policy prevents DevPartner instrumentation of your code. By default, assemblies must have the `SkipVerification` permission to be profiled. If you remove this permission from the permission set of the policy under which the code executes, or add imperative security declarations to the assembly that cause this permission to be revoked, the assembly can not be profiled.

To remedy this condition, enable secure profiling in one of two ways.

- Set the following global environment variable and retry profiling the application:

  ```
  NM_NO_FAST_INSTR=1
  ```

  This solution allows you to work around this issue, although it does exact a slight performance penalty.

- Change the policy for the assembly using the .NET Framework Configuration tool MMC snap-in, or by temporarily removing any imperative security declarations in the assembly.

See the *.NET Framework Developers Guide* in the Visual Studio online help for more information on security policy in Visual Studio.

Collecting Data for Unmanaged Code

When you build your unmanaged C++ application for coverage profiling with Native C/C++ Instrumentation, DevPartner works with the compiler to add instructions to your application image to collect coverage data at run time.

Use the DevPartner Native C/C++ Instrumentation Manager to choose the type of instrumentation that DevPartner applies at compile time. Instrumentation is the process of adding instructions to an image. You must instrument native (unmanaged code) C/C++ projects before you can collect data with DevPartner.

For Visual Studio 2010 and later, instrumentation settings apply only to the active solution build configuration. This means that for each solution, you can have seperate instrumentation settings to build debug and release versions of the application. This feature provides the ability to batch build all the required build configurations into your build process. You must apply your instrumentation settings project otherwise default instrumentation settings apply.

For Visual Studio 2010 and later, instrumentation settings are saved to the **`project.vcxproj.user`** files, which are in the same location as the project files.

To instrument unmanaged code:

1. Open the solution that contains the unmanaged C/C++ project for which you want to collect data.
2. If using Visual Studio 2010 or later, optionally use the Configuration Manager to select a build configuration. See Microsoft help for information on creating and editing solution build configurations.
3. Choose DevPartner > **Native C/C++ Instrumentation Manager**.
4. Select the **Instrument the projects checked below when I build my solution** check box and select a type of instrumentation**.** The type of instrumentation you choose must match the type of analysis you subsequently run.
5. Select the projects to be instrumented. By default, DevPartner instruments all unmanaged code in the solution. Clear the check boxes of modules to be omitted. Click **OK**.
6. If creating instrumentation settings for another build configuration in the solution, select the build configuration, then repeat the steps to create the instrumentation settings for that build configuration.
7. Rebuild the solution. DevPartner instruments the unmanaged C/C++ projects you selected. Select **Start with Coverage Analysis** to begin the analysis session.

   DevPartner saves the project selections you make in the Instrumentation Manager with the solution. Once you use the Instrumentation Manager to configure instrumentation, you can turn instrumentation on and off with the **Native C/C++ Instrumentation** option from the DevPartner menu or the **Native C/C++ Instrumentation** button on the DevPartner toolbar. Use the **Instrumentation Manager** only to change settings.

To remove instrumentation from an application at a later time, clear the **Native C/C++ Instrumentation** option from the DevPartner menu. The next time you start a coverage analysis session or rebuild the solution, Visual Studio rebuilds the solution without instrumentation.

Note: If your application calls unmanaged Visual Studio components, you must compile these components with DevPartner instrumentation for coverage analysis in Visual Studio. See the DevPartner Studio online help in Visual Studio for more information.

Mixed-mode C++ Files

With unmanaged (native) C++, you can compile your application as managed code with the `/clr` option, but mark sections of your code with `#pragma` (native). The compiler generates native code for any methods defined in the `#pragma` section. DevPartner does not support mixed-mode C++ files. When profiling a program that includes a C++ file with both managed and unmanaged (native) sections, DevPartner collects coverage data only for the managed code portions, not the native code portions from `#pragma`. To collect data for unmanaged C++ code, place the unmanaged code in a separate file and instrument it, as described in "Collecting Data for Unmanaged Code" on page 119.

Collecting Data from Multiple Processes

Applications may run more than one process. For example, when you profile an ASP.NET application you may see the browser process (`iexplore`), the IIS process (`inetinfo`), and the ASP worker process (`aspnet_wp` or `w3wp`).

When you run a multi-process application under coverage analysis, the DevPartner Session Control toolbar displays the active processes in the process selection list.

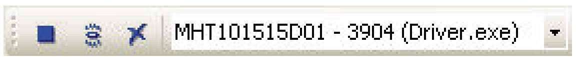


Figure 4-6. Session Control toolbar with the Process Selection list

Use the process selection list to focus data collection. When you take a snapshot, DevPartner creates a session file with data for the process selected in the process selection list.

Collecting Data from Remote Systems

You can use DevPartner to enable coverage data collection for application components running on a remote system. For example, you might want to collect coverage data for both client and server portions of a client/server application. With DevPartner, you can collect coverage data for client and server processes as you run the client application.

To collect data simultaneously from a client system and a remote system, install DevPartner on the client and install DevPartner and the DevPartner Remote Server license on the remote system. See the *DevPartner Installation Guide* and the *Distributed License Management Licensing Guide* for more information about the Remote Server license.

A server connected through a Terminal Services connection does not require the DevPartner Remote Server license. See "Using Terminal Services and Remote Desktop" on page 21 for information on Terminal Services.

On the remote system, select the relevant projects and review the DevPartner properties to ensure that they match options set on the client system. DevPartner restarts server processes, such as IIS, after you change options. This restart is necessary for changes to take effect.

Be sure to specify instrumentation if you are analyzing an unmanaged C++ application. If your application calls unmanaged C++ components, you must instrument those components if you want to collect data from them. See “Collecting Data for Unmanaged Code” on page 119.

Correlated Data

When using IE and IIS as browser and Web server, or using COM to make inter-process calls, DevPartner automatically recognizes a client/server relationship between the processes. To preserve the relationship between the methods of DCOM objects or the relationship between HTTP client and server (IE and IIS), DevPartner correlates the data from those sessions. It then combines the correlated data with the client session data into a single session file.

The correlated session file contains the coverage data for both the client and server portions of your application. The correlated session file appears in Visual Studio, like any other session file, with **`_co`** appended to the file name, as in **`appname_CO.dpcov`**.

You can use **DevPartner** > **Correlate** > **Coverage Files** to combine data from different session files when there is no COM-based relationship or client/server relationship between IE and IIS. You can also use the `NMCORRELATE` command line utility to manually combine data.

Collecting Data From .NET Web Applications

If you develop Web Forms, XML Web Services, or ASP.NET applications, you can use DevPartner to collect coverage data for both client and server portions of your application. You can configure DevPartner to collect data for IIS and ASP.NET running on the local computer or on a remote server.

If the Web application calls unmanaged (native) C++ components, you must instrument them using the DevPartner commands in Visual Studio. To collect data for native C++ components called by your application, you must instrument and rebuild the objects with Native C/C++ Instrumentation, as described in “Collecting Data for Unmanaged Code” on page 119. Instrument for coverage analysis. DevPartner collects data for only one analysis type in a session.

Note: DevPartner session files are saved with the current solution. Opening a Web project from IIS directly, as opposed to opening the project through Visual Studio, may cause a different solution file to be used. DevPartner session files created in the first solution would not be visible in the second solution.

Prerequisites

For DevPartner coverage analysis to successfully profile an ASP.NET application, the following two conditions must be met:

- The project must include a **`web.config`** file.
- The **`web.config`** file must include a compilation element with the debug attribute set to true. For example:

  ```
  <compilation debug=”true”/>
  ```

DevPartner can also collect data for in-process or out of process components called by your application.

Analyze ASP.NET Applications without Debugging

For optimum results, run coverage analysis without debugging.

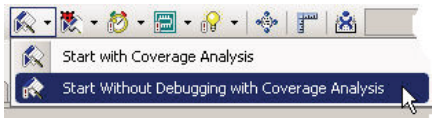

Start with Coverage Analysis
Start Without Debugging with Coverage Analysis

Figure 4-7. Start Without Debugging option

Only one script debugger can be active at one time. If you debug a Web application, both Visual Studio and DevPartner attempt to load a script debugger. A message displays indicating that the script debugger failed to attach to IE. The session continues without interruption despite the error message.

To avoid the error message, you can either disable script debugging in `iexplore` or run coverage analysis without debugging.

Unexpected File Save Dialogs or Saved Session Files

Under certain circumstances, you may see an unexpected File Save dialog box after stopping an ASP.NET application, or find that unexpected session files have been saved if you have configured DevPartner to automatically save session files.

When running coverage analysis on an ASP.NET application, DevPartner collects data for IE as the primary profiled process. DevPartner also saves session data for secondary processes, such as an ASP.NET worker process (`w3wp or aspnet_wp`). When the primary process stops, DevPartner stops data collection and generates a final correlated session file that contains both client data (for IE) and server data (for IIS and ASP.NET) worker processes. You can also take a snapshot of the server process alone by selecting the process in the Session Control toolbar.

In most cases the client and server processes are terminated by user action. However, the ASP.NET worker process can also shut down automatically during profiling. This can occur if you have edited the `processModel Attributes` section of the **`machine.config`** file on the system on which the process runs in one of the following ways:

- ◆ Changed the value of the `requestLimit` or `requestQueueLimit` attribute from “Infinite” to a value low enough to cause the process to be shut down during the session
- ◆ Changed the value of the `timeout` or `idleTimeout` attribute from Infinite to a value low enough to cause the process to be shut down during the session
- ◆ Changed the value of the `memoryLimit` attribute to a percentage low enough to cause the process to recycle during the session

When the process is shut down, DevPartner takes a final snapshot and generates a session file. DevPartner handles the session file in one of the following ways:

- ◆ If the ASP.NET worker process is the selected process in the Session Control toolbar, DevPartner opens the session file in Visual Studio and adds it to the solution. This action is repeated for each instance of the ASP.NET worker process that is spawned and terminated.

- If the ASP.NET worker process is not the selected process, the session file is cached. When the IE client process stops, or when a snapshot of the IE process is taken, DevPartner creates a session file for IE, and a correlated session file that includes data for IE, IIS, and all instances of the ASP.NET worker process spawned and terminated up to that point.

When the analysis session has ended, DevPartner continues to display the File Save dialog box or automatically save session files for instances of the ASP.NET worker process that are spawned and terminated.

To avoid generation of extra session files due to frequent termination of the ASP.NET worker process, you can edit the **`machine.config`** file and set the limiting attribute to a value high enough to prevent premature termination of the process.

Note: Always make a backup copy before editing the **`machine.config`** file.

Collecting Data from Classic Web Script Applications

When you run a classic Web script application with DevPartner coverage analysis enabled, DevPartner gathers data for HTML files and JScript and VBScript source files. If the scripting languages invoke in-process or out-of-process components, such as COM objects, DevPartner can collect data for these as well.

Instrumentation for the scripting languages occurs at run-time, just as it does for managed .NET languages. However you do need to instrument any unmanaged code components, such as COM objects, that you want monitored.

The following procedure is unique to classic Web script applications. To collect data for Web Forms, XML Web services, and ASP.NET applications you develop in Visual Studio, run the application just as you would run any other managed application.

To collect data for a classic Web script application, choose **Start > Programs > Micro Focus > DevPartner Studio > Utilities > Web Script Coverage.**

IE starts with DevPartner Coverage Analysis loaded. In addition to IE, a Session Control toolbar appears, which you can use to control data collection.

In the DevPartner-enabled instance of IE, open the HTML page or Web application for which you want to collect coverage data and exercise the application. Optionally, use the Session Control toolbar to focus data collection as the application executes.

Exit IE or, if using the Session Controls, execute a **Stop** action. The Save Session dialog box displays and the session file is automatically saved.

Web Service Requirements

For DevPartner coverage analysis to detect a Web service, the service must meet at least one of the following requirements:

- The Web service must be derived from the `System.Web.Services.WebService` base class.
- The Web service must contain the `WebService` attribute.

For DevPartner coverage analysis to detect a Web method, the method must contain the `WebMethod` attribute.

Deleting Temporary Files from NMSource

While analyzing scripts for coverage under IE or IIS, DevPartner creates an **`NMSource`** folder to hold temporary copies of the script source. This source is displayed in the Source tab of the Session window when you are analyzing session data.

Because this source may be needed at any time, DevPartner does not delete files from **`NMSource`**. The size of this folder can grow quickly, particularly when you are analyzing server programs under IIS.

You should regularly review the source files in the **`NMSource`** folder and delete any related to projects that are no longer active. **`NMSource`** is located in the **`\Program files\Internet Explorer`** folder.

Configuring IIS for Data Collection

To collect coverage data for IIS/ASP.NET applications running on a remote server, set the following configuration options.

Note: If IIS runs on a remote server, you must install DevPartner (and a Remote Server license) on that system and set the options described below on the remote system.

Script Debugging

You can set the following options in the Default Web Site Properties, or in the Web Application Properties for a specific application, of the IIS manager. The following options apply to IIS 5.0 or 6.0.

On the Home Directory or Directory tab, click **Configuration**. On the **Application Debugging** tab, set the **Debugging Flags** to:

- Enable ASP server-side script debugging
- Enable ASP client-side script debugging

Host Process Settings

If your Web application runs in the `dllhost` process, you may need to change the Application Protection options to enable DevPartner to collect coverage analysis data. You can set these options in the Default Web Site Properties, or in the WebApplication Properties for a specific application, of the IIS manager. The following options apply to IIS 5.0 or 6.0.

On the Home Directory or Directory tab, in the Application Settings section, set the Application Protection to one of the following:

- Low (IIS Process) Your application runs in the `inetinfo` process. DevPartner restarts IIS when you enable data collection and collects data from this process as your application runs.
- High (Isolated) Your application runs as a separate instance of `dllhost`. DevPartner recognizes the new process and collects data as your application runs.

When you have finished collecting data, restart IIS to remove DevPartner data collection from the process.

Configuring Internet Explorer for Coverage Analysis

To collect coverage analysis data from IE, select **Tools > Internet Options**. On the **Advanced** tab, set **Disable script debugging (Internet Explorer)** to OFF and set **Disable script debugging (Other)** to OFF.

Collecting Data from a Service

To run a coverage analysis session for a service, use **`DPAnalysis.exe`**. With **`DPAnalysis.exe`**, you can run sessions directly from the command line or through an XML configuration file.

Collecting Data from COM and COM+ Applications

You can collect data for an application that makes calls to COM or DCOM components with DevPartner.

If you profile an application that uses a mix of unmanaged COM and .NET objects (COM+), DevPartner collects line-level data for .NET portions of the application. DevPartner collects line-level data for unmanaged code components if they have been instrumented with DevPartner Native C/C++ Instrumentation. DevPartner can also collect line-level data for your unmanaged COM objects, if you first instrument them for coverage data collection. You can do this by building the project with instrumentation for coverage analysis in Visual Studio.

If profiling a C++ object, or any unmanaged code component that has not been instrumented, DevPartner collects only method-level data based on COM interfaces and DLL exports.

Merging Session Data

When testing an application using DevPartner, it is unlikely all application code is tested in one session. It is important to be able to gather coverage data collected in several sessions and analyze your total coverage statistics. To accumulate coverage data, merge the session files. Merging is the process of accumulating data from multiple sessions into a single file.

Files that contain merged session data are called merge files (**`.dpmrg`**). DevPartner can associate many merge files with a single project. DevPartner saves merge files as part of the active solution. They appear in the DevPartner Studio virtual folder in Solution Explorer.

You cannot merge correlated session files or Web Script session files produced from running IE. You can merge server-side session files from IIS.

To create a merge file, select **DevPartner > Merge Coverage Files** to create a new merge file or add data to an existing merge file. Merge files can also be created automatically, as described in “Merge Settings” on page 128.

When you merge session data, DevPartner:

◆ Maintains a record of all the images and methods that were loaded in any of the contributing session or merge files.

- Compares percent covered values and returns the superset of the data. For example, if you merge a session with 30% methods covered and a session with 20% methods covered, you may have not reached 50% coverage. There are likely parts of the code that were executed in both sessions.
- Uses data from the session or merge file that ran the latest image to determine if the methods and images are new, changed, or removed. DevPartner uses the time stamps of the images to determine the latest image.
- Calculates percent volatility values for each source and image. Percent volatility represents the percent of methods that changed in your code between sessions. It demonstrates your code stability.
- Maintains information about the files involved in the merge, when the merge occurred, and who performed the merge.

Reviewing Merge Data

DevPartner displays merge data in the Merge Data window. The Merge Data window contains the filter and Merge Data panes. The Merge Data pane contains the **Method List**, **Source**, **Merge History**, and **Merge Summary** tabs.

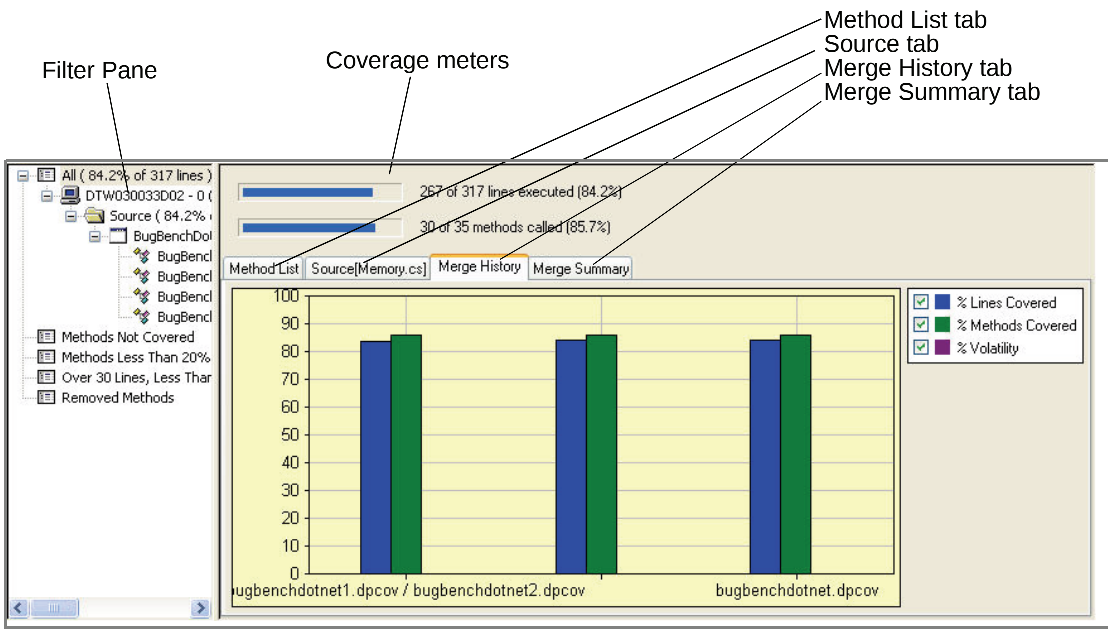


Figure 4-8. Merge Data window

- The **Method List** tab uses the State column in merge files. DevPartner uses the State column to distinguish methods that are new, changed, or removed between sessions.
- The **Merge History** tab displays a graphical representation of the progression of the **% Lines Covered**, **% Methods Covered**, and **% Volatility** values for the current merge file.
  - **% Lines Covered** is the percentage of lines in your source code that were executed.
  - **% Methods Covered** is the percentage of methods in your source code that were called.

  - ◇ **% Volatility** is the percentage of methods whose source code has changed since the last merge.

  If you have performed less than five merges in a merge file, DevPartner displays the **Merge History** tab as a bar chart. If you have performed five or more merges in a merge file, DevPartner displays the **Merge History** tab as a line chart.

  Hold your cursor over a point on the graph to see specific data for that merge.

  To show or hide a bar or line, choose or clear the check box in the key

- The **Merge Summary** tab displays summary information about the sessions and merge files that were merged into the file. It also contains information about each of the instrumented images used during the sessions, including the **Percent Volatility** for each image.

Note that if the source files have changed, merging coverage session files affects the synchronization of the method data that appears on the Method List tab and the line data that appears on the Source tab.

Merge States

If you change your code, DevPartner tracks those changes and adjusts the coverage data accordingly. It uses merge states to distinguish between changed, new, and removed methods and images. DevPartner displays information about these states in the **State** column on the **Method List**.

Methods

A method’s state can be new, changed, removed, or unchanged. The **State** column indicates new and changed methods; a blank entry in the **State** column indicates that the method has not changed.

Removed methods display in the **Removed Methods** filter. They are not used to calculate coverage statistics.

DevPartner does not distinguish between major and minor code changes. For example, when you make a change to a method that changes the number of lines in the method (for example, add or remove a comment), DevPartner marks the method as **Changed**. When you merge sessions that used executable files with different optimization options, DevPartner interprets this difference as a change and might mark some methods as **Changed**.

Images

Images can be loaded in one session and not in another. When an image is not loaded, DevPartner cannot determine what methods are in the image, or compare the image and its methods to find changes in relation to another session.

An image’s state can be new, activated, or inactive. Activated images are images that were present in another session in the merge file and have been reloaded. An inactive state can result from several conditions.

- DevPartner marks an image as inactive if the image has been removed.

Tip: To quickly determine the last session file you merged, examine the Merge History on the merge file **Merge Summary** tab.

- DevPartner marks an unmanaged code image as inactive any time it is not loaded. For example, if your application uses an unmanaged DLL but you do not load it during a session, when you merge that session with an earlier session that did load the DLL, DevPartner marks it as inactive. To obtain a complete coverage picture for an application that includes both unmanaged and managed code projects, make sure you run the unmanaged code portions of the application in the final coverage session you add to the merge file.
- DevPartner marks an unmanaged code image as inactive if the image was excluded from coverage data collection using the Exclude option, described on page 117.
- In managed applications, DevPartner marks an assembly as inactive only if the assembly (and all references to it) are removed from the application.

Inactive images are displayed in the Inactive Source filter in the filter pane. They are not used to calculate coverage statistics. DevPartner displays a value of 0% for the **Inactive Source** filter. When the **Inactive Source** filter is expanded, DevPartner shows coverage values for the individual inactive images. These values reflect coverage data for the sessions in which the images were active.

ASP.NET Modules in Merge Files

When you run a coverage session, DevPartner uses a repeatable algorithm to generate names for **`.aspx`** files compiled into an assembly. Because the algorithm is repeatable, DevPartner assigns the same name each time the assembly is registered. This feature provides a consistent name for each assembly, allowing you to accurately track changes for the assembly.

This naming operation takes place only when you run a coverage session. The default Visual Studio behavior remains unchanged when you build or rebuild a project that includes an **`.aspx`** file. Visual Studio assigns a randomly generated eight-character name to each **`.aspx`** file. When you edit the **`.aspx`** file and rebuild the assembly, Visual Studio assigns a new random eight-character name.

Merge Settings

When you generate session files, you control the default merge behavior by setting the merge property for the solution.

To set the merge property, select the solution in the Visual Studio Solution Explorer and display the Visual Studio Properties window. Choose a property under **Automatically Merge Session Files** in the **DevPartner Coverage, Memory and Performance Analysis** properties.

- If you want to selectively accumulate coverage data and be prompted to merge sessions you did not merge, use the **Ask me if I would like to merge it** setting.
- If you want to selectively accumulate coverage data and not be prompted about sessions you did not merge, use **Close without prompting**.
- If you want to accumulate coverage data in a merge file automatically for every session, use **Merge it automatically**.

Exporting Coverage Data

You can export coverage data in XML format or in CSV format. Exporting data in XML or CSV format facilitates using your own or third-party software to analyze the data, integrate the data with data produced by other tools, and archive the data in a data warehouse.

- You can export DevPartner coverage session files (with the **`.dpcov`** extension) and merged coverage files (with the **`.dpmrg`** extension) to XML format. When a saved coverage session file is open, the **Export DevPartner Data** command is available on the **File** menu. Refer to "Exporting Analysis Data to XML" on page 311 for information about exporting in XML format.

  You can also export data from the command line, as described in "Exporting Analysis Data to XML" on page 311.

- You can export Method List data to a comma-delimited (CSV) text file. Click the **Method List** tab, display the columns you want to export, right-click in the Method List and choose **Export Method List** from the context menu. You can open the comma-delimited text file in Microsoft Excel or another spreadsheet application.

Controlling Data Collection

DevPartner gives you three ways to control when coverage data is collected during the use of your application:

- You can use the session control toolbar to interactively control data collection as your program runs.
- You can use a session control file to assign session control actions to specific methods in your application modules.
- You can use the Session Control API to control data collection in your program.

Using the session control toolbar or Session Control API allows you to control data collection anywhere within a method. Using a session control file allows you to control collection only at the entrance to or exit from a method.

Using a session control file and using the Session Control API are described in "Analysis Session Controls" on page 303.

Analyzing from the Command Line

To automate data collection or run analysis sessions from the command line, use **`DPAnalysis.exe`**, the DevPartner command-line executable. For information on using **`DPAnalysis.exe`**, refer to "Starting Analysis from the Command Line" on page 285.

Using the Coverage Analysis Viewer

DevPartner Studio provides a lightweight Coverage Analysis Viewer for analyzing coverage session files independently of Visual Studio. To launch the viewer, do any of the following:

- On the Start menu, select **Programs > Micro Focus > DevPartner Studio > Coverage Analysis Viewer**.
- Double-click a `.dpcov` session file in Windows Explorer.
- Run a coverage analysis session using `DPAnalysis.exe` on the command line. DevPartner displays the session data in the Coverage Analysis Viewer.

What You Can Do in the Coverage Analysis Viewer

With a session file open, you can view, sort, save, or print coverage session data. In addition, you can:

- View the source code for a method
- Sort the data on the **Method List** tab
- Export the contents of the file as XML
- Export the contents of the Method List in CSV format

What you Cannot Do in the Coverage Analysis Viewer

- Instrument an unmanaged application for coverage analysis
- Start a coverage session
- Add files to a Visual Studio solution

Session files generated outside of Visual Studio are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

Integration with DevPartner Error Detection

You can use DevPartner error detection with coverage analysis to collect coverage data and check for errors during the same session when you run your managed application or unmanaged C/C++ application. You must instrument unmanaged C/C++ applications for Error Detection and Coverage with the Native C/C++ Instrumentation Manager before collecting data.

See “Error Detection” on page 23 for more information about error detection with DevPartner.

Submitting Data to Visual Studio Team System

DevPartner Studio supports Microsoft Visual Studio Team System if the Team Explorer client is installed and a Team Foundation Server connection is available. Refer to “Visual Studio Team System Support” on page 20 for general information about Team System support.

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected item. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected item. For a coverage analysis session file, access work item submission in for a method selected in the Method List tab. When submitting a work item, DevPartner populates the **Work Item** form with data from the visible columns in the **Methods List** tab. To change the method data you submit in the **Work Item**, change the columns displayed in the **Method List**.

Chapter 5

Finding Memory Problems

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with memory analysis. The second section provides reference information for an in-depth understanding of how to use DevPartner memory analysis.

Refer to the DevPartner online help for additional task-oriented information about memory analysis.

What is Memory Analysis?

The DevPartner memory analysis feature enables you to analyze memory allocation in your managed Visual Studio application.

DevPartner memory analysis presents memory data in context, enabling you to navigate chains of object references and calling sequences of the methods in your code. This provides an in-depth view of how your program uses memory and the critical information that you need to optimize memory use.

When you run your 32-bit or 64-bit application under memory analysis, DevPartner shows you the amount of memory used by an object or class, tracks the references that hold an object in memory, and identifies the lines of source code within a method that are responsible for allocating the memory.

DevPartner memory analysis includes three analysis types: Memory Leaks, Temporary Objects, and RAM Footprint. You can perform all three types of memory analysis in a single memory analysis session.

Each analysis type contains a real-time graph, a dynamically updated class list, and several session controls that enable you to control data collection and other memory-related events, such as forcing a garbage collection on the active process and creating a detailed view of the heap.

Note: The DevPartner memory analysis feature analyzes managed code only, and is therefore not supported in the DevPartner for Visual C++ Bounds-Checker Suite.

Because memory analysis is integrated into Visual Studio, you can use it to test applications as you develop them. You can also run memory analysis sessions from the command line, or as part of an automated test scenario, by using the DevPartner command-line executable **`DPAnalysis.exe`** with traditional command-line switches or an XML configuration file. For information, see “Starting Analysis from the Command Line” on page 285.

Using Memory Analysis Out of the Box

The following Ready, Set, Go procedure introduces you to using one of the three DevPartner Studio memory analysis features: Memory Leaks analysis.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject described in a shaded box, read the additional text following the box.

Analyzing an application with DevPartner Studio does not require elevated system privileges. The system privileges you use to create and debug your application are sufficient for DevPartner to analyze the application.

Ready: Consider What You Want to Analyze

Does your application performance slow down over time or when you perform certain operations? Does your application perform poorly under load conditions or when other applications are running? If you see any of these symptoms in your application you may be experiencing memory-related issues.

DevPartner memory analysis collects data only from managed applications. In order to collect memory analysis data for your application, the solution must contain at least one managed code project (for example, C#, Visual Basic, or managed C++). It must also include a startup project. If the solution includes multiple startup projects, DevPartner prompts you to choose a startup project for the session.

The following procedure assumes:

- You are working in a supported release of Visual Studio.
- You are testing a single-process, managed application.
- You can build and run your application.
- Your solution contains at least one managed code project.
- Your solution includes a startup project.

See “DevPartner Studio Supported Project Types” on page 267 for a comprehensive list of supported project types for DevPartner memory analysis.

The amount of memory consumed by your application has a major impact on how well the application performs. The larger the amount of memory allocated, the more likely the application runs slowly and scale poorly.

Leaked memory — the allocation of memory that is not reclaimed — can bloat your application’s RAM footprint. Automatic garbage collection relieves you of the responsibility to explicitly free the objects that you create, so memory is not “leaked” in the classic C++ sense, but it is still possible to retain references to objects that the program never uses.

As long as a reference to an object exists, the referenced object is considered to be a **live object** by the garbage collector; a live object cannot be collected. This condition, like leaked memory in C++, is undesirable. Such references can be difficult to track down and that is where memory analysis helps you.

This procedure assumes a single process application, but you can use DevPartner Studio to analyze complex, multi-process applications. Refer to “Collecting Data from Multiple Processes” on page 174 for additional information on how to profile multi-process applications.

Set: Properties and Options

There are a minimal set of configuration settings specific to memory analysis sessions.

For this procedure, you can use the default DevPartner properties and options. No additional set-up is required.

If you find that your application slows down too much while running memory analysis, you may be able to improve performance by excluding system objects from the analysis. See “Setting Properties and Options” on page 142 for details on changing the **Track System Objects** setting and other memory analysis settings.

Go: Collect Memory Analysis Data

Before starting a Memory Leaks analysis, it is useful to understand the workflow of the analysis.

Clicking **Start/Stop** marks the beginning and end of a tracking period for new memory allocations, excluding all other memory allocations by the application.

When you click **View Memory Leaks** some or all objects that were allocated during the tracking period are done with their tasks and are ready to be garbage collected.

Memory analysis analyzes all of the allocations collected during the tracking period and identifies leaks as objects that still have live references and cannot be collected.

View Memory Leaks forces a garbage collection and creates a session file to display these leaked objects in several graphic and list views. In the scenario depicted in Figure 5-1, Memory Leaks Analysis workflow time line, on page 134, the Memory Leaks session file would contain two leaked objects B and C that survived garbage collection. From the data, decide which leaked objects are expected and which ones are real leaks.

Garbage collection can be a system garbage collection or user-initiated by selecting either the **Garbage Collection** icon or the **View Memory Leaks** icon.

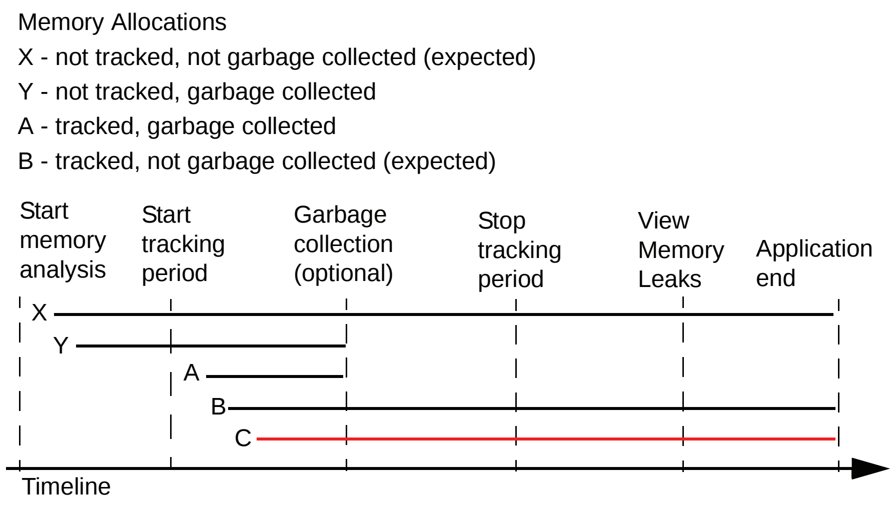
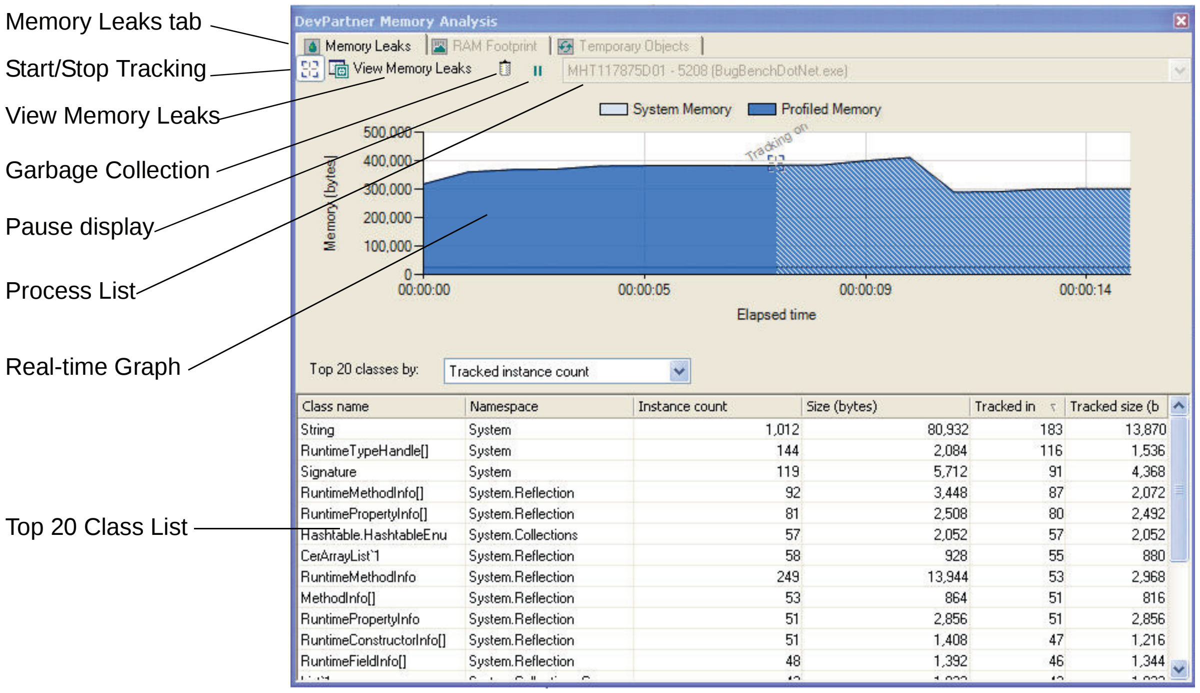

Memory Allocations
X - not tracked, not garbage collected (expected)
Y - not tracked, garbage collected
A - tracked, garbage collected
B - tracked, not garbage collected (expected)
Start memory analysis
Start tracking period
Garbage collection (optional)
Stop tracking period
View Memory Leaks
Application end
X
Y
A
B
C
Timeline

Figure 5-1. Memory Leaks Analysis workflow time line

You are now ready to perform a Memory Leaks analysis.

1 In Visual Studio, open the solution associated with your application.

2 Choose **DevPartner > Start Without Debugging with Memory Analysis**. Wait for the application to start and for the DevPartner Memory Analysis window to display the Session Control Window.



Top 20 classes by: Tracked instance count

Class name	Namespace	Instance count	Size (bytes)	Tracked in	Tracked size (b
String	System	1,012	80,932	183	13,870
RuntimeTypeHandle[]	System	144	2,084	116	1,536
Signature	System	119	5,712	91	4,368
RuntimeMethodInfo[]	System.Reflection	92	3,448	87	2,072
RuntimePropertyInfo[]	System.Reflection	81	2,508	80	2,492
Hashtable.HashtableEnu	System.Collections	57	2,052	57	2,052
CerArrayList`1	System.Reflection	58	928	55	880
RuntimeMethodInfo	System.Reflection	249	13,944	53	2,968
MethodInfo[]	System.Reflection	53	864	51	816
RuntimePropertyInfo	System.Reflection	51	2,856	51	2,856
RuntimeConstructorInfo[]	System.Reflection	51	1,408	47	1,216
RuntimeFieldInfo[]	System.Reflection	48	1,392	46	1,344

Figure 5-2. Memory Analysis Session Control window

3 Click the **Memory Leaks** tab.

4 Warm up the application by exercising the features that you plan to test. Warming up the application eliminates initialization allocations from the tracking period.

5 Click **Start/Stop** to start tracking new memory allocations and exclude previous memory allocations.

6 Exercise the application feature that you are collecting data from and run it through a complete cycle, but do not stop the application.

For this procedure, limit the tracking period to exercising a single feature within your application. This reduces the complexity of the session data and improves performance.

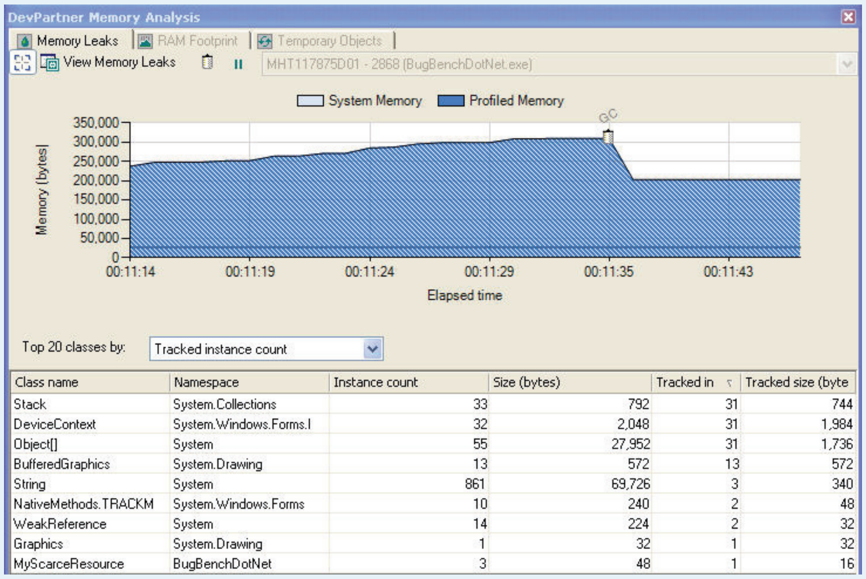
Patterns that appear in the session control window graph as you exercise the application provide the initial diagnosis of how the application is using memory. Different memory problems show characteristic patterns, so the real-time graph provides important clue about the existence and nature of a problem. This helps determine the type of memory analysis to perform.

For example, a steadily rising pattern that does not return to baseline or respond as expected to garbage collection may indicate leaked memory.

For in-depth information on other characteristic patterns in the real-time graph, see “Using the Session Control Window in Memory Analysis” on page 145.

For a complex application, the number of classes displayed in the list may be large. Right-click in the list and choose **Show Top 20 Classes with Source** from the context menu to limit the class list to your application’s source code methods.

7 Click **Force Garbage Collection** to issue a garbage collection on the active process.



Top 20 classes by: Tracked instance count

Class name	Namespace	Instance count	Size (bytes)	Tracked in	Tracked size (byte
Stack	System.Collections	33	792	31	744
DeviceContext	System.Windows.Forms.I	32	2,048	31	1,984
Object[]	System	55	27,952	31	1,736
BufferedGraphics	System.Drawing	13	572	13	572
String	System	861	69,726	3	340
NativeMethods.TRACKM	System.Windows.Forms	10	240	2	48
WeakReference	System	14	224	2	32
Graphics	System.Drawing	1	32	1	32
MyScarceResource	BugBenchDotNet	3	48	1	16

Figure 5-3. Session Control Window after garbage collection

8 Examine the list in the lower half of the Session Control window. The list shows information about the classes that contain objects with active references after the garbage collection.

9 Click **Start/Stop** to stop the tracking period and exclude new memory allocations. If your application is active in the background, the values contained in the list may change, but the tracked instances do not increase.

10 Click **View Memory Leaks** to force another garbage collection and create a Memory Leaks analysis session file.

11 Close the application.

Memory analysis automatically creates a second file, a Temporary Objects analysis session file, which is in focus in Visual Studio.

12 Click the **Leak...Analysis Snap.dpmem** tab to bring the Memory Leaks snapshot session file into focus.

Analyze the Memory Analysis Data

The Memory Leaks analysis session file records all objects allocated during the tracking period that had an active reference at the time you clicked **View Memory Leaks**. Use the session file to examine objects, methods, and critical execution paths to help determine why objects are still in memory.

Memory Leaks analysis helps identify unneeded objects in the context of the application and to find the best point in the reference chain to remove the references that keep these objects in memory.

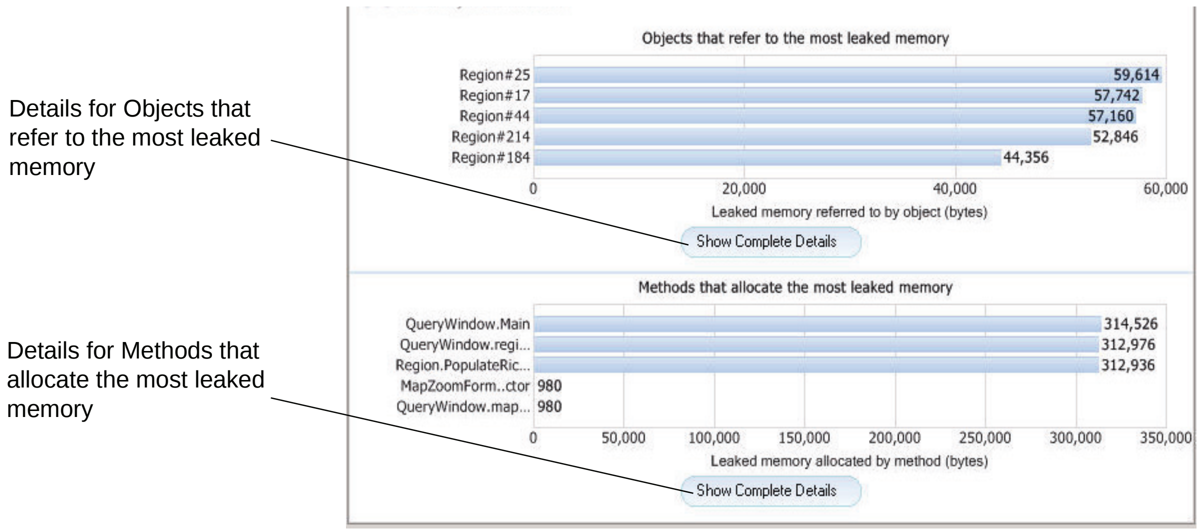


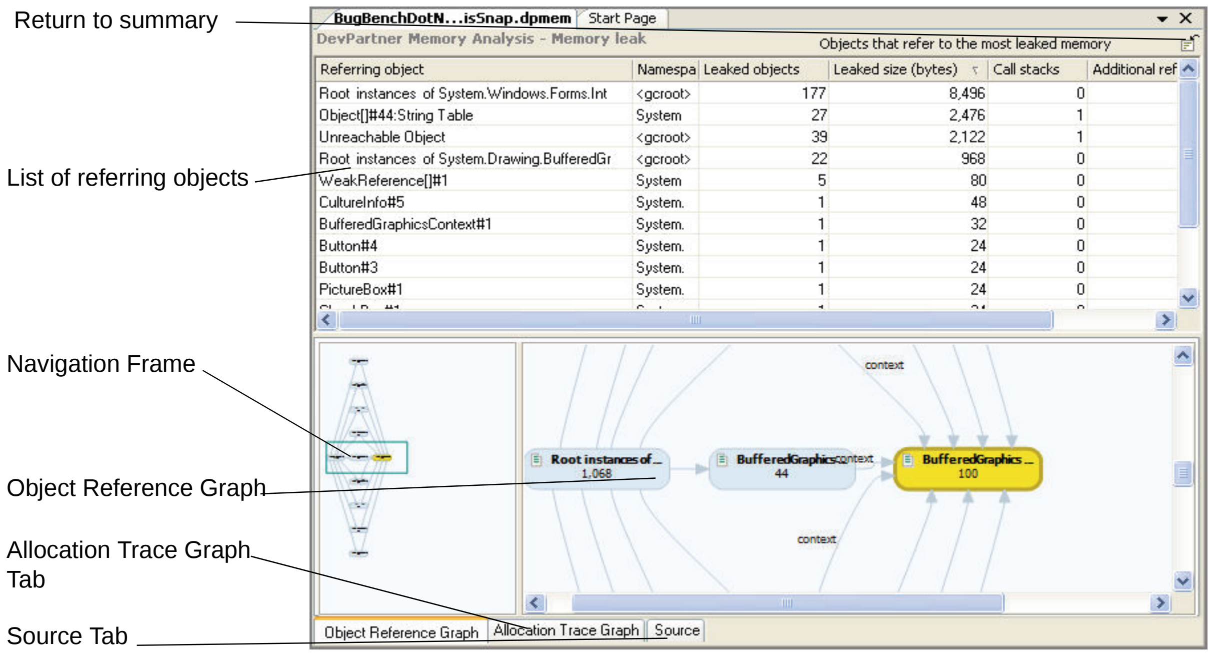
Figure 5-4. DevPartner Memory Analysis - Memory Leaks Analysis summary

The **DevPartner Memory Analysis - Memory leaks analysis** summary contains bar charts for **Objects that refer to the most leaked memory** and **Methods that allocate the most leaked memory**.

The remainder of this procedure guides you through inspecting details in both summaries.

1 Click **Show Complete Details** below **Objects that refer to the most leaked memory**.

Refer to "Using the Object Reference Graph" on page 149 for in-depth information on working with memory analysis session files.



Referring object	Namespa	Leaked objects	Leaked size (bytes)	Call stacks	Additional ref
Root instances of System.Windows.Forms.Int	<gcroot>	177	8,496	0	
Object[]#44:String Table	System	27	2,476	1	
Unreachable Object	<gcroot>	39	2,122	1	
Root instances of System.Drawing.BufferedGr	<gcroot>	22	968	0	
WeakReference[]#1	System	5	80	0	
CultureInfo#5	System.	1	48	0	
BufferedGraphicsContext#1	System.	1	32	0	
Button#4	System.	1	24	0	
Button#3	System.	1	24	0	
PictureBox#1	System.	1	24	0	

Figure 5-5. DevPartner Memory Analysis - memory leaks object reference details

2 Examine the DevPartner Memory Analysis - Memory Leaks view that displays a list of referring objects sorted by **Leaked size (byte)**. The **Referring object** responsible for the most leaked memory appears at the top of the list.

The tabs at the bottom of the window display an Object Reference Graph, Allocation Trace Graph, and a Source window.

3 Choose the **Object Reference Graph** tab and then select a referring object in the list at the top.

When you select a referring object from the list, the **Object Reference Graph** highlights the selected object.

4 In the **Object Reference Graph**, hover the mouse over an object node to get information about the leaked memory associated with that object.

5 Drag the **navigation frame** in the **overview pane** to focus on various parts of the **Object Reference Graph.**

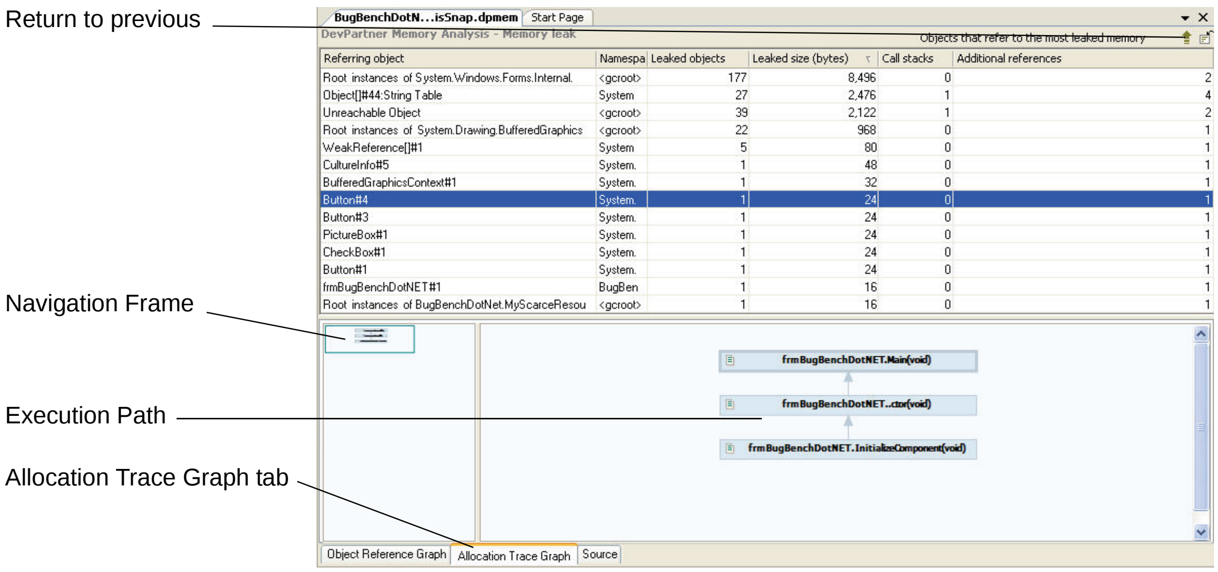
6 Right-click a **Referring object** in the list and choose **View leaked objects referenced by this object**. The default sort order is **Referenced size (bytes)**, which highlights the amount of memory that could be freed if the object was collected.

You should understand why the objects are still referenced and at this stage, you can decide where in the code you want to break the references (if needed).

If you need more information or deeper program understanding, use the tabbed views to examine object references, identify the execution paths that allocated the memory, and locate the lines responsible for source code.

- The **Object Reference** graph provides a graphical representation of objects and the related object references. The display depicts each object with related information such as memory used by itself and its children, or the percentage of memory used by the object.
- The **Allocation Trace** graph provides a graphical representation of the execution paths in your code. This gives you the context of where the object was allocated.
- The Source window displays the related source code for each object.

Consider the expensive object references and decide whether or not the application can be optimized by managing the object references differently.



Return to previous
Navigation Frame
Execution Path
Allocation Trace Graph tab
BugBenchDotN...isSnap.dpmem
Start Page
DevPartner Memory Analysis - Memory leak
Objects that refer to the most leaked memory
frmBugBenchDotNET.Main(void)
frmBugBenchDotNET..ctor(void)
frmBugBenchDotNET.InitializeComponent(void)
Object Reference Graph
Allocation Trace Graph
Source

Referring object	Namespa	Leaked objects	Leaked size (bytes)	Call stacks	Additional references
Root instances of System.Windows.Forms.Internal.	<gcroot>	177	8,496	0	2
Object[]#44:String Table	System	27	2,476	1	4
Unreachable Object	<gcroot>	39	2,122	1	2
Root instances of System.Drawing.BufferedGraphics	<gcroot>	22	968	0	1
WeakReference[]#1	System	5	80	0	1
CultureInfo#5	System.	1	48	0	1
BufferedGraphicsContext#1	System.	1	32	0	1
Button#4	System.	1	24	0	1
Button#3	System.	1	24	0	1
PictureBox#1	System.	1	24	0	1
CheckBox#1	System.	1	24	0	1
Button#1	System.	1	24	0	1
frmBugBenchDotNET#1	BugBen	1	16	0	1
Root instances of BugBenchDotNet.MyScarceResou	<gcroot>	1	16	0	1

Figure 5-6. DevPartner Memory Analysis - Allocation Trace graph

7 When you decide that you want to make changes, select the **Allocation Trace Graph** tab to see the execution paths that created the object and allocated the memory.

8 Choose the **Source** tab and select an object in the Object List. Notice the source code reference change for each object that you select.

9 Right-click an object in the list and choose **Edit source** to display the related source code line in the Visual Studio editor.

There is no editable source code available for system objects.

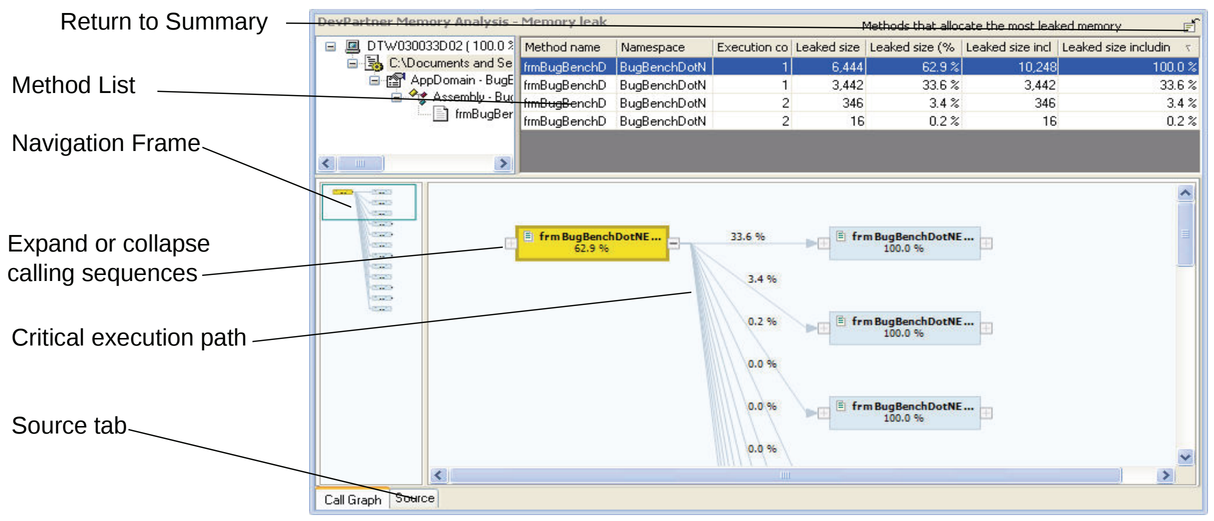
10 Close the Visual Studio editor.

For an in-depth example, see "Objects that Refer to the Most Leaked Memory" on page 159. For various techniques to access source code, see "Navigating the Source Tab" on page 152.

After identifying source code that correlates to an object reference, memory management beyond the object level to the inter-relationships between objects and object references can be seen. From here, begin making decisions on whether or not the object references can be managed differently to improve application performance.

Object reference management changes could involve using smaller objects, weak references, different sequencing of object references, or limiting the number of layers of abstraction.

For Web applications, awareness of a client-server relationship may allow you to capitalize on a garbage collection on the server when scalability is a focus.



Method name	Namespace	Execution co	Leaked size	Leaked size (%	Leaked size incl	Leaked size includin
frmBugBenchD	BugBenchDotN	1	6,444	62.9 %	10,248	100.0 %
frmBugBenchD	BugBenchDotN	1	3,442	33.6 %	3,442	33.6 %
frmBugBenchD	BugBenchDotN	2	346	3.4 %	346	3.4 %
frmBugBenchD	BugBenchDotN	2	16	0.2 %	16	0.2 %

Figure 5-7. DevPartner Memory Analysis - method list Call graph

11 Select the **DevPartner Memory Analysis - Memory leaks analysis session file** tab and click to return to the Summary page.

In addition to the Objects that refer to the most leaked memory, you can analyze the Methods that allocated the most leaked memory.

12 From the Summary page, choose **Show Complete Details** for Methods that allocate the most leaked memory.

The Method List displays source methods that allocated the most leaked memory.

13 Select a method in the Method List to display the **Call Graph**.

14 In the **Call Graph** window, hover the mouse over a method node or the line between method nodes. Compare leaked size contributed by the method node and its children.

The critical execution path is highlighted with a bold, gold-colored line.

15 Use the + and - controls to expand and collapse the calling sequences to various levels for method nodes.

16 In the list above, right-click on a method name and select **View Source** from the context menu.

17 In the list, right-click on a method name and choose **View Summary**.

For an in depth example, see "Methods that Allocate the Most Leaked Memory" on page 161.

Saving Session Files

When you have finished reviewing memory analysis data you can save the session files.

1 Close both session file windows in Visual Studio. DevPartner prompts you to save the session file.

2 Click **Ok** to save the file with the default file name and location.

DevPartner saves session files as part of the active solution. They appear in the DevPartner Studio virtual folder in Solution Explorer. Memory analysis session files take the **`.dpmem`** extension.

By default, DevPartner physically saves the session files in your project's output folder. DevPartner automatically increments the file name based on the contents of the default folder (for example, **`MyApp-TemporaryObjectSnap1.dpmem`**, **`MyApp-LeakAnalysisSnap1.dpmem`**, and so on). If you save session files to a location other than the default folder, you must manage the file naming and numbering.

For projects that do not have an output folder, such as a Visual Studio 2005 Web site project, DevPartner physically saves the files to the project folder.

Session files generated from the command line utility are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

The remainder of this chapter provides reference information and an in depth exploration of each DevPartner memory analysis feature: Memory Leaks, Temporary Objects, and RAM Footprint.

This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a memory analysis session, continue reading the rest of this chapter for additional information, or refer to the DevPartner online help for task-based information.

Memory Problems in Managed Visual Studio Applications

Managed Visual Studio applications benefit from a sophisticated memory management environment with garbage collection. Unlike unmanaged (native) C++, in which you explicitly free the memory that you allocate, the garbage collector frees memory once the object for which it was allocated is no longer in use, or more accurately, no longer “reachable” by the application.

Because of the built-in memory management in managed code projects, many developers assume that managed languages relieve them of the headaches traditionally associated with memory management. However, memory allocation and use in managed Visual Studio programs can still cause performance bottlenecks and resource depletion.

Does your application exhibit any of the following symptoms?

- ◇ Performance slows down over time
- ◇ Runs slowly, or slows down noticeably when you perform certain operations
- ◇ Performs poorly under load conditions
- ◇ Performs poorly when other applications are running

Any of these symptoms indicate that your application has a performance problem. Refer to the following lists of questions to better understand if the problem is related to memory use.

- Several application classes must load before the program executes a particular function and each application class uses memory.
  - ◇ Does your program load classes that are only related to performing current tasks?
  - ◇ How many instances of a particular class does your application create and are all instances needed?
- Object allocation also incurs memory use which may lead to performance problems.
  - ◇ Do you know if your program allocates too many objects, or allocates them efficiently?
  - ◇ Does the garbage collector clear the objects that your program allocates?
  - ◇ Are the objects being collected as expected, or do the objects remain in memory long past their usefulness?

How Memory Analysis Helps You

The DevPartner memory analysis feature provides a comprehensive view of memory use in your managed application. DevPartner provides three different types of memory analysis to help you isolate different kinds of memory-related problems. Regardless of which type of analysis you use, all types include the following features:

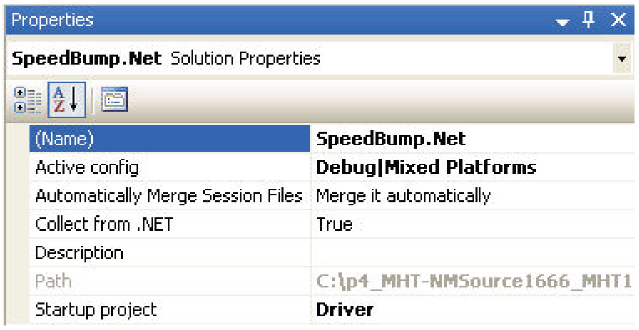
- **Real-time graph** — DevPartner presents a live view of memory use in your application while it runs. This view appears in the **Session Control Window**. You can see how much memory is being used by your application code (profiled code), system and other application code (excluded code), and how memory consumption compares to the memory reserved for the managed heap.
- **Dynamic list of classes** — DevPartner updates the list of profiled classes in real time while your application runs. This shows you the number of objects allocated and number of bytes used by each class, as your application runs.
- **Detailed heap views** — You can capture a snapshot of a detailed view of the managed heap at any time during program execution. DevPartner stores this data in a session file that you can then use to analyze memory problems in depth. DevPartner provides multiple ways to drill down into the session data, so you can see how your application uses memory and ultimately identify the methods or lines of code responsible for the most memory use.

Setting Properties and Options

Before beginning a memory analysis session, it is often useful to fine-tune data collection to include or omit certain types of information. Use Solution Properties, Project Properties, and DevPartner options to better focus your analysis session.

Solution Properties

To view properties that affect memory analysis at the solution level, select the solution in the Solution Explorer and press F4 to view the Properties Window.


Properties

SpeedBump.Net Solution Properties

(Name)	**SpeedBump.Net**
Active config	**Debug|Mixed Platforms**
Automatically Merge Session Files	Merge it automatically
Collect from .NET	True
Description	
Path	C:\p4_MHT-NMSource1666_MHT1
Startup project	**Driver**

Figure 5-8. Solution properties

The following Solution properties may affect memory analysis:

- **Collect from .NET -** Running your managed application with memory analysis overrides this property if it is set to **False**. Memory analysis always collects data from managed applications.
- **Startup project -** If your solution includes multiple projects, you can change the startup project. The **Project** properties for the startup project govern data collection for all projects active in the session.

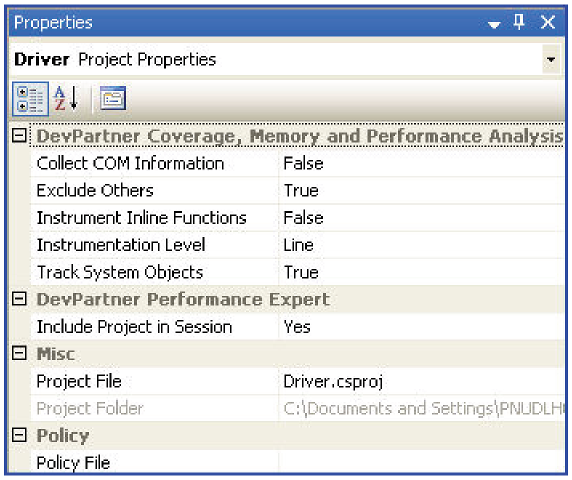
Note that your solution must include a startup project. If the solution contains multiple startup projects, DevPartner prompts you to choose a startup project for the session before analysis begins.

Only projects for which the **Action** in the **Common Properties** > **Startup Projects** page of the solution properties is set to **Start** are included in the prompt dialog. If the desired startup project does not appear in the prompt, open the solution properties page and set the **Action** for the project to **Start**. If you choose a new startup project for a subsequent session, review the properties for the new startup project to ensure the data collection options are correct.

Project Properties

To review project level properties, select a project in the Solution Explorer and review the properties that can be set for projects within the solution.

Changes that you make here affect coverage analysis, memory analysis, performance analysis, and Performance Expert.



Properties

Driver Project Properties

DevPartner Coverage, Memory and Performance Analysis	
Collect COM Information	False
Exclude Others	True
Instrument Inline Functions	False
Instrumentation Level	Line
Track System Objects	True
DevPartner Performance Expert	
Include Project in Session	Yes
Misc	
Project File	Driver.csproj
Project Folder	C:\Documents and Settings\PNUDLH
Policy	
Policy File	

Figure 5-9. Project properties

The following project-level property affects memory analysis:

- **Track System Object -** Set this property to **False** to ignore system or third-party object allocations when tracking allocated objects in memory analysis sessions.
  The default state, **True**, enables you to see memory allocations made by system or other non-profiled resources

Options

To review DevPartner option settings for memory analysis sessions, choose **DevPartner > Options > Analysis**.

- **Precision -** Choices are one, two, three, or four decimal places
- **Units -** Choices are Bytes, Kilobytes, or Megabytes
- The **Session Control File** option allows you to create a set of rules and actions to control the data that DevPartner collects as your application or module runs. Refer to "Creating a Session Control File Within Visual Studio" on page 303 for more information about session control files.

Other Visual Studio options, such as the **Environment > Fonts and Colors** options, also affect DevPartner features.

Starting a Memory Analysis Session

You may choose to run a memory analysis session with or without debugging. From the DevPartner menu, the only option is to start a memory analysis session without debugging.

After you open a project or solution, the selection to the right of the memory analysis icon allows you to start the session with or without debugging.

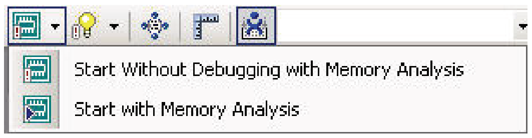

Start Without Debugging with Memory Analysis
Start with Memory Analysis

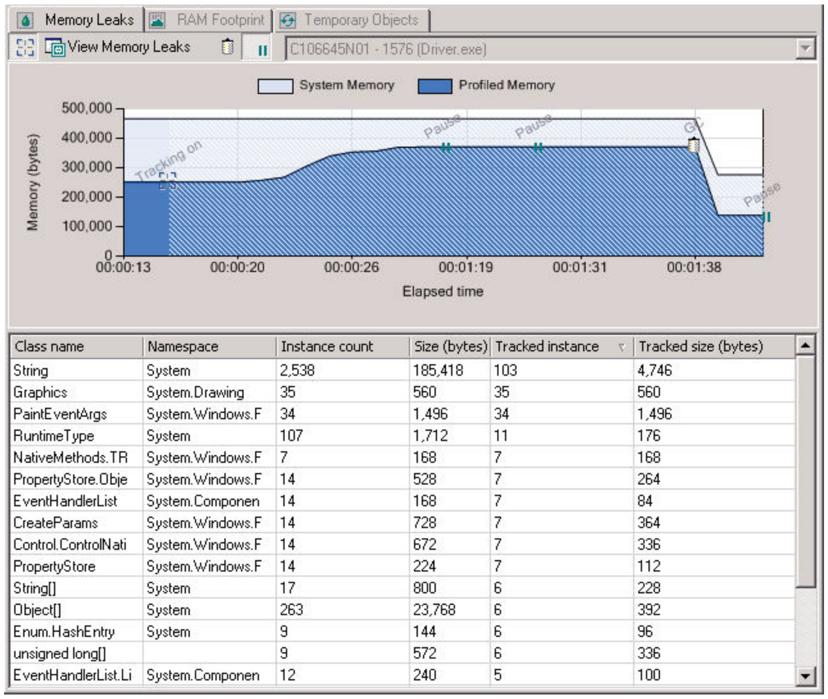
Figure 5-10. Memory Analysis Icon with option to start with or without debugging

Start Without Debugging with Memory Analysis is the default setting for memory analysis due to ease of understanding analysis results, and improved performance. You may instrument your code with break points and **Start with Memory Analysis with debugging** to isolate the performance of specific sections in your code.

An alternative to setting break points to isolate sections of your code is to use either the **`SessionControl.txt`** file or the Session Control API to perform memory analysis actions while your program runs. Refer to "Creating a Session Control File Within Visual Studio" on page 303 for more information about session control files.

Using the Session Control Window in Memory Analysis

When you start a new memory analysis session, DevPartner opens the **Se**ssion Control Window. Each tab in the Session Control Window corresponds to one of the types of memory problems you can analyze: Memory Leaks, Temporary Objects, and RAM Footprint. Each tab contains a view of the real-time graph, the dynamically updated class list, and several session controls that enable you to control data collection and other memory-related events, such as garbage collection. The data shown in the class list and the session controls available differ slightly, depending on the tab selected.



Class name	Namespace	Instance count	Size (bytes)	Tracked instance	Tracked size (bytes)
String	System	2,538	185,418	103	4,746
Graphics	System.Drawing	35	560	35	560
PaintEventArgs	System.Windows.F	34	1,496	34	1,496
RuntimeType	System	107	1,712	11	176
NativeMethods.TR	System.Windows.F	7	168	7	168
PropertyStore.Obje	System.Windows.F	14	528	7	264
EventHandlerList	System.Componen	14	168	7	84
CreateParams	System.Windows.F	14	728	7	364
Control.ControlNati	System.Windows.F	14	672	7	336
PropertyStore	System.Windows.F	14	224	7	112
String[]	System	17	800	6	228
Object[]	System	263	23,768	6	392
Enum.HashEntry	System	9	144	6	96
unsigned long[]		9	572	6	336
EventHandlerList.Li	System.Componen	12	240	5	100

Figure 5-11. DevPartner Memory Analysis Session Control window

Patterns in the Real-time Graph

Observe the real-time graph when you start a session. The pattern that appears in the graph as you exercise your application shows initial diagnosis of how your application is using memory. Different memory problems show characteristic patterns, so the real-time graph provides the first clue as to the existence of a problem and the nature of the problem. This helps determine what kind of memory analysis to perform.

- A steadily rising pattern that does not return to baseline or respond as expected to garbage collection may indicate leaked memory. Run Memory Leaks analysis.
- A pattern that does return to baseline but is characterized by periodic spikes in memory use indicates that your application is creating lots of objects as it runs. Run Temporary Objects analysis.
- If your application consistently consumes nearly all the reserved system memory in the managed heap, and the amount is large relative to the anticipated resources of your target users' systems, you may want to reduce the overall memory footprint of your application. Run RAM Footprint.

Dynamic Class List

The class list shows the 20 profiled classes that consume the most memory. The list is updated dynamically as your application runs under memory analysis. Use the class list to observe which classes are associated with increases in memory consumption or increases in object creation. Because the list is updated in real time, you may be able to spot potential problem areas as you exercise your application.

The following columns are available in the class list. Columns that indicate data is displayed in units present data in bytes, kilobytes, or megabytes, depending on the options that you set in **DevPartner Analysis Display Options**.

- To change the sort order of the class list, select a column heading in the **Top 20 Classes by** list.
- To limit the class list to classes for which source code is available, right-click anywhere in the list and choose **Show Top 20 Classes with Source** from the context menu.

When you display the class list with the **Show Top 20 Classes with Source** option, array classes appear in the list if the array element type is in the source code.

Table 5-1. Column Headings In the Dynamic Class List for Memory Analysis

Columns	Data Displayed
Class names	Name of the class
Namespace	Namespace of the class
Instance count	Number of objects of this class currently in memory
Size (units)	Amount of memory used by instances of this class. Default sort for Temporary Object and RAM Footprint analysis.
Tracked instance count (Visible in Memory Leaks analysis only)	Number of tracked objects of this class currently in memory. Default sort in Memory Leaks analysis.
Tracked size (units) (Visible in Memory Leaks analysis only)	Amount of memory used by all of the tracked objects of this class that are currently in memory.

Tracked objects are objects allocated after the user clicks Start Tracking and before the user toggles the selection to Stop Tracking.

DevPartner Memory Analysis Session Control Window

The Session Control window provides a number of ways to interactively control data collection and display.

Table 5-2. Memory Analysis Session Control window features

Session Control	Function
Process	Use the list of processes at the upper right of the tabbed area to choose a process to monitor. New processes (that are configured to be profiled) are added to the list as they begin to execute. The default selection is the start process.
Start/Stop Tracking ▣ (Memory Leaks only)	Starts or stops tracking memory allocations (toggle). When you click this button, the graph changes color to indicate the portion tracked.

Table 5-2. Memory Analysis Session Control window features

Session Control	Function
Clear All Memory (Temporary Objects only)	Clears all memory data collected to this point. Does not affect garbage collection.
Force Garbage Collection	Forces a garbage collection on the active process.
Pause Display	Pauses the display (toggle) but does not stop data collection. When Pause is clicked again, the graph display begins redrawing the current memory use activity.

View Session Results

As your application runs, you can capture a snapshot of memory use by clicking the appropriate **View...** button. This creates a session file that contains memory usage data. You can create as many session files as you need during a given run of your application. Capturing a snapshot does not stop data collection.

Table 5-3. Snapshot commands for Memory Analysis View Session Results

Snapshot Command	Function
View Memory Leaks	Forces a garbage collection on the active process and opens a session file that displays detailed memory leaks data.
View Temporary Objects	Creates a session file that displays detailed temporary objects data. Does not force a garbage collection on the active process.
View RAM Footprint	Forces a garbage collection and opens a session file displaying detailed RAM footprint data.

Unsaved session files open automatically in Visual Studio after they are created and all session files become part of the active solution when saved. They appear in the DevPartner Studio virtual folder in the Solution Explorer pane.

Session files first appear in the form of a Results Summary. Use the Results Summary to drill into the session data and locate problem areas in source code.

Session File Integration

When your application execution stops, DevPartner displays the results of the memory analysis sessions in a Session window in Visual Studio. DevPartner stores the collected data in a memory analysis session file, with a `.dpmem` extension.

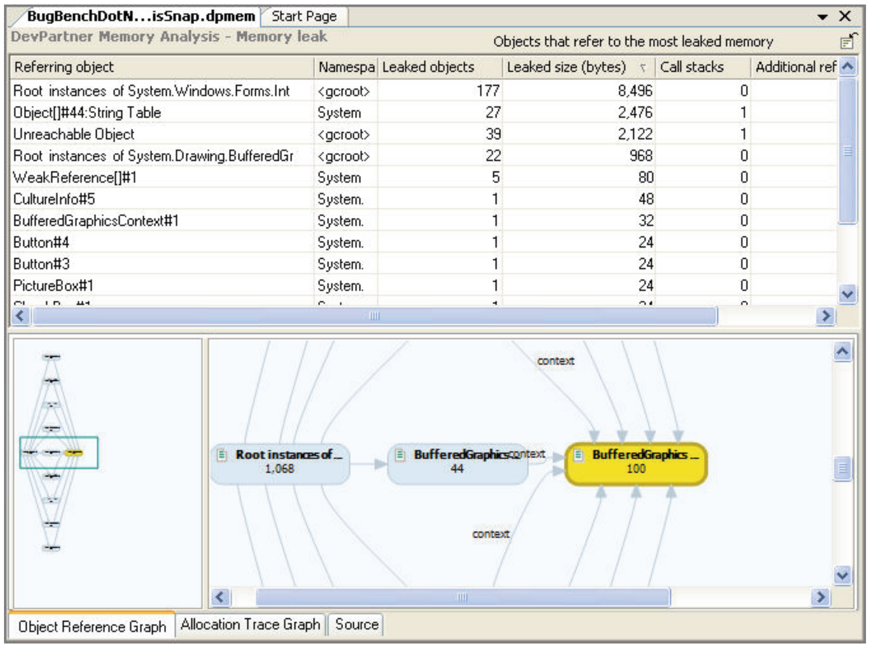
DevPartner automatically adds the session files to the DevPartner Studio folder that you can view in the Solution Explorer for the active solution. To review an existing memory analysis session file, double-click the file in Solution Explorer.

From the Session window, you can analyze results within the development environment. Drill down into the data to examine object references or the call relationships of the methods that allocated the objects, jump to the source code for a particular method, and open the source code for any method for editing in Visual Studio.

Using the Object Reference Graph

When analyzing objects that remain in memory, you want to understand what prevents them from being collected by the garbage collector. The Object Reference graph shows the complete chain of objects between the selected object and the garbage collection root or roots that are keeping the selected object alive.

The Object Reference graph does not show all referring objects, but those referring objects that point to a garbage collection root. For reasons of completeness, the graph also occasionally shows objects in the object reference path even if they are not on the shortest path to a garbage collection root.


BugBenchDotN...isSnap.dpmem
Start Page
DevPartner Memory Analysis - Memory leak
Objects that refer to the most leaked memory
context
Root instances of ... 1,068
BufferedGraphics context 44
BufferedGraphics ... 100
context
Object Reference Graph
Allocation Trace Graph
Source

Referring object	Namespa	Leaked objects	Leaked size (bytes)	Call stacks	Additional ref
Root instances of System.Windows.Forms.Int	<gcroot>	177	8,496	0	
Object[]#44:String Table	System	27	2,476	1	
Unreachable Object	<gcroot>	39	2,122	1	
Root instances of System.Drawing.BufferedGr	<gcroot>	22	968	0	
WeakReference[]#1	System	5	80	0	
CultureInfo#5	System.	1	48	0	
BufferedGraphicsContext#1	System.	1	32	0	
Button#4	System.	1	24	0	
Button#3	System.	1	24	0	
PictureBox#1	System.	1	24	0	

Figure 5-12. Memory Analysis Object Reference graph

The Object Reference graph automatically redraws when you select an object in the Object List.

The Object Reference graph consists of two frames:

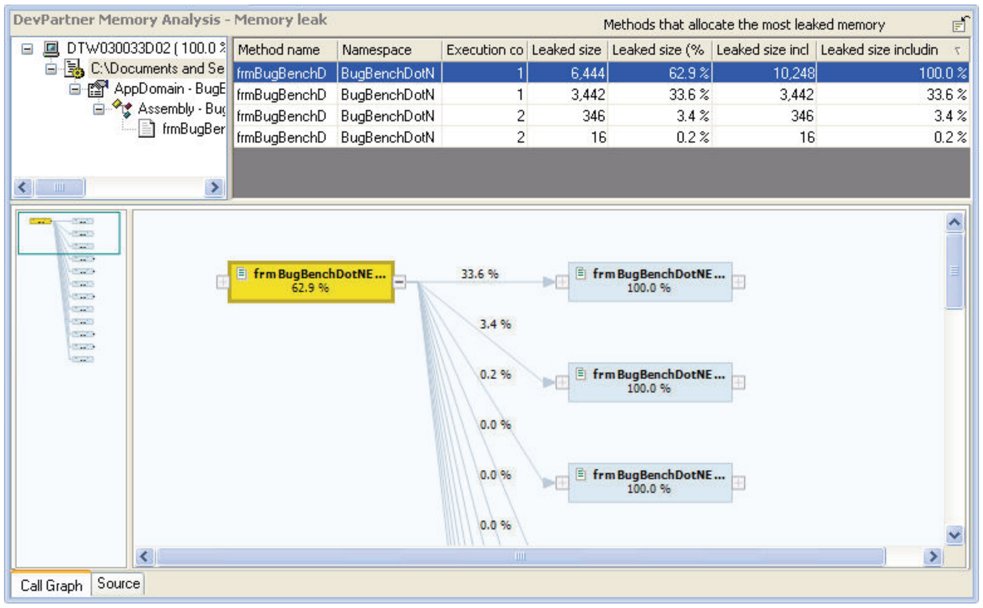
- The left frame presents an overview pane of the Object Reference graph. The overview pane contains a navigation frame that allows you to quickly locate and view different parts of a large graph.
- The right frame presents the object reference relationships for the object you selected in the Object List.

The node highlighted in yellow represents the selected object. Numeric data on the node indicates leaked size or referenced size, depending on context. Object reference paths are indicated by lines with arrows indicating the order of reference. Labels on the connecting lines indicate the member variable that holds the reference.

Using the Call Graph to Identify Execution Paths

The Call graph consists of two frames:

- The left frame shows an overview pane of the Call graph that is useful to navigate a large graph. As you move the **navigation frame** in the overview, the view in the right frame changes dynamically.
- The right frame shows the Call graph. Methods are shown as nodes. Links between nodes represent calling relationships. Expand nodes to view the order of program execution.


DevPartner Memory Analysis - Memory leak
Methods that allocate the most leaked memory
DTW030033D02 (100.0 %
C:\Documents and Se
AppDomain - BugE
Assembly - Bug
frmBugBer
frm BugBenchDotNE... 62.9 %
33.6 %
3.4 %
0.2 %
0.0 %
0.0 %
0.0 %
frm BugBenchDotNE... 100.0 %
frm BugBenchDotNE... 100.0 %
frm BugBenchDotNE... 100.0 %
Call Graph
Source

Method name	Namespace	Execution co	Leaked size	Leaked size (%	Leaked size incl	Leaked size includin
frmBugBenchD	BugBenchDotN	1	6,444	62.9 %	10,248	100.0 %
frmBugBenchD	BugBenchDotN	1	3,442	33.6 %	3,442	33.6 %
frmBugBenchD	BugBenchDotN	2	346	3.4 %	346	3.4 %
frmBugBenchD	BugBenchDotN	2	16	0.2 %	16	0.2 %

Figure 5-13. Memory Analysis Call graph

Call graphs are read left to right. The first node initially shown in the Call graph is the base node. This represents the method selected in the Method List. Nodes to the left of the selected node are called parent nodes. Nodes to the right of a node are called child nodes.

The upper half of the node shows the node name, which is the name of the function or method being displayed by the node. The bottom half shows the node value, which is a percentage value associated with the node. This value is the percentage of memory the node is using of the total memory being used by the node (and all of its child nodes).

The smaller rectangles on the left and right sides of the nodes are called links. They represent either a method call or invocation. The percentages on the lines tying nodes together are called link values and show a percentage value associated with the link. The link value shows the percentage of memory that child (and its children) are using of the total memory being used by its parent node.

Nodes that have no associated parent/child nodes are called “dead end nodes”. They represent the end of a path of execution, either at the start or end of an order of method calls.

To show the Method name list and associated Call graph for temporary objects, click **Show Complete Details** from the **Entry points that allocate the most memory** graph or the **Methods that use the most memory** graph.

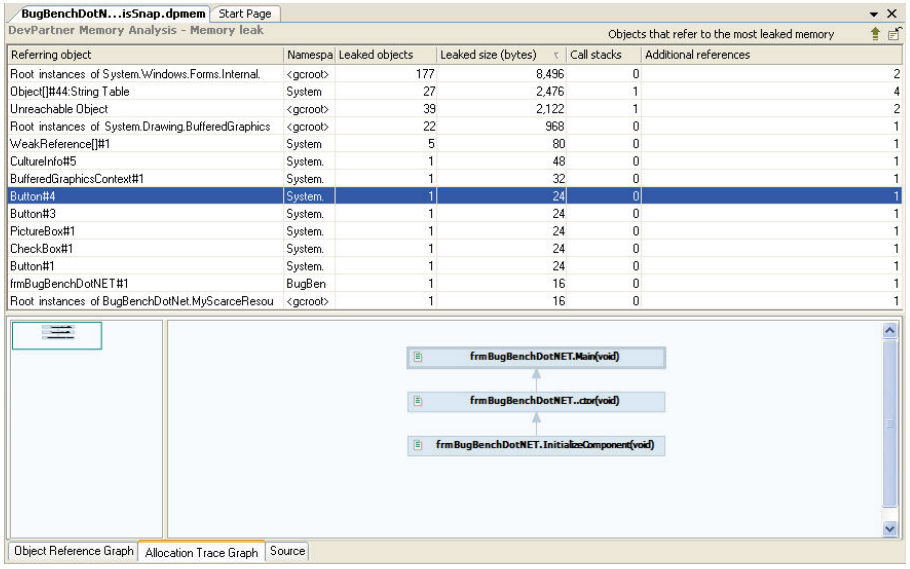
Once the Method name list and associated Call graph appears, you can display Call graphs for the methods in the Method name list by selecting a method in the Method name list. To view the source code for a method, select the node representing a method and click the **Source** tab at the bottom of the Call graph window.

Critical Paths

When you display a Call graph, DevPartner computes the critical path for the selected method and all of its children. The critical path is the sequence of child method calls that resulted in the largest cumulative memory allocation. The critical path is highlighted with a bold, gold-colored line.

Using the Allocation Trace Graph

DevPartner displays the method calls that allocated memory in the Allocation Trace graph. The Allocation Trace graph is available in RAM Footprint and Memory Leaks session files. It is visible in any view that includes an object list.


BugBenchDotN...isSnap.dpmem
Start Page
DevPartner Memory Analysis - Memory leak
Objects that refer to the most leaked memory

Referring object	Namespa	Leaked objects	Leaked size (bytes)	Call stacks	Additional references
Root instances of System.Windows.Forms.Internal.	<gcroot>	177	8,496	0	2
Object[]#44:String Table	System	27	2,476	1	4
Unreachable Object	<gcroot>	39	2,122	1	2
Root instances of System.Drawing.BufferedGraphics	<gcroot>	22	968	0	1
WeakReference[]#1	System	5	80	0	1
CultureInfo#5	System.	1	48	0	1
BufferedGraphicsContext#1	System.	1	32	0	1
Button#4	System.	1	24	0	1
Button#3	System.	1	24	0	1
PictureBox#1	System.	1	24	0	1
CheckBox#1	System.	1	24	0	1
Button#1	System.	1	24	0	1
frmBugBenchDotNET#1	BugBen	1	16	0	1
Root instances of BugBenchDotNet.MyScarceResou	<gcroot>	1	16	0	1

frmBugBenchDotNET.Main(void)
frmBugBenchDotNET..ctor(void)
frmBugBenchDotNET.InitializeComponent(void)
Object Reference Graph
Allocation Trace Graph
Source

Figure 5-14. Memory Analysis Allocation Trace graph

To display the Object List and associated Allocation Trace graph, do one of the following:

- Click **Show Complete Details** under the **Objects that refer to the most leaked memory** (Memory Leaks) or **Objects that refer to the most allocated memory** (RAM Footprint) graph in a memory analysis Results Summary.
- Drill down from the **Methods that allocate the most leaked memory** (Memory Leaks) or the **Methods that allocate the most memory** (RAM Footprint) view in a Memory Analysis Results Summary.

To view the Allocation Trace graph for an object, do one of the following:

- Select the object in the Object List and click the Allocation Trace graph tab at the bottom of the session file window.
- Right-click an object in the Object List and choose View Allocation Trace graph from the context menu.

DevPartner redraws the Allocation Trace graph for the selected object.

To view and edit the source code for any node in the Allocation Trace graph, right-click on the node and choose **Edit Source** from the context menu. DevPartner opens your source code for editing, at the selected method.

Viewing and Editing Source Code

Selecting the **Source** tab displays source code for the profiled application.

A **Source** tab view can be accessed from many points in the DevPartner memory analysis session windows, either by context menus, or by simply clicking the **Source** tab at the bottom of the session window. In addition to source code, the **Source** tab includes data about the individual lines of source code. The data available on the **Source** tab varies, based on the type of memory analysis performed and your data column display choices.

In addition to viewing data about your source code, you can jump directly to the source code in the Visual Studio editor by choosing **Edit Source** from one of the DevPartner memory analysis context menus. DevPartner opens the source file for editing at the line that corresponds to the object node, method node, or line of code in the **Source** tab from which you executed the **Edit Source** command.

Note: If the source code does not display or contains unintelligible characters, DevPartner may not have been able to determine the encoding of the source file. If you know the encoding, right-click in the source pane and choose **Encoding...** from the context menu. Select the correct encoding in the dialog and click **OK** to display the source file. From this context menu, you can also change to a different source file.

The **Source** tab consists of a view of application source code and includes data columns that contain information about the source methods used by your application. The data columns available are tailored to the context in which the **Source** tab appears. Different sets of data columns are available in Memory Leaks analysis, Temporary Objects analysis, and RAM Footprint analysis sessions.

Navigating the Source Tab

You can jump to the relevant line of source code on the **Source** tab from any object or method (for which you have source code) in the session window.

- From any Memory Leaks, RAM Footprint, or Temporary Objects results summary, click **Show Complete Details** to drill down into the session data
- In the session window, click the **Source** tab (at the bottom of the window)

Viewing Source Code

Use the following techniques to view the related source code in memory analysis.

Table 5-4. Viewing source code

View or Graph	Viewing Source Code
Object View	Select an object in the **Object List** and click the **Source** tab.
Object Reference Graph or Allocation Trace Graph	Right-click an object node and choose **View Source** on the context menu.
Method View	Select a method in the **Method List** and click the **Source** tab.
Call Graph	Right-click a method node in the **Call graph** and choose **View Source** on the context menu

Editing Source Code

Use the following techniques to edit related source code in memory analysis.

Table 5-5. Editing source code

Graph or List	Editing Source Code
Object Reference graph or Allocation Trace graph	Right-click an object node and choose **Edit Source** on the context menu. DevPartner opens the source file in Visual Studio for editing
Call graph, Object list, or Method list	Right-click a method in the **Object List, Method List,** or a node in the **Call graph** and choose **Edit Source** on the context menu. DevPartner opens the source file in Visual Studio for editing.

Customizing the Source Tab Data Columns

- Right-click on a column heading and use **Choose Columns** to change the data columns displayed in the **Source** tab. **Source** tab columns are not sortable.

Changing the Source File

- Right-click on the title bar of the **Source** tab window and use **Choose Another Source File…** to select a different source file. This creates a new mapping for the source file. It does not affect any other source paths.

Identifying Memory Problems

Consider the following scenario:

When your Quality Assurance team reports the first test results for your new managed application, you are pleased to learn that it does what it is supposed to do. But in later tests, QA runs longer test cycles and reports that the longer the application runs, performance slows down.

That is not what you want to hear. How do you know what part of your application to examine first? When you find the problem, how do you correct it?

To find problems in your application, run the application under DevPartner memory analysis. You do not have to wait until you suspect a memory problem to use DevPartner. Make testing your application's memory use with DevPartner a routine part of the development process.

DevPartner helps you quickly learn how your application uses memory resources, revealing current or potential problem areas.

After you start a memory analysis session without debugging, use the Session Control Window to observe how your program uses memory.

The real-time graph presents a visual representation of memory use. The class list updates dynamically to show the classes that use the most memory as your program runs. Right-click the class list to switch between the **Top 20 classes** and the **Top 20 classes with source**.

The **Session Control** buttons in the user interface allow you to take a snapshot of the managed heap for detailed analysis.

When you run a memory analysis session, you can choose to examine one of three important potential problem areas:

- Memory leaks
- Temporary object creation
- Overall RAM footprint

Table 5-6. Symptoms and Analysis tools

Symptom	Analysis Tool
Performance degrades over time; recovers on restart. Performance improves after restarting the application, but degrades again.	Memory Leaks
Scalability problems; temporary performance degradation.	Temporary Objects Memory Leaks
Sluggish performance, does not improve after restarting the application.	RAM Footprint Temporary Objects

First choose the appropriate memory analysis feature for the symptom that your application exhibits. You eventually want to run your application under all three types of memory analysis. Even if you do not find a problem, the thorough analysis enhances your understanding of how your program uses memory resources.

Running a Memory Analysis Session

The first thing you notice when running any memory analysis session is the real-time graph on the Session Control window. The real-time graph provides a visual representation of how your application uses memory resources. Observe the pattern the graph takes as you exercise your application. Different memory usage scenarios create characteristic patterns, so the real-time graph provides the first clue to the existence and nature of a memory problem.

Tip: Pay careful attention to the shape of the real-time graph as you run your application. You can often diagnose a memory problem immediately by observing and learning from the pattern of the graph.

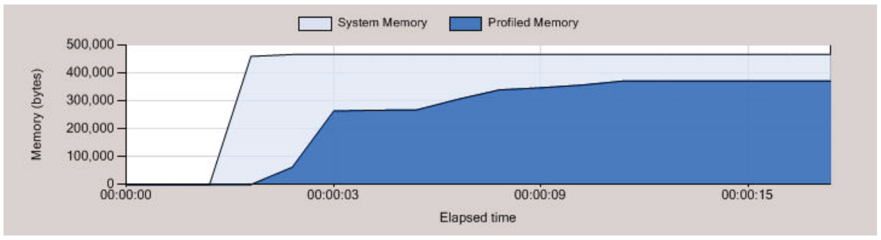


Figure 5-15. Memory Analysis Session Control window real-time graph

For example, if the graph shows a rising pattern that never returns to baseline, as in Figure 5-15, your application is probably leaking memory. You may suspect that the progressive slowdown of your application that your QA team noticed is consistent with a memory leak, but the real-time graph confirms that diagnosis.

If the graph returns to the baseline, but you see periodic spikes in memory use, your application is creating large numbers of objects as it runs. Granted, the allocated memory is being freed, but such an application may not scale well under load.

If your application slows down in response to an increase in users or inputs, the slowdown could indicate a scalability issue. Again, the real-time graph indicates the nature of the problem, enabling you to immediately point your diagnostic efforts in the right direction.

Even in the absence of a suggestive pattern, the real-time graph provides important information. For example, if your application consistently consumes nearly all of the memory allocated for the managed heap, and that amount is large relative to the anticipated resources of your target users' systems, you may want to reduce the overall memory footprint of your application. The next sections in this chapter provide detailed information about such cases and their implications for application performance.

Locating Memory Leaks

The amount of memory consumed by your application has a major impact on how well the application performs. The larger the amount of memory allocated, the more likely it is that the application runs slowly and scale poorly.

Leaked memory—the allocation of memory that is not reclaimed—can bloat your application's RAM footprint. Automatic garbage collection relieves you of the responsibility to explicitly free the objects that you create, so memory is not "leaked" in the classic C++ sense, but it is still possible to retain references to objects that the program never uses.

As long as a reference to an object exists, the referenced object is considered to be a **live object** by the garbage collector; a live object cannot be collected. This condition, like leaked memory in C++, is undesirable. Such references can be difficult to track down and that is where memory analysis helps you.

Consider Memory Leaks analysis.

Running a Memory Leaks Analysis Session

The Ready, Set, Go section "Using Memory Analysis Out of the Box" on page 132 also includes a procedure for using the Memory Leaks feature. The following is a quick summary of that process.

Isolating Memory Leaks with the Memory Leaks Feature

1 Start your application under memory analysis. Use the **Memory Leaks** tab in the **S**ession Control window.

2 Exercise the relevant features of your program to force any startup initialization to complete. The application warm-up also excludes initialization memory allocations from your analysis.

3 Click **Start/Stop** to begin tracking only newly allocated objects.

4 Exercise the feature of your program that you wish to test.

5 Click **Force garbage collection** to force a garbage collection on the active process.

6 Click **Start/Stop** again to end the tracking period and to exclude any new memory allocations.

7 Check the **Tracked instance count** and **Tracked size** columns in the **Class List**. If you see that objects have been allocated but not collected, click **View Memory Leaks** to capture a view of the managed heap that shows the tracked objects that remain after garbage collection.

   View Memory Leaks appears only after you click **Start/Stop** tracking for the first time.

DevPartner displays a snapshot of the state of the managed heap. The data is displayed as a **Memory Leaks Results Summary**. From the results summary page, you can drill into the memory use data, identify the problem, and locate the method(s) responsible in the source code.

Note: To enable DevPartner to properly identify most garbage collection roots in Memory Leaks or RAM Footprint sessions, **Start Without Debugging with Memory Analysis**. If you attempt to collect Memory Leaks or RAM Footprint data for an application started under **Start with Memory Analysis** (with debugging), all garbage collection roots appears as "`unidentified GC roots`" in the session data.

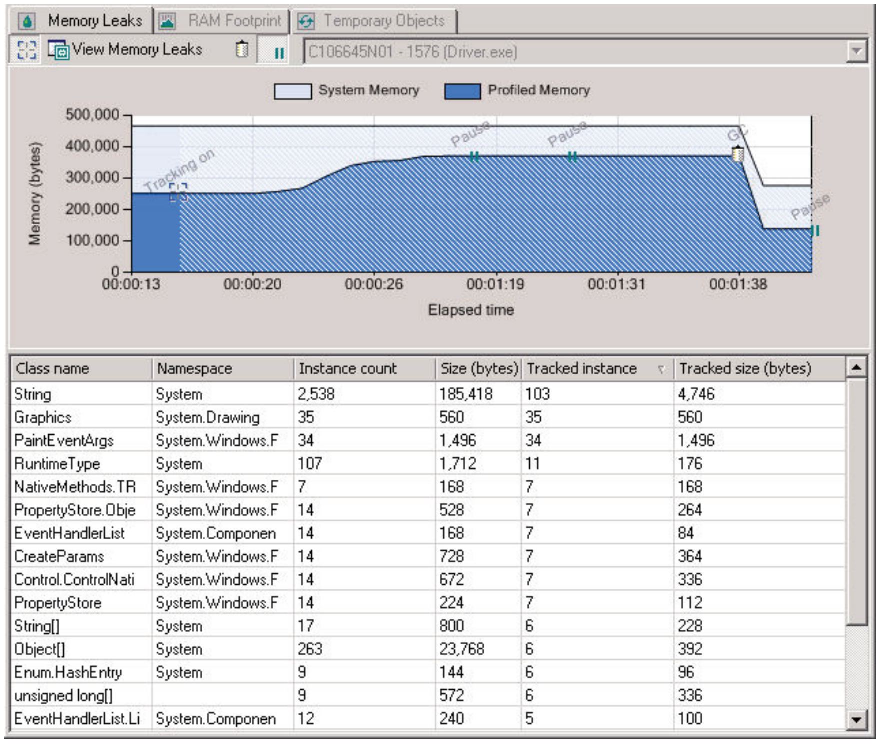
Understanding Memory Leaks Analysis Results

DevPartner Memory Leaks analysis defines a memory leak as any object that is allocated on the managed heap during a specified period of time, and has not been freed when you collect memory data. Memory Leaks analysis helps to reveal where your application holds memory that should be freed. Use this information to determine how to change your code so this memory frees.

To uncover memory leaks, run your application under the DevPartner Memory Leaks analysis feature and exercise the application in a way that should free previously allocated objects.

If memory use consistently rises and does not decrease (or does not decrease as you would expect it to) in response to garbage collection, your application could be leaking memory.

For example, see Figure 5-16. The real-time graph in this figure shows a rise in memory use that did not return to baseline after garbage collection. If you look at the Tracked instance count column for the classes that belong to your application, you notice that garbage collection is not collecting some tracked objects. Look for the number of uncollected instances in the Tracked instance count column in the Session Control window.



Class name	Namespace	Instance count	Size (bytes)	Tracked instance	Tracked size (bytes)
String	System	2,538	185,418	103	4,746
Graphics	System.Drawing	35	560	35	560
PaintEventArgs	System.Windows.F	34	1,496	34	1,496
RuntimeType	System	107	1,712	11	176
NativeMethods.TR	System.Windows.F	7	168	7	168
PropertyStore.Obje	System.Windows.F	14	528	7	264
EventHandlerList	System.Componen	14	168	7	84
CreateParams	System.Windows.F	14	728	7	364
Control.ControlNati	System.Windows.F	14	672	7	336
PropertyStore	System.Windows.F	14	224	7	112
String[]	System	17	800	6	228
Object[]	System	263	23,768	6	392
Enum.HashEntry	System	9	144	6	96
unsigned long[]		9	572	6	336
EventHandlerList.Li	System.Componen	12	240	5	100

Figure 5-16. Session Control window data display

Once DevPartner alerts you to a possible leak, use the **Memory Leaks Results Summary** (session file) that DevPartner creates to locate the source of the leak so you can fix it. The Memory Leaks analysis results summary gives you the following ways to drill down into your data:

- Objects that Refer to the Most Leaked Memory
- Methods with the Most Leaked Memory

Each chart shows the top five objects or methods that are associated with leaked memory. To see more information about the top five objects or methods, click **Show Complete Details** for that chart.

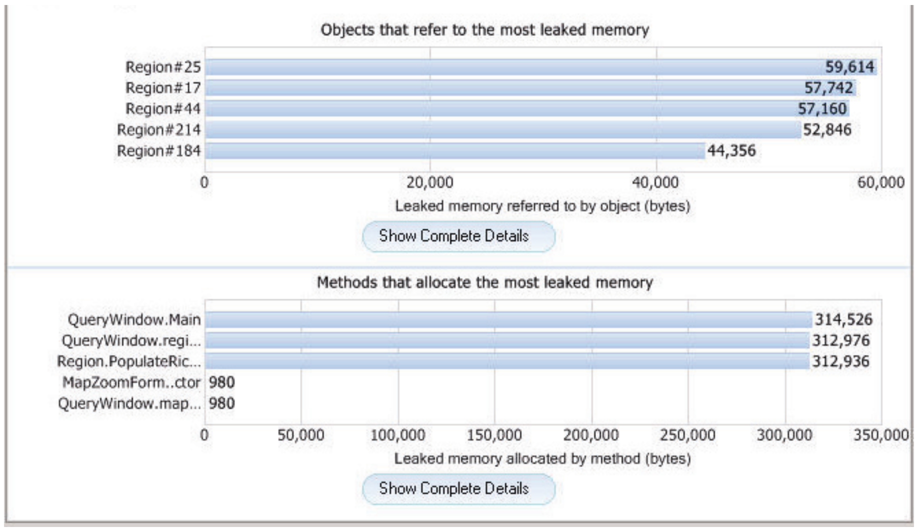


Figure 5-17. Results Summary appears when you click View Memory Leaks

The starting point you choose depends on the problem that you want to solve and your preferred approach to the problem. For example:

- If you notice that a limited set of specific objects are being leaked, you can use the **Objects that refer to the most leaked memory** graph to quickly see which objects hold references to the leaked objects.
- If you are familiar with the source code for the allocating method and can tell by examining the source code whether the leaked object should have been cleared, you may want to start with the **Methods that allocate the most leaked memory** chart.

From both the objects and methods charts, you can quickly switch to a view that shows another aspect of your data.

When viewing complete details for **Objects that refer to the most leaked memory**, you can select these views:

- Object Reference Graph
- Allocation Trace Graph
- Source

When viewing complete details for **Methods that allocate to the most leaked memory**, you can select these views:

- Call Graph
- Source

The following example uses **Objects that refer to the most leaked memory** as a starting point.

Objects that Refer to the Most Leaked Memory

The following example shows a case where a limited set of objects causes leaked memory. The example also presents other possible approaches for problem diagnosis.

The garbage collector cannot clear an object as long as there is at least one existing reference to that object. When your application runs, it creates objects. Some objects are needed for as long as the program runs. These are permanent, or long-lived objects. However, most objects should become eligible for garbage collection once they are no longer referenced by another object.

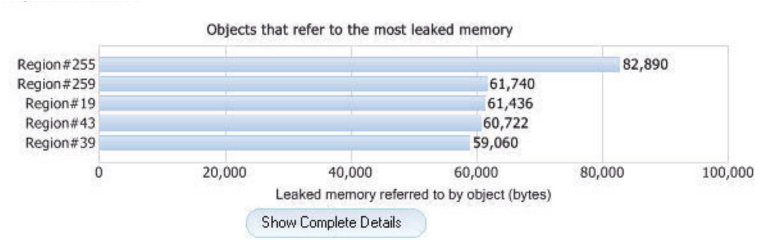
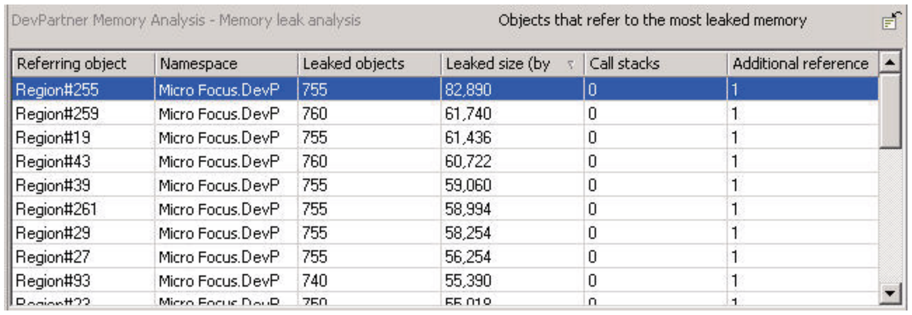


Figure 5-18. Objects that refer to the most leaked memory

The chart in Figure 5-18 shows the top five objects that hold references to the most leaked memory. These objects prevent the leaked objects from being freed. Referring objects that account for the biggest memory hit appear at the top of the chart. The data indicates that a particular set of objects is responsible for the leaked memory. Use this chart as the starting point to drill into session data and locate the source of the memory leaks.

Clicking **Show Complete Details** (below the bar graph; see Figure 5-19 on page 159) opens a detailed display for objects that refer to leaked memory.

The top panel of this display lists all of the objects that refer to leaked memory, as displayed in the chart in Figure 5-19. This list includes the top five objects displayed in the original bar graph and other objects that refer to smaller amounts of leaked memory.


DevPartner Memory Analysis - Memory leak analysis
Objects that refer to the most leaked memory

Referring object	Namespace	Leaked objects	Leaked size (by	Call stacks	Additional reference
Region#255	Micro Focus.DevP	755	82,890	0	1
Region#259	Micro Focus.DevP	760	61,740	0	1
Region#19	Micro Focus.DevP	755	61,436	0	1
Region#43	Micro Focus.DevP	760	60,722	0	1
Region#39	Micro Focus.DevP	755	59,060	0	1
Region#261	Micro Focus.DevP	755	58,994	0	1
Region#29	Micro Focus.DevP	755	58,254	0	1
Region#27	Micro Focus.DevP	755	56,254	0	1
Region#93	Micro Focus.DevP	740	55,390	0	1
Region#23	Micro Focus.DevP	750	55,018	0	1

Figure 5-19. List of objects that refer to the most leaked memory

The default setting is to sort the objects by the **Leaked size** (total size of the leaked objects referred to by the selected object) column. You can also sort the list by any of the other columns to help you see patterns in the data. If you right-click an item in the list and choose **View leaked objects referenced by this object**, you see the objects that were actually leaked.

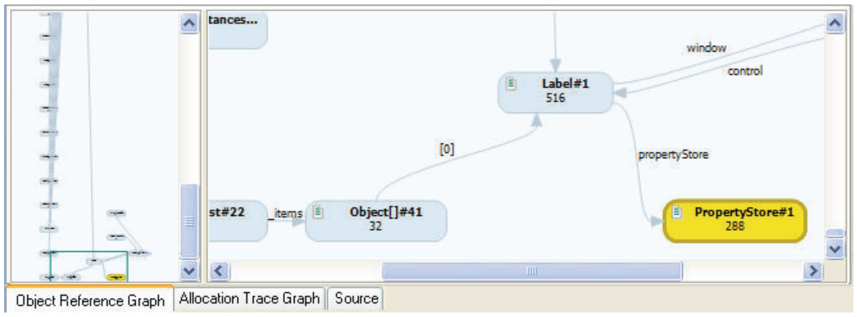


Figure 5-20. The Object Reference graph shows why an object is still in memory

Select a referring object that you want to examine. It is important to be able to quickly understand the sequence of references that keep these objects in memory. Click the Object Reference graph tab to view the reference graph. The Object Reference graph shows why the garbage collector did not clear the selected object. The display shows the chain of objects between the selected object and the garbage collection root(s) that are keeping the object alive.

Scroll down the list of objects to evaluate the other objects. Some object reference graphs are quite simple, while others may be quite complex. You may find evidence that indicates conditions such as many references to small objects or a few references to large objects. The goal is to use this graph to determine the point in the chain of referring objects where it is most efficient to eliminate the leak.

The chain of referring objects shown in the Object Reference graph can range greatly in complexity. In many cases, there are multiple referrers and the graph becomes very complex. Drag the navigation frame in the overview pan**e** or click on a node to change the nodes displayed in the detail pane. If an analysis presents you with a complex graph, simplify the view by right-clicking a node and selecting **Show Fewer Referrers**. You can also drag nodes within the graph for easier viewing.

The labels such as **elements** on the connecting arrows represent the referring data member in the next class in the graph. Bracketed numbers identify arrays. If you know your code well, the labels speed up the process of identifying potential problem areas.

You can also right-click a node and select **Edit Source** to open the related source code within Visual Studio, or view the related source code by selecting the **Source** tab. DevPartner highlights the line in the method that allocated the object in the graph.

To increase program understanding, you can view the source for each node in the graph sequentially and see the events that led to the allocation of the memory that leaked. DevPartner offers alternate ways to view these program events. For example, the Allocation Trace graph shows who called each method that allocated the selected object.

You can go directly from the object in the list to the source code. In real-world problem solving, you should drill down using an appropriate method for the problem that you are trying to solve or that corresponds to the way you think about your code.

Alternate Methods of Solving the Problem

The preceding example focused on use of the object reference path to locate the source of a leak. There are other ways to approach the memory leak source. For example:

- Look at the Allocation Trace graph to determine who called the method that allocated the object. From there go to the source code.
- Go directly to the source code from the list of objects.

Remember that DevPartner presents different views of your data on the **DevPartner Memory Analysis - Memory leaks analysis** summary. The first example used **Objects with the Most Leaked Memory**. However, depending on the complexity of the data, or on your own preferences, you could examine a problem from any of the following graphs on the **DevPartner Memory Analysis - Memory leaks analysis** summary.

Methods that Allocate the Most Leaked Memory

The following graph, which appears in the lower half of the **DevPartner Memory Analysis - Memory leaks analysis** summary, shows the top five methods that allocated objects that were leaked. When you click **Show Complete Details**, DevPartner provides a list of all methods that leaked objects, with access to a Call graph view and to the source code for the method, if available.

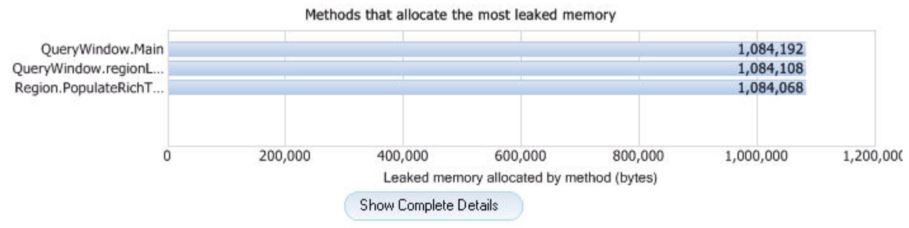
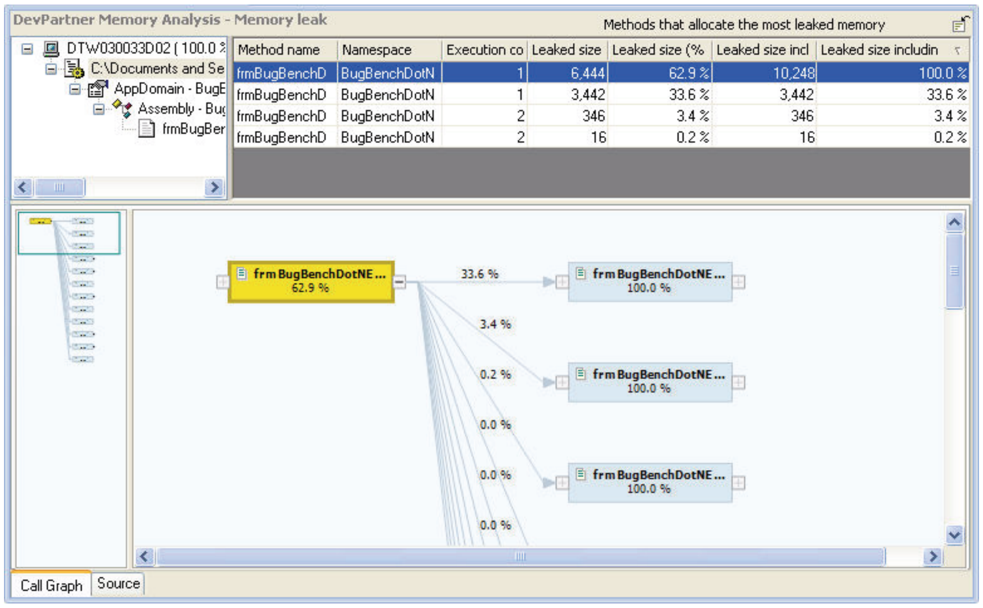


Figure 5-21. Methods that allocate the most leaked memory

Select a method in the Method list to see the objects that were allocated from the method that were leaked. You can also view the source code for the method which shows the lines that allocated the leaked objects along with statistics about the number and size of objects leaked on the line.


DevPartner Memory Analysis - Memory leak
Methods that allocate the most leaked memory
DTW030033D02 (100.0 %
C:\Documents and Se
AppDomain - BugE
Assembly - Bug
frmBugBen
frm BugBenchDotNE... 62.9 %
33.6 %
3.4 %
0.2 %
0.0 %
0.0 %
0.0 %
frm BugBenchDotNE... 100.0 %
frm BugBenchDotNE... 100.0 %
frm BugBenchDotNE... 100.0 %
Call Graph
Source

Method name	Namespace	Execution co	Leaked size	Leaked size (%	Leaked size incl	Leaked size includin
frmBugBenchD	BugBenchDotN	1	6,444	62.9 %	10,248	100.0 %
frmBugBenchD	BugBenchDotN	1	3,442	33.6 %	3,442	33.6 %
frmBugBenchD	BugBenchDotN	2	346	3.4 %	346	3.4 %
frmBugBenchD	BugBenchDotN	2	16	0.2 %	16	0.2 %

Figure 5-22. Details for methods that allocate the most leaked memory

For example, to drill into to the data:

- Right-click a method in the Method list**.**
- From the selected method, go to a list of the objects allocated by the method or to a Call graph for the method.
- From an object in the Object list, view a list of referenced objects, an Object Reference graph, or an Allocation Trace graph**.**
- From a method, or a node in a Call graph or Allocation Trace graph, view source code with object allocation data for individual lines.
- From a method or object in a list, or a node in a Call graph, Object Reference graph, or Allocation Trace graph, or a line of source code; choose **Edit Source** to open the source to the appropriate line for editing.

Solving Scalability Problems with Temporary Objects

When performing memory analysis with DevPartner, you can use Temporary Objects analysis to diagnose and correct scalability problems.

Examples of Scalability Problems

Scalability problems surface when an application runs well until users work with the application more intensively. For a client-server application, this might happen when the number of users increases. For a standalone application, this might happen after numerous text manipulations or mathematical computations. These can be labeled as *scalability* problems. As the scale of the work done by the application increases, performance degrades.

A Possible Cause: Temporary Objects

One possible cause of scalability problems is the creation of too many temporary objects. In this case, object creation becomes a performance bottleneck—a problem that requires correction.

Creating and using objects is important within managed Visual Studio programs. Unfortunately, some coding techniques have the side-effect of creating many objects.

Part of the problem is the creation of objects such as those created with the **`String`** class. It takes processing cycles to create objects and later destroy these objects. If you can reduce the number of objects created, you can generally expect better performance.

Object Life Span

DevPartner tracks the objects allocated by your code and categorizes them based on how long it takes for them to be collected. There are three categories:

- Short-lived — collected at the first garbage collection after the object was allocated (generation 0)
- Medium-lived — collected at the second garbage collection after allocation (generation 1)
- Long-lived — survives across many (or all) garbage collections during the run of the program (generation 2)

Note: The Microsoft .NET Framework garbage collector supports three generations, designated 0, 1, and 2. Objects allocated since the last run of the garbage collector are in generation 0. Objects that survive one garbage collection after allocation become generation 1 objects. Generation 1 objects that survive one or more additional garbage collections become generation 2 objects.

DevPartner combines short- and medium-lived object allocations in a temporary objects category.

Medium-lived objects have the greatest impact on performance, and cause the garbage collector to work harder than necessary. Individual short-lived objects have less impact on garbage collection, although there is still a performance penalty for calling the object's constructor. However, creating large numbers of short-lived objects may cause bottlenecks and memory shortages.

If you believe that your code has scalability issues, use DevPartner to monitor memory used by your code as it executes. If the real-time graph in the Session Control window shows an up-and-down, wavelike pattern—which suggests that your application is creating many temporary objects—you can use DevPartner to analyze the application for temporary object creation.

DevPartner categorizes the results of temporary object analysis by entry points and by methods. Regardless of which technique that you use to drill into the data, DevPartner helps you see how much memory the temporary objects consume and identify the specific lines of code that allocate the temporary objects.

Running a Temporary Objects Analysis Session

Use the following procedure to analyze your application for problems caused by temporary object creation:

1. Start your application under memory analysis. Use the **Temporary Objects** tab in the Session Control window.

2. Exercise your application, then do one of the following to see the parts of your program that have allocated the most temporary objects:

   a. Click **View Temporary Objects** when you observe a system garbage collection (the falling pattern) in the Session Control window.

   b. Click the **Force Garbage Collection** icon, then immediately click **View Temporary Objects**.

   c. Quit the program. DevPartner forces a garbage collection and creates a temporary objects session file.

Note: If you are running your application in the debugger, do not use the debugger to stop your application. DevPartner does not produce a session file in response to this action. Stop the application normally to generate a session file.

3. To examine the temporary object allocation behavior for a specific part of your application, click **Clear all memory** to clear the collected temporary object allocation data. Afterwards exercise the relevant part of your application, force a garbage collection, and click **View Temporary Objects**.

   DevPartner always creates a temporary objects session file when you quit an application running under memory analysis. DevPartner also creates a temporary objects session file in response to the snapshot action executed in a session control file or the Session Control API.

DevPartner displays a snapshot of the state of the managed heap. The data is displayed as a **Temporary Objects Results Summary**. From the results summary page you can drill into the object creation data, identify the problem, and locate the method(s) responsible in the source code.

Identifying Scalability Problems

DevPartner enables you to locate potential trouble spots and then drill down into your application's use of temporary objects to identify problems and improve the overall quality of your code.

Real-time Graph

The real-time graph provides a high level view that enables you to identify problematic areas.

If your application is creating large numbers of short- and medium-lived objects, you see a peak in profiled memory in the real time graph which diminishes when the garbage collector runs. If you exercise the feature again after garbage collection, you see another peak which is caused by creating a new group of temporary objects.

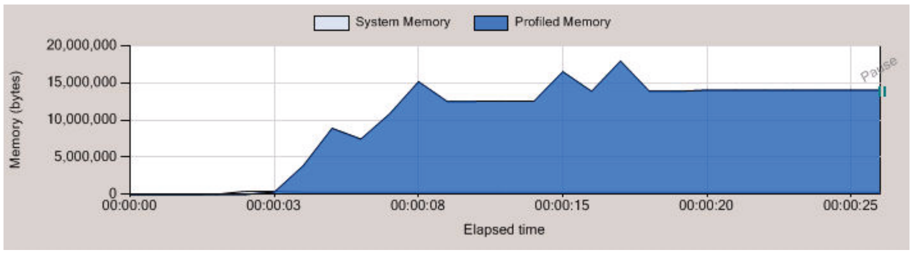


Figure 5-23. Real-time graph suggesting excessive temporary object creation

The classes responsible for the most objects appear in the list of profiled classes which are sorted by the **Size** column. This highlights the classes whose objects consume the most temporary memory. Also notice the **Instance Count** which shows how many object instances were created for each class.

Figure 5-23 shows a real-time graph that suggests excessive temporary object creation. Spikes in the graph show where your application is creating lots of objects. Excessive object creation can create major performance or scalability issues in a managed application and especially in server applications. Even if scalability is not an issue, methods that allocate many short-lived objects often indicate easy-to-fix performance problems.

Viewing Temporary Objects

Click **View Temporary Objects** to collect data at a specific point in your application. DevPartner displays a **Temporary object analysis** page that categorizes the data by entry points and by methods that create the most temporary objects.

An **entry point** is a profiled method that is called by excluded (that is, system or third-party) code. When your application runs, monitoring begins with the first call to a profiled, or user-code, method. (User-code methods are methods in your application source code.) This is called an entry point. All calls made to other user-code methods from that point are considered to be part of the entry point.

Methods that are called only by other user-code methods are not entry points. However, such methods could be responsible for large amounts of temporary memory use. The second chart on the Results Summary highlights methods that allocate lots of temporary memory, but are not necessarily entry point methods. Thus, if a child method called by an entry point is the major memory allocator in your application, you can locate that method in **Methods that use the most memory** without having to follow the Call graph for the entry point method that called it.

From the Results Summary view you can drill into the data in order to understand how much memory the objects allocated by these methods are consuming, and to identify the lines of code that are creating the short- and medium-lived objects.

Analyzing Temporary Object Data

Clicking **Show Complete Details** that is below either chart opens a detailed view of all the entry points, or all the methods in your application that allocated temporary memory. In addition to the complete list of methods, the view includes a Call graph and a **Source** tab.

The available data columns in the Method List provide more extensive data about your application’s methods than those in the list of profiled classes on the Session Control window.

Call Graph

Click on an entry point in the entry points list or a method in the method list to view a Call graph for the method. The Call graph shows the selected method and its child methods, and highlights the **critical path** with a bold, gold-colored line. The **critical path** is the sequence of child method calls that resulted in the largest cumulative memory allocation for the selected method.

Methods appear as nodes in the Call graph. Each node can display data about memory allocated by the method. In addition, the links between nodes can display data about memory allocated by that branch of the graph. The data is expressed as percentages of memory allocated.

- **Nodes -** The percentage of memory allocated by the method that is attributable to the body of the method itself.
- **Links -** The percentage of memory allocated by the method that is attributed to child methods that are executed in that branch.

This is how DevPartner shows you not only which methods are responsible for the temporary objects your application creates, but exactly where in the paths of execution the allocations occur.

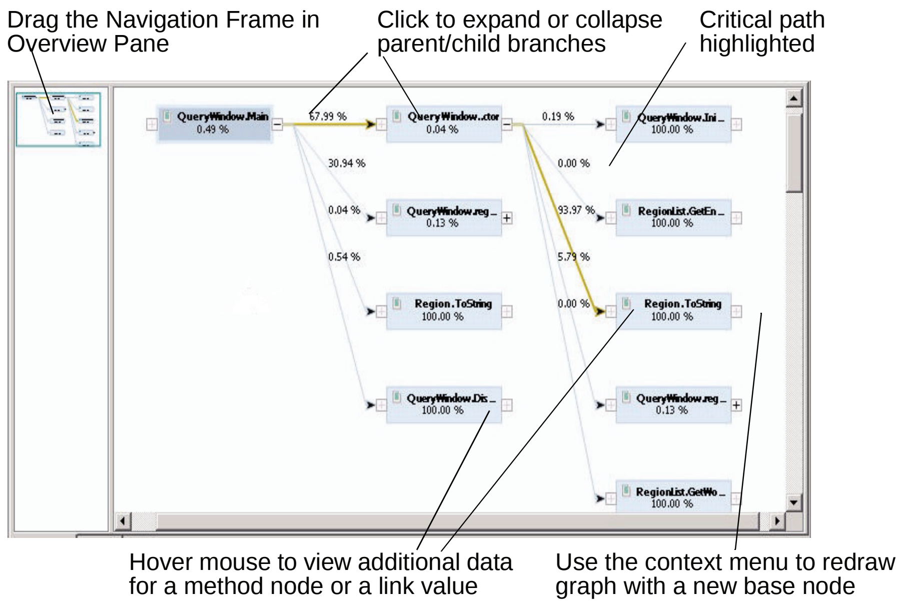

Drag the Navigation Frame in Overview Pane
Click to expand or collapse parent/child branches
Critical path highlighted
QueryWindow.Main 0.49 %
67.99 %
QueryWindow..ctor 0.04 %
0.19 %
QueryWindow.Ini... 100.00 %
30.94 %
0.00 %
0.04 %
QueryWindow.reg... 0.13 %
93.97 %
RegionList.GetEn... 100.00 %
0.54 %
5.79 %
Region.ToString 100.00 %
0.00 %
Region.ToString 100.00 %
QueryWindow.Dis... 100.00 %
QueryWindow.reg... 0.13 %
RegionList.GetWo... 100.00 %
Hover mouse to view additional data for a method node or a link value
Use the context menu to redraw graph with a new base node

Figure 5-24. A Call graph for an entry point method shows the critical path

Right-click on any node in the Call graph to:

- Redraw the Call graph for the selected node
- View source code for the selected node
- Edit source code for the selected node

Source View

When you view the source code for an entry point or method in the **Entry points that allocate the most memory** or the **Methods that use the most memory** method lists**,** DevPartner opens the Source tab view to the selected method. In addition to the source code, DevPartner provides detailed information about the memory allocated by individual lines in your application code. The information includes how often the line executed; the number of short-, medium-, and long-lived objects, including or excluding child objects, allocated on the line; and the accumulated sizes (memory load) of these objects.

Interpreting Results to Fix Scalability Problems

The following list suggests some of the possible ways to interpret memory analysis results to fix memory-related scalability problems.

- Look at the Temporary Objects analysis page to determine if an entry point or non-entry point method consumes the most temporary memory.
- If the largest consumer of temporary objects is an entry point, drill down using the **Entry points that allocate the most memory** view to determine which methods in the entry point's execution path require the most temporary space and should be modified or called less often.
- If the largest consumer is a non-entry point method, drill down using the **Methods that use the most memory** view to determine which parts of your code to modify.
- Compare the number of short- and medium-lived objects, as well as the amount of temporary space they consume. Use this information to determine which parts of your code to modify.
- If both short- and medium-lived objects consume similar amounts of temporary space, you can run a performance analysis to find out how much time the constructor uses to create the temporary object.
- Use the Call graph to understand the relationship between methods that allocate temporary memory. Examine characteristics of different methods: percentages of memory consumed; actual bytes used; and numbers of temporary objects created. Use this information to identify which method to modify.
- Use the **Source** tab to identify specific lines in your code that allocate temporary objects. Examine the kind and sizes of objects created, and how often the line is executed. Use this information to identify more efficient ways to use objects.

Using RAM Footprint to Improve Performance

Some managed applications consume hundreds of megabytes of RAM while they are running. This chapter examines some specific memory problems in this chapter: memory leaks, which can cause your application to consume more and more memory as it runs until it eventually exhausts the heap, and periodic spikes in memory use caused by excessive temporary object creation, which can lead to scalability issues. These problems adversely impact your application's memory use. These problems also contribute to your application's memory footprint. Your application may be well-behaved with respect to these errors—but performance may still seem slow, especially when run in various end-user environments.

One possible cause of sluggish performance is that your application may use excessive amounts of memory as it runs. What is excessive? That depends on the environment—hardware and software—in which your application is used. You may have a good idea of the end user environment, but the environment can change. For example, the end user may run several other applications at the same time as they run yours which competes for memory resources. Nor can you force hardware upgrades on your users every time you release a new version of your application. All of this makes a strong argument for keeping your application's memory footprint small.

More specifically, the memory use addressed here is the RAM footprint, not just overall memory use. The biggest effect to application performance—and your end-users' perception of your application—is to force the application to rely on the operating system's virtual memory system. Paging managed objects into virtual memory greatly decreases application performance.

What can you do to optimize your application's use of RAM resources? DevPartner provides RAM Footprint analysis as part of its memory analysis capability. Run RAM Footprint analysis regularly as you develop your application. The way your application uses RAM resources is most likely a result of application design and architecture. It is much easier to re-design a feature early in the development process than to wait until the application is ready for beta release.

Measuring RAM Footprint

DevPartner helps you focus your performance tuning effort on the areas that have the greatest impact on RAM consumption. When you run your application under RAM Footprint analysis, DevPartner enables you to:

Tip: See the DevPartner Studio online help for procedures related to measuring a RAM Footprint.

- View the real-time graph of your application's RAM consumption, and view the real-time list of profiled classes associated with the most bytes of memory.
- Take snapshots of the managed heap which you use to examine the objects responsible for the most memory use.

To measure RAM footprint:

1. Start your application under memory analysis. Use the **RAM Footprint** tab in the Session Control window.
2. Exercise your application to get it into a steady state for which you wish to examine memory use.
3. Click **View RAM Footprint** to see a detailed snapshot of the managed heap at that point in time.
4. Remember that the garbage collector only runs when available memory is exhausted, so the memory graph may not accurately represent the amount of memory in use at a given time. When your program is in an idle steady state, click **Force garbage collection** to force the garbage collector to run and update the memory graph.

DevPartner displays a snapshot of the state of the managed heap. The data is displayed as a **RAM Footprint Results Summary**. From the results summary page you can drill into the session data and locate the objects and methods responsible for the most memory use.

To enable DevPartner to properly identify most garbage collection roots in Memory Leaks or RAM Footprint sessions, **Start Without Debugging with Memory Analysis**. If you attempt to collect Memory Leaks or RAM Footprint data for an application started under **Start with Memory Analysis** (with debugging), all garbage collection roots appears as "`unidentified GC roots`" in the session data.

Use the RAM Footprint analysis page to gain an in-depth understanding of how your application uses memory. The **RAM Footprint Results Summary** gives you the following ways to examine and drill down into your data:

- Object Distribution
- Objects that refer to the most allocated memory
- Methods that allocate the most memory

Which you use first depends on the data presented and, to some extent, on the way you tend to think about your application.

Object Distribution

DevPartner presents the distribution of objects in memory as a pie chart so you can immediately see the proportion of memory used by your application (**Profiled objects**) relative to that used by system code (**System objects**).

Interpreting the **Object Distribution** chart:

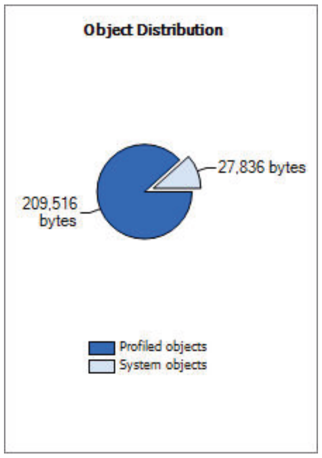


Figure 5-25. DevPartner Memory Analysis Object Distribution chart

- If your application (**Profiled objects**) is the largest wedge in the pie, and memory use is moderate to high relative to expected resources in the target deployment environment, you should determine which parts of your application allocate the most memory. To do this, use the **Objects that refer to the most allocated memory** or the **Methods that allocate the most memory** chart to drill down into the data. Ultimately, you want to locate in source code those parts of the application that you can change or restructure to use less memory.

- If the **Profiled Objects** part of the pie chart is small, your application is not the main allocator of memory. This is a good thing. But if the application still seems sluggish or if overall memory use is high, you may want to investigate how your application is using unmanaged code or system resources. Unmanaged code can pin objects in memory. Visual Studio applications often spend a great deal of time in the .NET Framework; you may find that you can call .NET Framework methods more efficiently, or less often.

Drill into the RAM footprint data by using either of the following two analysis paths:

- Objects that refer to the most allocated memory
- Methods that allocate the most memory

Objects that refer to the most allocated memory

Objects that refer to the most allocated memory shows the objects that held references to live objects at the time the session file was generated. The size displayed is the total of all objects referenced from this instance.

- Click **Show Complete Details** to drill into the data for these objects.

Objects that refer to the most allocated memory enables you to focus on instances of objects that are responsible for the largest amounts of memory. Organizing the data by instances of objects that hold references to allocated memory highlights **large objects**, that is, the objects for which a maximum amount of memory would be reclaimed if the object could be garbage collected.

While an individual object might be small, it becomes much larger, i.e., a large object, when you include the memory consumed by the objects to which it refers. When the garbage collector runs, it cannot collect objects that are referenced by other objects. Thus an object that refers to many other objects may account for a considerable amount of memory. If you can collect such an object, you can also collect any other objects to which it holds a unique reference. Such large objects are obvious targets when you are trying to reduce an application's RAM footprint.

The **Objects that refer to the most allocated memory** view includes a list of live object instances with data about each object's impact on memory at the time the session file was created. It also includes a tabbed window in which you can view an Object Reference graph, Allocation Trace graph, and Source view.

This view helps you identify the largest objects in memory. **Referenced Size** data includes memory attributable to all child objects for which the object is the only parent. Considered singly, objects tend to be small. However, an object with several child objects, each of which may also have child objects, plus per-object overhead for parent and child objects, may actually consume a large amount of memory.

DevPartner uses the **Object Reference path** to roll up the bytes associated with child objects and attributes them to the parent object. The advantage of this view is that the view lets you focus on those objects that provide the biggest benefit if you can change the way they are allocated.

Once you zero in on the objects that consumed the largest amount of memory, you may immediately see changes that you could make to reduce memory consumption. However, you may want to investigate further to understand the implications of freeing or changing the way the application uses a particular object.

- Double-click the selected object in the instance list, or use the context menu to view the live objects referenced by the selected object.

Live Objects Referenced by <*object name*>

The **Live objects referenced by object** view shows you all of the live objects which are referenced in memory that are referenced by the selected parent object. In other words, these child objects could also be collected if the parent object could be collected.

All Objects Referenced by <*object name*>

The **All objects referenced by object** view displays an instance list of the objects referenced by an object selected in the **Live Objects Referenced by object** window.

Like its parent windows, the data presented in **All Objects Referenced by object** is organized by instances of objects that hold references to allocated memory. This view enables you to further examine the chain of references that are keeping objects in memory. In the **All Objects Referenced by object** window, you can examine the entire chain of objects referenced by any of the child objects in **Live Objects Referenced by object**.

You can continue to drill down from any object and view all the objects to which it holds a reference through the entire sequence of object references.

The Object Reference and Allocation Trace Graphs

All of the Object views discussed above include an **Object Reference Graph** and an **Allocation Trace Graph**.

The **Object Reference Graph** shows live objects in memory at the time that the session file was created. A live object is an object on which methods can be invoked. When the garbage collector runs, the collector identifies the objects that have valid references. A valid reference means that an object is reachable from the application's garbage collection roots. Reachable objects are marked as live objects and cannot be collected. The **Object Reference Graph** shows these object references and helps to explain why the objects are still in memory.

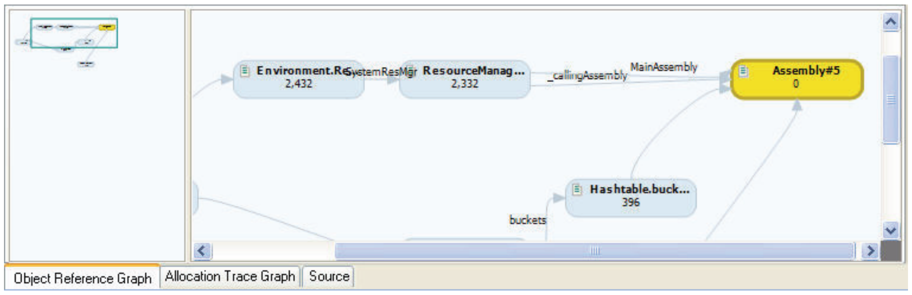


Figure 5-26. The Object Reference graph

Methods in your application allocate objects and the memory that the objects use. It is useful to know the sequence of method calls that allocated memory. The **Allocation Trace Graph** shows the method calls that allocated an object.

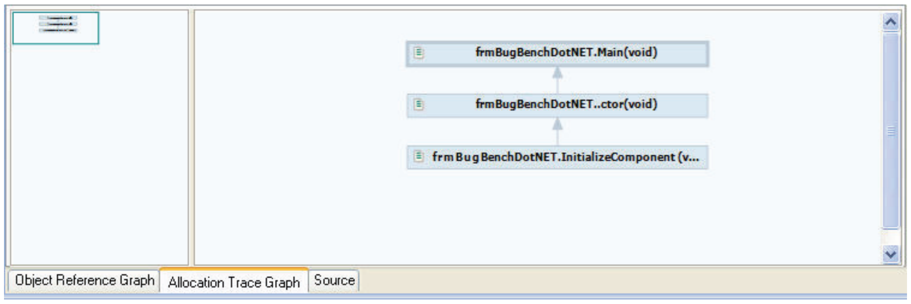

frmBugBenchDotNET.Main(void)
frmBugBenchDotNET..ctor(void)
frmBugBenchDotNET.InitializeComponent (v...
Object Reference Graph
Allocation Trace Graph
Source

Figure 5-27. Allocation Trace graph

Methods that Allocate the Most Memory

The **Methods that allocate the most memory** view displays a **Method List** showing the source methods that allocated the most live memory for the application. This view displays methods that allocate the most memory in the managed heap, but cannot be freed by garbage collection while the application is in its current state.

The **Live size including children (%)** column indicates the percentage of memory used by the method (and its child methods) relative to total allocated memory in the managed heap at the time the session file was created. The display focuses attention on the most memory intensive methods.

In addition to a view of the source code, this view also includes a Call graph which shows the execution path responsible for the memory allocation. See “Call Graph” on page 166 for more information.

Live Objects Allocated by This <*method name*>

The **Live objects allocated by this <*method name*>** view displays a list of the live object instances allocated by the method that you selected in the **Methods that allocate the most memory** view. In this case the view is limited to live objects allocated by the methods that were selected in the previous window. This enables you to drill down from the methods in your application that were the largest allocators of live memory. From here, examine the objects that were not available for garbage collection when the RAM footprint snapshot was taken.

The list of allocated objects includes objects created by non-profiled (system) methods that are called by the user-code method selected in the **Methods That Allocate the Most Memory** view. For example, if your method uses methods in the WinForms library, objects allocated by those methods appear in the list of allocated objects.

In order to understand how your application allocates objects, drill down to examine all of the objects referenced by any live object allocated by the method under study. The **All objects referenced from this instance** view is identical to the view described under “All Objects Referenced by <object name>” on page 171.

Through an entire sequence of object references, continue to drill down from any object and view all the objects to which it holds a reference.

Optimizing Memory Use

Once you understand how your application uses memory, you can begin to optimize memory use. Objects are typically the largest memory consumers, so start your analysis with them.

Your application probably creates several objects as it runs. To optimize performance, do you simply reduce the number of objects created? How do you know where to focus your performance tuning efforts?

Fortunately, DevPartner does much of the cost/benefit calculating for you. Remember that individual objects may be small, but when you consider objects with their children, some objects are much larger than others. DevPartner uses the concept of large object to alert you to those objects which, with their child objects, are large consumers of memory. Focusing your tuning efforts on these object allocations promises the most rapid route to a reduced RAM footprint.

Be aware of medium-lived objects. Medium-lived objects survived the first garbage collection to move to generation 1; they are collected during the second garbage collection after the transaction completes. This is the new amount of memory your transaction requires. If you could reduce the number of objects allocated, you could probably improve performance.

Look at live objects at several points as your application executes. Have you allocated objects that are unneeded for the remainder of the transaction? Could you share any live objects between multiple transactions? Have you allocated any objects that the application does not need until a later time? If the answer is yes to any of these questions and you can change how your application allocates objects, you can probably reduce the RAM footprint and improve performance.

Analyzing Web Applications with Memory Analysis

With DevPartner Studio, you can analyze memory use in managed Web applications developed in Visual Studio, including applications that use Web Forms, XML Web services, and ASP.NET. To collect server-side data, DevPartner Studio must be installed on the server system.

If the server application runs on a remote computer, install DevPartner Studio and the DevPartner Studio Remote Server license on the remote system to collect the server data. See the *DevPartner Studio Installation Guide* and the *Distributed Licensing Management Installation Guide* for more information. To configure data collection on the server, use the DevPartner memory analysis properties in Visual Studio.

Note: DevPartner session files are saved with the current solution. Opening a Web project from IIS directly, as opposed to opening the project through Visual Studio, may cause a different solution file to be used. DevPartner session files created in the first solution would not be visible in the second solution

Collecting Server-side Memory Data

You may want to collect memory analysis data for parts of a Web or client/server application. With DevPartner, you can collect memory data for managed code in any process as you run the client application.

To collect remote process data, install DevPartner Studio on the client and DevPartner Studio and the DevPartner Remote Server license on the remote computer. Use this configuration to collect data for a distributed application as it is actually deployed. See the *DevPartner Studio Installation Guide* and the *Distributed Licensing Management License Installation Guide* for more information.

Collecting Data from Multiple Processes

Web or client/server applications may run more than one process, but DevPartner collects memory analysis data for only managed applications. For example, when you profile an ASP.NET application, DevPartner does not collect data for the browser process (`iexplore`). However, DevPartner collects data for managed code that runs in the `aspnet_wp` or `w3wp` processes.

When you run such applications under memory analysis, the memory analysis session control window in Visual Studio displays the server and surrogate processes in the process selection list. Use the process list to focus data collection.

See "Starting Analysis from the Command Line" on page 285 for information on using **`DPAnalysis.exe`** and an XML Configuration file to profile multi-process applications.

Prerequisites for Analyzing Web Applications

For DevPartner memory analysis to successfully profile an ASP.NET application, the following two conditions must be met:

- The project must include a **`web.config`** file.
- The project must be configured for debugging. To do this, the **`web.config`** file must include a compilation element with the debug attribute set to true. For example:
  `<compilation debug="true" />`

Running a Memory Analysis Session on a Web Application

Follow these steps to analyze memory use in a Web application:

1. In Visual Studio, open the Solution that contains the project for the application.
2. Review the **DevPartner Coverage, Memory and Performance** properties for the projects within the solution.
3. Select the project in the Solution Explorer.
4. To display the Properties Window, select **View** > **Properties Window**.

Note: Logging off or rebooting the system changes the analysis option only if you are connected to a Terminal Server through a Terminal Services Client or to any system through Remote Desktop.

If you reboot or start up slower computers with memory analysis enabled, the Service Control Manager may report unresponsiveness when starting SMTP, FTP, and WWWP services. You can safely ignore these messages. All of the services start successfully. Lack of response is reported because DevPartner instruments IIS when memory analysis is enabled and these services depend on IIS.

5. If the server components do not run on the local computer, use the DevPartner data collection properties to set up remote data collection options.

6. Depending on the type of data you wish to collect, and the version of IIS on the server, you may need to make configuration changes to IIS.

Tip: See the online help for information about configuration changes for IIS.

7. In Visual Studio, choose **DevPartner > Start Without Debugging with Memory Analysis**, or click **Start with Memory Analysis** on the DevPartner toolbar.

   Use the memory analysis session control window to select which of the following types of analysis to perform:

   - ◇ Memory Leaks
   - ◇ Temporary Objects
   - ◇ RAM Footprint

   For further guidance on selecting the type of analysis to perform, see "Identifying Memory Problems" on page 154.

8. In the Session Control window, select a server process for which you want to collect data. Exercise the application from the client and click **View...** to take snapshots of the managed heap as desired. You can select another server process in the same session and take additional snapshots if desired.

9. DevPartner collects memory data for the ASP.NET or IIS process as you exercise the client. No data is collected for the Internet Explorer (client) process.

10. Use the **Session Control** buttons on the memory analysis Session Control window to control data collection. As an alternative, you can use a session control file or the Session Control API to automate data collection.

If You Get Unexpected File Save Dialogs or Saved Session Files

Under certain circumstances, you may see an unexpected File Save dialog box after quitting an ASP.NET application, or find that unexpected session files have been saved if you have configured DevPartner to automatically save session files.

In memory analysis sessions, DevPartner does not collect data for Internet Explorer. (DevPartner collects memory analysis data only for managed code.) Thus, the ASP.NET worker process (`w3wp` or `aspnet_wp`) becomes the primary profiled process when running memory analysis on an ASP.NET application. DevPartner stops data collection and generates a final session file whenever the primary profiled process ends. In most cases, this occurs in response to a user action. However, the ASP.NET worker process can also shut down automatically during profiling if you have edited the process **Model Attributes** section of the **`machine.config`** file on the system on which the process runs in one of the following ways:

- Changed the value of the `requestLimit` or `requestQueueLimit` attribute from "Infinite" to a value low enough to cause the process to be shut down during the session
- Changed the value of the `timeout` or `idleTimeout` attribute from "Infinite" to a value low enough to cause the process to be shut down during the session
- Changed the value of the `memoryLimit` attribute to a percentage low enough to cause the process to recycle during the session

When the process is shut down, DevPartner takes a final snapshot, generates a session file, and ends the session. If IE (the client process started by the user) is still active, the IE process can spawn new instances of the ASP.NET worker process. Each of these ASP.NET worker processes generates a session file when it terminates, resulting in a saved session file, or a File Save dialog. However, this session data is not part of the original memory analysis session, and is usually of little value.

To remedy this situation, you can edit the **`machine.config`** file and set the limiting attribute to a value high enough to prevent premature termination of the process.

Note: Always make a backup copy before editing the **`machine.config`** file.

DevPartner continues to collect analysis data whenever the ASP.NET worker process runs and terminates until you explicitly disable analysis under **DevPartner Coverage, Memory, and Performance Analysis** in the Visual Studio properties window.

If You Get a Security Exception

When attempting to collect data for a managed application, a security exception message displays if your security policy prevents DevPartner instrumentation of your code. By default, assemblies must have the `SkipVerification` permission to be profiled. If you remove this permission from the permission set of the policy under which the code executes, or add imperative security declarations to the assembly that cause this permission to be revoked, the assembly can not be profiled.

To remedy this condition, enable secure profiling in one of two ways.

- Set the following global environment variable and retry profiling the application:

  ```
  NM_NO_FAST_INSTR=1
  ```

  This solution allows you to work around this issue, although it does exact a slight performance penalty.

- Change the policy for the assembly using the .NET Framework Configuration tool MMC snap-in, or by temporarily removing any imperative security declarations in the assembly.

See the *.NET Framework Developers Guide* in the Visual Studio online help for more information on security policy in Visual Studio.

Using Memory Analysis In Your Development Cycle

You do not have to wait until you suspect that you have problem to begin testing. If you run DevPartner memory analysis early and often, and know what to look for when you analyze your application, you can correct problems early, at a point when problems are both easier to identify and require less risk to fix.

Memory problems in managed applications are often the result of larger design and architecture decisions, rather than simple coding errors. For example, one source of memory loss is an object that is not collected because of an out-dated reference to it that is not freed. This can be the result of revisions made in another part of the code. The later these problems are identified in the development cycle, the more difficult and expensive they are to fix.

As a result, memory analysis is valuable as part of a continuous testing program throughout the development cycle. Using memory analysis during unit testing provides an understanding of how the individual modules handle memory. Once you identify and fix areas that need improvement, retest to verify the fix. Then, as you integrate the modules into your application, repeat your memory testing again to ensure that new memory problems do not appear.

Submitting Data to Visual Studio Team System

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected method. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected method.

You can submit data for a method selected in any method list view in a DevPartner memory analysis session file as a **Work Item**. Valid work items include a selected method on the following analysis types:

- Memory Leaks - methods that allocate the most leaked memory
- RAM Footprint - methods that allocate the most memory
- Temporary Objects - methods that use the most memory and entry points that allocate the most memory.

When you submit a Work Item, DevPartner populates the appropriate work item type form with data from the visible columns in the view. To change the method data you submit in the **Work Item**, right-click any column header and select **Choose Columns...** from the context menu.

For more information about DevPartner Studio integration with Visual Studio Team System, “Visual Studio Team System Support” on page 20.

Chapter 6

Automatic Performance Analysis

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with performance analysis. The second section provides reference information for an in-depth understanding of DevPartner Studio's performance analysis feature.

See the DevPartner Studio online help for more task information about performance analysis.

What is Performance Analysis?

DevPartner Studio's performance analysis feature allows you to find bottlenecks that slow down the performance of 32-bit and 64-bit applications, regardless of whether the bottleneck is in your code, in third party components, or in the operating system.

DevPartner performance analysis:

- analyzes performance as your components are really used, even if the components are on distributed systems.
- allows you to target data collection on a specific phase of your application, so you can focus your performance tuning efforts.
- can distinguish between time spent in threads of your application and time spent in threads of other running applications, so you get accurate, reproducible results that are independent of outside influences.

Using Performance Analysis Out of the Box

The following Ready, Set, Go procedure introduces you to using DevPartner to analyze code performance.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject being described in the shaded box, read the additional text following the box.

Note: Analyzing an application with DevPartner Studio does not require elevated system privileges. The system privileges you use to create and debug your application are sufficient for DevPartner to analyze the application.

Ready: Consider What You Want to Analyze

Before using performance analysis, consider what you want to analyze.

The following procedure assumes:

- You are testing a single-process, managed application.
- You can build and run your application.
- Your solution includes a startup project.

Note: Refer to "DevPartner Studio Supported Project Types" on page 267 for a comprehensive list of supported project types for DevPartner performance analysis.

When analyzing your applications, decide what data you are interested in collecting before beginning your performance session. In some cases, there are steps you need to take before beginning a session. For example, some set-up would be required if:

- there are modules you want to omit from the performance analysis
- there are unmanaged modules that you would like analyzed
- you want to include code run on a remote server

For this procedure, all managed, local code in your application is analyzed.

Set: Properties and Options

Once you have decided what code you want to analyze, you can set several properties and options to focus your data collection.

For this procedure, you can use the default DevPartner properties and options. No additional set-up is required.

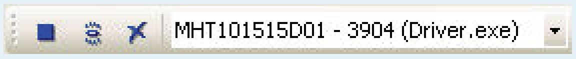
Using Solution Properties and Project Properties, you can choose whether your analysis session data should include information for .NET assemblies, COM that runs outside your application, time spent in threads of other running applications, line-level or method-level analysis, and so on. Using DevPartner options, you can change display options, exclude parts of your application from analysis, or create a session control file to manage data collection. Refer to "Setting Properties and Options" on page 185 if you would like more information about customizing your settings.

Go: Collect Performance Data

After considering what you want to analyze and setting the appropriate properties and options, you are ready to collect performance data.

1. From Visual Studio, open the solution associated with your application.
2. Select **DevPartner > Start without Debugging with Performance Analysis** to begin a performance analysis session.

   During a session, the **Session Control Toolbar** options are active.

   DevPartner session controls let you focus your performance analysis on any phase of your application. You can use the session controls to stop data collection, take a snapshot of the data currently collected and then continue recording, or clear data that has been collected but not yet saved in a snapshot.
3. Run the code you want to analyze.
4. Click the **Snapshot** icon . (Click twice if necessary to bring focus to the session window.) When you take a snapshot, DevPartner creates a file containing the collected data, called a session file, and displays the session file data.
5. Return to your application and continue running your tests.
6. When you are finished running your tests, exit your application. The final session file displays in Visual Studio.

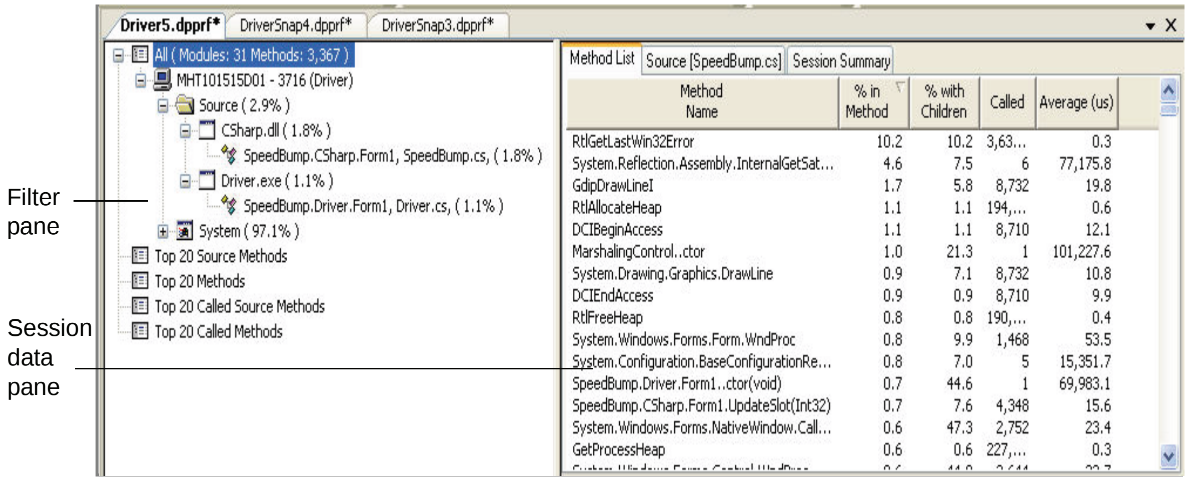
Note: If a security exception message displays when you attempt to collect data for a managed application, refer to page 189 for information about changing your security policy.

You can analyze performance with or without debugging. Generally, run performance analysis without debugging as results from non-debug sessions are easier to interpret. If you run your application in the debugger, some timing values might be larger than expected, especially if breakpoints were hit during the session.

Analyze the Data

When you take a snapshot or exit your application, DevPartner displays the session file in Visual Studio, as shown in Figure 6-1. The session window consists of:

- The filter pane, which lists the source files and images in your application. The filter pane shows the time spent in each file as a percentage of the time spent in the session. The filter pane also provides a set of filters you can use to focus on the most significant data.
- The session data pane, which contains the **Method List**, **Source**, and **Session Summary** tabs. The session data pane displays data for the file or filter selected in the filter pane.



- All (Modules: 31 Methods: 3,367)
  - MHT101515D01 - 3716 (Driver)
    - Source (2.9%)
      - CSharp.dll (1.8%)
        - SpeedBump.CSharp.Form1, SpeedBump.cs, (1.8%)
      - Driver.exe (1.1%)
        - SpeedBump.Driver.Form1, Driver.cs, (1.1%)
    - System (97.1%)
  - Top 20 Source Methods
  - Top 20 Methods
  - Top 20 Called Source Methods
  - Top 20 Called Methods

Method Name	% in Method	% with Children	Called	Average (us)
RtlGetLastWin32Error	10.2	10.2	3,63...	0.3
System.Reflection.Assembly.InternalGetSat...	4.6	7.5	6	77,175.8
GdipDrawLineI	1.7	5.8	8,732	19.8
RtlAllocateHeap	1.1	1.1	194,...	0.6
DCIBeginAccess	1.1	1.1	8,710	12.1
MarshalingControl..ctor	1.0	21.3	1	101,227.6
System.Drawing.Graphics.DrawLine	0.9	7.1	8,732	10.8
DCIEndAccess	0.9	0.9	8,710	9.9
RtlFreeHeap	0.8	0.8	190,...	0.4
System.Windows.Forms.Form.WndProc	0.8	9.9	1,468	53.5
System.Configuration.BaseConfigurationRe...	0.8	7.0	5	15,351.7
SpeedBump.Driver.Form1..ctor(void)	0.7	44.6	1	69,983.1
SpeedBump.CSharp.Form1.UpdateSlot(Int32)	0.7	7.6	4,348	15.6
System.Windows.Forms.NativeWindow.Call...	0.6	47.3	2,752	23.4
GetProcessHeap	0.6	0.6	227,...	0.3

Figure 6-1. Performance Analysis Session window

Using the Filter Pane and the Session Data Pane

To begin evaluating your data, start by using a filter and examining the **Method List** to find methods that occupied a significant percentage of your program's processing time.

1 In the **Filter** pane, click on the **Top 20 Source Methods** filter. This reduces the displayed data and help you focus on your source methods.

Knowing the time spent in system files is useful when assessing performance, but using this filter to eliminate system files from the display can help you focus your performance tuning efforts.

2 Examine the data on the **Method List** tab. The **Method List** tab displays information about the amount of time spent in each method.

Scanning the **Method List** for methods with high values in one or more of the columns helps target specific areas for performance improvement.

3 On the **Method List** tab, look at the **% in Method** column, which shows the time spent in the method as a percentage of the time spent in the session. (By default, the data is sorted in descending order by the **% in Method** column. If not, click the column header.)

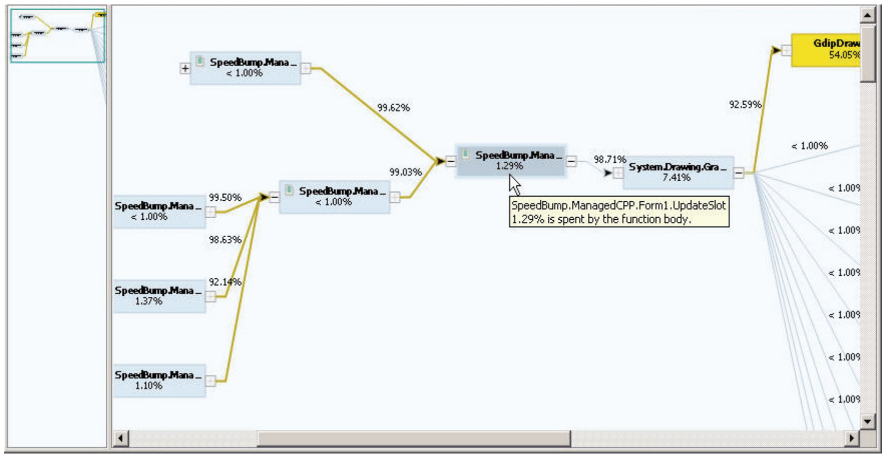
4 Look at the **% with Children** column, which shows time spent in the method and its child methods as a percentage of the time spent in the session.

5 Look at the **Average** column, which shows the average execution time of the method.

Examine the values in these columns helps determine which areas of your code to target for improvement.

Viewing a Call Graph

Some performance issues become apparent only when seen in the context of the calls made between parent and child methods. In these cases, examining a Call graph can be helpful. A Call graph is a graphical representation of the calling relationships of your application's methods.



SpeedBump.ManagedCPP.Form1.UpdateSlot
1.29% is spent by the function body.

Figure 6-2. The Call graph

You can access the Call graph from the context menu in the Method List**,** **Source** tab, or from the **Call Graph** icon .

6 Right-click on a method in the Method List and choose **Go to Call Graph** on the context menu. The method's Call graph displays, with the critical path highlighted in your default system color.

Tip: The following steps are an introduction to using Call graphs. More information about Call graphs is presented in "Analyzing a Call Graph" on page 197.

7 Click the plus/minus icons at the edges of any node to expand or collapse the view of parent (left) or child (right) nodes. Compare the percentage figure shown in a node with the percentage shown on the lines to child nodes to follow the path(s) that might be causing performance problems.

8 Hover the mouse-pointer over a node or over the percentage value on a link between method nodes to view a more detailed description.

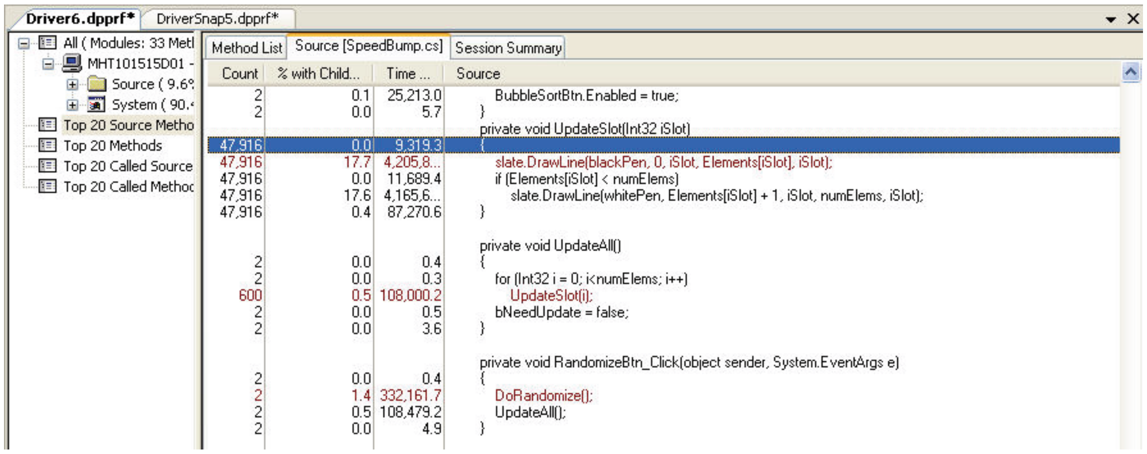
9 Identify a method that is a target for potential performance improvement. Right click and select **Go to Method Source** from the context menu. (If you select a system method, the source is not available.)

The method's source code is displayed on the **Source** tab in the session window, but the call graph still has focus.

10 Close the Call graph to begin working with the source code.

Viewing Source Code

The **Source** tab displays the source code for the selected file or method. Use the **Source** tab to help you identify the lines of code that might be causing performance problems.



```
 Count  % with Child...    Time ...  Source
     2            0.1    25,213.0      BubbleSortBtn.Enabled = true;
     2            0.0         5.7    }
                                     private void UpdateSlot(Int32 iSlot)
47,916            0.0     9,319.3    {
47,916           17.7   4,205,8...     slate.DrawLine(blackPen, 0, iSlot, Elements[iSlot], iSlot);
47,916            0.0    11,689.4      if (Elements[iSlot] < numElems)
47,916           17.6   4,165,6...       slate.DrawLine(whitePen, Elements[iSlot] + 1, iSlot, numElems, iSlot);
47,916            0.4    87,270.6    }

                                     private void UpdateAll()
     2            0.0         0.4    {
     2            0.0         0.3      for (Int32 i = 0; i<numElems; i++)
   600            0.5   108,000.2        UpdateSlot(i);
     2            0.0         0.5      bNeedUpdate = false;
     2            0.0         3.6    }

                                     private void RandomizeBtn_Click(object sender, System.EventArgs e)
     2            0.0         0.4    {
     2            1.4   332,161.7      DoRandomize();
     2            0.5   108,479.2      UpdateAll();
     2            0.0         4.9    }
```

Figure 6-3. The Source tab

11 The source code for the selected method is displayed on the **Source** tab, as shown in Figure 6-3. The **Source** tab shows performance session data about each executed code line.

DevPartner highlights the slowest line in each method. Determine if there is an opportunity for performance improvement and modify the code accordingly.

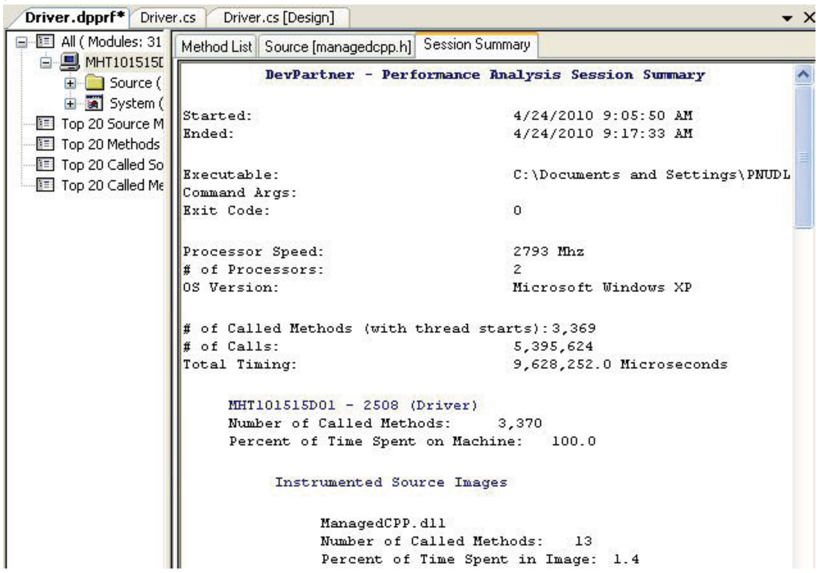
Comparing Sessions

After locating and correcting performance problems, you can run another performance session and compare the session files from before and after your changes. DevPartner displays a comparison window showing the differences between the sessions. For more information about comparing sessions, refer to "Comparing Sessions" on page 200.

Viewing Session Summary Data

The **Session Summary** tab displays a synopsis of the performance analysis session.

.



```
          DevPartner - Performance Analysis Session Summary

Started:                               4/24/2010 9:05:50 AM
Ended:                                 4/24/2010 9:17:33 AM

Executable:                            C:\Documents and Settings\PNUDL
Command Args:
Exit Code:                             0

Processor Speed:                       2793 Mhz
# of Processors:                       2
OS Version:                            Microsoft Windows XP

# of Called Methods (with thread starts):3,369
# of Calls:                            5,395,624
Total Timing:                          9,628,252.0 Microseconds

     MHT101515D01 - 2508 (Driver)
     Number of Called Methods:     3,370
     Percent of Time Spent on Machine:   100.0

          Instrumented Source Images

               ManagedCPP.dll
               Number of Called Methods:    13
               Percent of Time Spent in Image:  1.4
```

Figure 6-4. The Session Summary tab

12 Click on the **Session Summary** tab.

The Session Summary includes contextual information about the session, such as the date and time of the session, the processor speed and operating system, and so on. This information can be useful when viewing an older session file, particularly one that was created by someone else.

The summary also includes performance data from the filter pane and the **Method Lis**t tab, showing data for both the files and the methods that were analyzed.

13 Scroll through the tab to view the session summary data.

Saving Session Files

When you have finished reviewing performance data you can save the session file.

1 Close the session file window in Visual Studio. DevPartner prompts you to save the session file.

2 Click **Ok** to accept the default file name and location.

DevPartner saves session files as part of the active solution. They appear in the **`DevPartner Studio`** virtual folder in Solution Explorer. Performance analysis session files take the **`.dpprf`** extension.

By default, DevPartner physically saves the session files in your project's output folder. DevPartner automatically increments the file name based on the contents of the default folder (for example, **`MyApp.dpprf`**, **`MyApp1.dpprf`**, and so on). If you save session files to a location other than the default folder, you must manage the file naming and numbering.

For projects that do not have an output folder, such as a Visual Studio 2005 Web site project, DevPartner physically saves the files to the project folder.

Session files generated from the command line are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

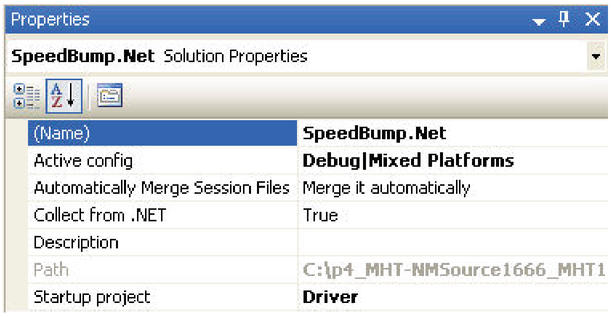
This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a performance analysis session, continue reading the rest of this chapter for additional information, or refer to the DevPartner online help for task-based information.

Setting Properties and Options

Before beginning a performance analysis session, it is often useful to fine-tune data collection to include or omit certain types of information. Use **Solution Properties**, **Project Properties**, and **DevPartner Options** to better focus your analysis session.

Solution Properties

To view performance properties available at the solution level, select the solution in the Solution Explorer and press F4 to view the Properties window.



(Name)	SpeedBump.Net
Active config	Debug|Mixed Platforms
Automatically Merge Session Files	Merge it automatically
Collect from .NET	True
Description	
Path	C:\p4_MHT-NMSource1666_MHT1
Startup project	Driver

Figure 6-5. Solution properties

The following solution property affects performance analysis:

- **Collect from .NET -** Visible only for managed code applications. Set this property to false if you do not want DevPartner to collect information for .NET assemblies.

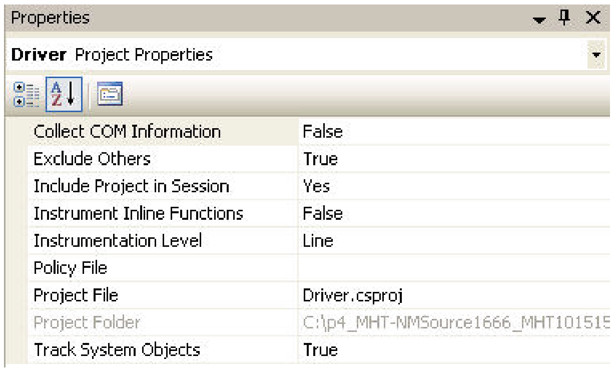
  This property affects only coverage analysis and performance analysis sessions. Memory analysis and Performance Expert always collect data from managed applications, even when this value is set to false.

  The **Collect from .NET** property is not available with DevPartner for Visual C++ Boundschecker Suite.

- **Startup project -** Your solution must include a startup project. If the solution contains multiple startup projects, before analysis begins DevPartner prompts you to choose a startup project for the session.

Project Properties

To review project level properties, select a project in the Solution Explorer and review the properties that can be set for projects within the solution.



Collect COM Information	False
Exclude Others	True
Include Project in Session	Yes
Instrument Inline Functions	False
Instrumentation Level	Line
Policy File	
Project File	Driver.csproj
Project Folder	C:\p4_MHT-NMSource1666_MHT101515
Track System Objects	True

Figure 6-6. Project properties

The following project properties affect performance analysis:

- **Collect COM Information -** DevPartner collects method level data based on DLL exports and COM interfaces. Select False if you do not want DevPartner to collect information for COM that runs outside your application.
- **Exclude Others -** Excludes time spent in threads of other running applications. The resulting session data includes only time spent in threads of your application.

  While collecting performance information, DevPartner monitors context switching and tracks how much of the CPU time is spent working in threads outside of the application. After collecting the timing data, DevPartner subtracts the time spent in other threads from the clock time to determine exactly how much time was spent in your application.

  Select True to enable this feature; select False to disable it.
- **Instrument Inline Functions -** Set this property to True to instrument inline functions. (Instrumentation is described in "About Instrumentation" on page 188.) Inline functions are not instrumented by default if inline optimizations are enabled.

  When working with managed C++ applications, DevPartner does not collect data for functions explicitly inlined with the `__forceinline` keyword, even if you choose **True** for the **Instrument Inline Functions** property.
- **Instrumentation Level -** Choose **Method** or **Line**. (Instrumentation is described in "About Instrumentation" on page 188.)
  - **Method**: Method-level instrumentation allows your performance analysis session to run faster, but provides only method-level data.
  - **Line**: Line-level instrumentation enables you to drill down to specific lines in your source code. When working with .NET applications, if you choose Line as your Instrumentation Level and install a JIT-compiled assembly in the global assembly cache (GAC), DevPartner performance analysis cannot provide line-level data about the assembly. DevPartner is unable to instrument the JIT-compiled assembly. To collect line-level data, do not pre-JIT assemblies when running performance analysis.

All property settings persist unless you explicitly change them.

Options

To review DevPartner option settings for performance analysis sessions, choose **DevPartner > Options > Analysis.**

- The **Display** option allows you to set the precision, scale, and units used when displaying your data.
- The **Exclusions** option allows you to omit one or more images from data collection. Refer to "Excluding Images" on page 188 for more information on exclusions.
- The **Session Control File** option allows you to create a set of rules and actions to control the data that DevPartner collects as your application or module runs. Refer to "Analysis Session Controls" on page 303 for more information about session control files.

Other Visual Studio options, such as the **Environment > Fonts and Colors** options, also affect DevPartner features.

Excluding Images

When you run an application under performance analysis, DevPartner collects data for all source and system images. However, you can use the Exclusions option to omit one or more images from analysis.

While viewing Analysis Options (DevPartner **>** **Options** **>** **Analysis**) select **Exclusions - Performance**.

From the **Show** list at the top of the page, select one of the following:

- Global exclusions
- Local exclusions in current user folder
- Local exclusions in executable folder

The **Local exclusions in current user directory and Local exclusions in executable directory** options are available only when a solution is open and the executable folder differs from the current working folder.

Click **Insert** to add an image to the exclusion list. Type a name, or browse to the image you want to exclude. Allowable file types for exclusion are **`.exe`**, **`.dll`**, **`.ocx`**, and **`.netmodule`**. Use the **Files of type** list to limit the types of files displayed.

If you choose a .NET module (**`.netmodule`**), only the unmanaged parts of the module are excluded.

To remove an image from the list of exclusions, select the item and click **Delete**.

Select the **Exclude system images check-box** to exclude uninstrumented system DLLs from DevPartner performance profiling.

Global exclusions are saved in **`nmexclud.txt`** in the **`\Analysis`** sub-folder in the DevPartner installation folder. Local exclusions are saved in **`nmexclud.txt`** in the application executable folder or in the current working folder. To save a copy of the exclusion list (**`nmexclud.txt`**) to another location, click **Save To**.

Note: To fully monitor a running application, DevPartner always profiles a few specific Win32 APIs. As a result, certain system DLLs cannot be excluded individually and always appear in the System Images list of the session file, unless you select **Exclude system images** to exclude all system images.

Exclusions do not apply to files compiled with **Native C/C++ Instrumentation**. For example, if you attempt to exclude an instrumented unmanaged C/C++ image, DevPartner still collects information for that file, although no system call information is collected. If you wish to exclude an unmanaged C/C++ image from data collection, do not instrument that image.

About Instrumentation

When you run a managed application, DevPartner inserts hooks into the byte code for each assembly as it is loaded by the compiler, a process called instrumentation. This code contains instructions that DevPartner uses to collect performance data while your application is running. DevPartner instrumentation does not change the actual files on disk; it only modifies the in-memory representation of files as they execute.

Unlike managed code, which DevPartner instruments at runtime, you must instrument unmanaged C/C++ code when compiling. To instrument unmanaged code, DevPartner inserts hooks directly into your source code. Use the Instrumentation Manager in which you specify the type of instrumentation to be used and specify any projects in the solution to exclude from instrumentation. (Refer to "Collecting Data from Unmanaged Code" on page 190 for more information about the Instrumentation Manager.) When you rebuild the unmanaged project, the hooks are inserted. To remove the hooks, turn off instrumentation by deselecting the Native C/C++ Instrumentation option from the DevPartner menu, and rebuild the project.

Collecting Data from Various Types of Applications

This section provides information about using DevPartner performance analysis to collect data from different types of applications.

DevPartner supports all Visual Studio managed code languages, as well as unmanaged C/C++. DevPartner can also collect performance data for JScript and VBScript Web applications when using Internet Explorer or IIS.

Refer to Appendix A, "DevPartner Studio Supported Project Types" for a complete list of languages and project types supported in each version of Visual Studio.

Collecting Data From Managed Code

Many applications developed in Visual Studio are managed applications, such as C#, Visual Basic, and managed C++ applications.

DevPartner requires PDB (program database file) information to collect detailed information about your managed application source code. If no source data appears on the Source tab or source files do not appear in the Filter pane make sure **`.pdb`** files are being generated.

Managed application files for which no PDB information is available appear in the System folder in the **Filter** pane.

When attempting to collect data for a managed application, a security exception message appears if your security policy prevents DevPartner instrumentation of your code. By default, assemblies must have the `SkipVerification` permission to be profiled. If you remove this permission from the permission set of the policy under which the code executes, or add imperative security declarations to the assembly that cause this permission to be revoked, the assembly can not be profiled.

To remedy this condition, enable secure profiling in one of two ways.

- Set the following global environment variable and retry profiling the application:

  ```
  NM_NO_FAST_INSTR=1
  ```

  This solution allows you to work around this issue, although it does exact a slight performance penalty.

- Change the policy for the assembly using the .NET Framework Configuration tool MMC snap-in, or by temporarily removing any imperative security declarations in the assembly.

See the *.NET Framework Developers Guide* in the Visual Studio online help for more information on security policy in Visual Studio.

Collecting Data from Unmanaged Code

When you build your unmanaged C++ application for performance profiling with **Native C/C++ Instrumentation**, DevPartner works with the compiler to add instructions to your application image to collect performance data at run time.

Use the DevPartner Native C/C++ Instrumentation Manager to choose the type of instrumentation that DevPartner applies at compile time. Instrumentation is the process of adding instructions to an image. You must instrument native (unmanaged code) C/C++ projects before you can collect data with DevPartner.

For Visual Studio 2010 and later, instrumentation settings apply only to the active solution build configuration. This means that for each solution, you can have seperate instrumentation settings to build debug and release versions of the application. This feature provides the ability to batch build all the required build configurations into your build process. You must apply your instrumentation settings project otherwise default instrumentation settings apply.

For Visual Studio 2010 and later, instrumentation settings are saved to the **`project.vcxproj.user`** files, which are in the same location as the project files.

To instrument unmanaged code:

1. Open the solution that contains the unmanaged C/C++ project for which you want to collect data.
2. If using Visual Studio 2010 or later, optionally use the Configuration Manager to select a build configuration. See Microsoft help for information on creating and editing solution build configurations.
3. Choose DevPartner > **Native C/C++ Instrumentation Manager**.
4. Select the **Instrument the projects checked below when I build my solution** check box and select a type of instrumentation**.** The type of instrumentation you choose must match the type of analysis you subsequently run.
5. Select the projects to be instrumented. By default, DevPartner instruments all unmanaged code in the solution. Clear the check boxes of modules to be omitted.Clear the check boxes of modules to be omitted. Click **OK**.
6. If creating instrumentation settings for another build configuration in the solution, select the build configuration, then repeat the steps to create the instrumentation settings for that build configuration.
7. Rebuild the solution. DevPartner instruments the selected unmanaged C/C++ projects. Click **Start with Performance Analysis** on the DevPartner toolbar to begin the analysis session.

DevPartner saves project selections in the **Native C/C++ Instrumentation Manager** with the solution. Once you use the Instrumentation Manager to configure instrumentation, you can turn instrumentation on and off with the **Native C/C++ Instrumentation** option from the DevPartner menu or the **Native C/C++ Instrumentation** button on the DevPartner toolbar. Use the **Native C/C++ Instrumentation Manager** only to change settings.

To remove instrumentation from your application, deselect the **Native C/C++ Instrumentation** option from the DevPartner menu. The next time you start a performance analysis session or rebuild the solution, Visual Studio rebuilds the solution without instrumentation.

Note: If your application calls Visual Studio components, you must compile these components with DevPartner instrumentation for performance analysis in Visual Studio. See the DevPartner Studio online help in Visual Studio for more information.

Mixed-mode C++ Files

With unmanaged (native) C++, you can compile your application as managed code with the `/clr` option, but mark sections of your code with `#pragma` (native). The compiler generates native code for any methods defined in the `#pragma` section. DevPartner does not support mixed-mode C++ files. When profiling a program that includes a C++ file with both managed and unmanaged (native) sections, DevPartner collects coverage data only for the managed code portions, not the native code portions from `#pragma`. To collect data for unmanaged C++ code, place the unmanaged code in a separate file and instrument it, "Collecting Data from Unmanaged Code" on page 190.

Collecting Data from Multiple Processes

An application may run more than one process. For example, when you profile an ASP.NET application you may see the browser process (`iexplore`), the IIS process (`inetinfo`), and the ASP worker process (`aspnet_wp or w3wp`).

When you run a multi-process application under performance analysis, the DevPartner **Session Control** toolbar displays the active processes in the process selection list.

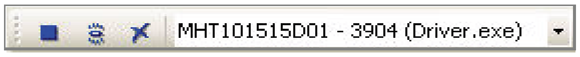


Figure 6-7. Session Control Toolbar with the Process Selection List

Use the process selection list to focus data collection. When you take a snapshot, DevPartner creates a session file with data for the process selected in the process selection list.

Collecting Data from Remote Systems

You can collect performance data for application components running on remote systems. For example, you might want to collect performance data for both client and server portions of a client/server application. With DevPartner, you can collect performance data for client and server processes as you run the client application.

To collect data simultaneously from a client system and a remote system, install DevPartner on the client and install DevPartner and the DevPartner Remote Server license on the remote system. See the *DevPartner Studio Installation Guide* and the *Distributed License Management Licensing Guide* for more information about the Remote Server license.

A server connected through a Terminal Services connection does not require the DevPartner Remote Server license. See "Using Terminal Services and Remote Desktop" on page 21 for information on Terminal Services.

On the remote system, select the relevant projects and review the DevPartner properties to ensure that they match the options set on the client system. DevPartner restarts server processes, such as IIS, after you change options. This restart is necessary for changes to take effect.

Be sure to specify instrumentation if you are analyzing an unmanaged C++ application. If your application calls unmanaged C++ components, you must instrument those components if you want to collect data from them, as described in "Collecting Data from Unmanaged Code" on page 190.

Correlating Data

When you use Internet Explorer (IE) and Internet Information Server (IIS) as browser and Web server, or you use COM to make inter-process calls, DevPartner automatically recognizes a client/server relationship between the processes. To preserve the relationship between the methods of DCOM objects or the relationship between HTTP client and server (IE and IIS), DevPartner automatically correlates the data from those sessions. It then combines the correlated data with the client session data into a single session file.

The correlated session file contains the performance data for both the client and server portions of your application. The correlated session file appears in Visual Studio, like any other session file, with **`_co`** appended to the file name, as in **`appname_CO.dpprf`**.

When you view a correlated session file in the **Call** graph, you can follow a COM call stack from the calling method to the called method. DevPartner scales the server-side data to match the clock speed of the client system.

You can use **DevPartner** > **Correlate** > **Performance Files** to manually combine data from different session files when there is no COM-based relationship or client/server relationship between IE and IIS. You can also use the `NMCORRELATE` command line utility to manually combine data, as described in "Starting Analysis from the Command Line" on page 285.

Collecting Data From .NET Web Applications

If you develop Web Forms, XML Web Services, or ASP.NET applications, you can use DevPartner to collect performance data for both client and server portions of your application. You can configure DevPartner to collect data for IIS and ASP.NET running on a local or remote computer.

To collect data for unmanaged C++ components called by your application, you must instrument and rebuild the objects with **Native C/C++ Instrumentation**, as described in "Collecting Data from Unmanaged Code" on page 190. If your Web application calls C++ components, you must instrument them using the DevPartner commands in Visual Studio. Be sure to instrument for performance analysis. DevPartner collects data for only one analysis type in a session.

Note: DevPartner session files are saved with the current solution. Opening a Web project from IIS directly, as opposed to opening the project through Visual Studio, may cause a different solution file to be used. DevPartner session files created in the first solution would not be visible in the second solution.

Prerequisites

For DevPartner performance analysis to successfully profile an ASP.NET application, the following two conditions must be met:

- The project must include a **`web.config`** file.

- The **`web.config`** file must include a compilation element with the debug attribute set to true. For example:

  ```
  <compilation debug="true"/>
  ```

DevPartner can also collect data for in-process or out of process components called by your application.

Analyze ASP.NET Applications without Debugging

For optimum results, run performance analysis without debugging.

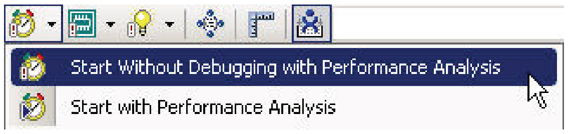

Start Without Debugging with Performance Analysis
Start with Performance Analysis

Figure 6-8. Start without debugging option

Only one script debugger can be active at one time. If you debug a Web application with debugging, both Visual Studio and DevPartner attempt to load a script debugger. A message displays indicating that the script debugger failed to attach to IE. The session continues without interruption despite the error message.

To avoid the error message, you can either disable script debugging in `iexplore` or run performance analysis without debugging.

Unexpected File Save Dialogs or Saved Session Files

Under certain circumstances, you may see an unexpected File Save dialog box after quitting an ASP.NET application, or find that unexpected session files have been saved if you have configured DevPartner to automatically save session files.

When you run performance analysis on an ASP.NET application, DevPartner collects data for Internet Explorer as the primary profiled process. DevPartner also saves session data for secondary processes, such as an ASP.NET worker process (`w3wp or aspnet_wp`). When the primary process terminates, DevPartner stops data collection and generates a final correlated session file that contains both client data (for IE) and server data (for IIS and ASP.NET) worker processes. You can also take a snapshot of the server process alone by selecting the process in the Session Control toolbar.

In most cases the client and server processes are terminated by user action. However, the ASP.NET worker process can also shut down automatically during profiling. This can occur if you have edited the `processModel` Attributes section of the **`machine.config`** file on the system on which the process runs in one of the following ways:

- Changed the value of the `requestLimit` or `requestQueueLimit` attribute from "Infinite" to a value low enough to cause the process to be shut down during the session
- Changed the value of the `timeout` or `idleTimeout` attribute from Infinite to a value low enough to cause the process to be shut down during the session
- Changed the value of the `memoryLimit` attribute to a percentage low enough to cause the process to recycle during the session

When the process is shut down, DevPartner takes a final snapshot and generates a session file. DevPartner handles the session file in one of the following ways:

- If the ASP.NET worker process is the selected process in the Session Control toolbar, DevPartner opens the session file in Visual Studio and adds it to the solution. This action is repeated for each instance of the ASP.NET worker process that is spawned and terminated.
- If the ASP.NET worker process is not the selected process, the session file is cached. When the IE client process is terminated, or when a snapshot of the IE process is taken, DevPartner creates a session file for IE, and a correlated session file that includes data for IE, IIS, and all instances of the ASP.NET worker process spawned and terminated up to that point.

When the analysis session has ended, DevPartner continues to display the File Save dialog box or automatically save session files for instances of the ASP.NET worker process that are spawned and terminated.

To avoid generation of extra session files due to frequent termination of the ASP.NET worker process, you can edit the **`machine.config`** file and set the limiting attribute to a value high enough to prevent premature termination of the process.

Note: Always make a backup copy before editing the **`machine.config`** file.

Collecting Data from Classic Web Script Applications

When you run a classic Web script application with DevPartner performance analysis enabled, DevPartner gathers data for HTML files and JScript and VBScript source files. If the scripting languages invoke in-process or out-of-process components, such as COM objects, DevPartner can collect data for these as well.

Instrumentation for the scripting languages occurs at run-time, just as it does for managed .NET languages. However you do need to instrument any unmanaged components, such as COM objects, that you want monitored.

Note: The following procedure is unique to classic Web script applications. To collect data for Web Forms, XML Web services, and ASP.NET applications you develop in Visual Studio, run the application just as you would run any other .NET application.

To collect data for a classic Web script application, choose **Start** > **Programs** > **Micro Focus** > **DevPartner Studio** > **Utilities** > **Web Script Performance.**

Internet Explorer (IE) opens with DevPartner Performance Analysis loaded. In addition to IE, a Session Control toolbar appears, which you can use to control data collection.

In the DevPartner-enabled instance of IE, open the HTML page or Web application for which you want to collect performance data and exercise the application. Optionally, use the Session Control toolbar to focus data collection as the application executes.

Exit Internet Explorer or, if using the Session Controls, execute a **Stop** action. The Save Session dialog box appears and the session file is saved.

Web Application Data Collection Tips

Before you begin collecting data for analysis:

- *Warm up* the application by exercising it for several minutes. Be sure to include the parts of the application in which you are interested.
- Execute the **Clear** session control action to discard data collected to that point. This eliminates data collection for the many one-time initializations that take place when you launch the application.
- Exercise the modules you are analyzing.
- Click **Snapshot** on the **Session Control** toolbar. This provides performance data for a representative sample of your code.
- Allow time for HTML pages to completely load. When testing manually, wait for the page to load. When creating scripts for automated testing, build in wait time so pages can load completely. Executing code on a page before the page is fully loaded may skew your profiling data.
- Be aware of caching. A Web application may return a page from the cache instead of running your application code. If your test uses the same input data repeatedly, caching skews results. If you do not want to measure the effects of the cache, you can turn caching off by editing the **`machine.config`** file and commenting out the line that reads:

```
<add name=”OutputCache” type=”System.Web.Caching.Output-
CacheModule”/>
```

Note: Always make a backup copy before editing the **`machine.config`** file.

Web Service Requirements

For DevPartner performance analysis to detect a Web service, the service must meet at least one of the following requirements:

- The Web service must be derived from the **`System.Web.Services.WebService`** base class.
- The Web service must contain the `WebService` attribute.

For DevPartner performance analysis to detect a Web method, the method must contain the `WebMethod` attribute.

Deleting Temporary Files from NMSource

While analyzing scripts for performance under IE or IIS, DevPartner creates an **`NMSource`** folder to hold temporary copies of the script source. This source is displayed in the Source tab of the Session window when you are analyzing session data.

Because this source may be needed at any time, DevPartner does not delete files from **`NMSource`**. The size of this folder can grow quickly, particularly when you are analyzing server programs under IIS.

You should regularly review the source files in the **`NMSource`** folder and delete any related to projects that are no longer active. **`NMSource`** is located in the **`\Program files\Internet Explorer`** folder.

Configuring IIS for Data Collection

To collect performance data for IIS/ASP.NET applications running on the local computer or on a remote server, set the following configuration options.

If IIS runs on the local system, set the options described below on the local system. If IIS runs on a remote server, you must install DevPartner (and a Remote Server license) on that system and set the options described below on the remote system.

Script Debugging

You can set the following options in the Default Web Site Properties, or in the WebApplication Properties for a specific application, of the Internet Information Services manager. The following options apply to IIS 5.0 or 6.0.

On the Home Directory or Directory tab, click **Configuration**. On the **Application Debugging** tab, set the **Debugging Flags** to:

- Enable ASP server-side script debugging
- Enable ASP client-side script debugging

Host Process Settings

If your Web application runs in the `dllhost` process, you may need to change the Application Protection options to enable DevPartner to collect performance analysis data. You can set these options in the Default Web Site Properties, or in the WebApplication Properties for a specific application, of the Internet Information Services manager. The following options apply to IIS 5.0 or 6.0.

On the Home Directory or Directory tab, in the Application Settings section, set the Application Protection to one of the following:

- Low (IIS Process): Your application runs in the `inetinfo` process. DevPartner restarts IIS when you enable data collection and collects data from this process as your application runs.

- High (Isolated): Your application runs as a separate instance of `dllhost`. DevPartner recognizes the new process and collects data as your application runs.

When you have finished collecting data, restart IIS to remove DevPartner data collection from the process.

Configuring Internet Explorer for Data Collection

To collect performance analysis data from Internet Explorer, select **Tools > Internet Options...** On the **Advanced** tab, set **Disable script debugging (Internet Explorer)** to OFF and set **Disable script debugging (Other)** to Off.

Collecting Data from a Service

To run a performance analysis session for a service, use **`DPAnalysis.exe`**. With **`DPAnalysis.exe`**, you can run sessions directly from the command line or through an XML configuration file. Refer to“Starting Analysis from the Command Line” on page 285 for information on **`DPAnalysis.exe`**.

Collecting Data from COM and COM+ Applications

You can collect data for an application that makes calls to COM or DCOM components with DevPartner.

If you profile an application that uses a mix of unmanaged COM and .NET objects (COM+), DevPartner collects line-level data for .NET portions of the application. DevPartner collects line-level data for unmanaged code components if they have been instrumented with DevPartner native C/C++ instrumentation. DevPartner can also collect line-level data for your Visual Basic COM objects, if you first instrument them for performance data collection. You can do this by building the project with instrumentation for performance analysis.

If you profile a C++ object, or any unmanaged code component that has not been instrumented, DevPartner collects only method-level data based on COM interfaces and DLL exports.

Collecting Data for Recursive Functions

A literal profile of an application that uses recursion contains double counts of recursive functions. DevPartner eliminates this duplication by detecting when it is already timing a function. It stops timing for the first function call and starts a new accumulation for the second call. Refer to the DevPartner Studio online help if you would like an in-depth description of how DevPartner handles collecting data for recursive functions.

Analyzing a Call Graph

A Call graph is a graphical representation of the calling relationships of your application’s methods. Use of call graphs was introduced in the Ready, Set, Go procedure earlier in this chapter. This section provides additional details about using the Call graph.

To view a Call graph from a session file, either click the **Show Call Graph** button or select a method from the **Method List** tab, right-click and select **Go to Call Graph**. A separate Call graph window appears.

DevPartner displays Call graphs showing the chain of calls leading up to a particular method call, and the methods that are subsequently called by that method.

The nodes are displayed sequentially from left to right in the order in which they were called. The first node initially shown in the Call graph is the base node. This represents the selected method or object. Nodes to the left of a node are called “parent nodes.” Nodes to the right of a node are called “child nodes.”

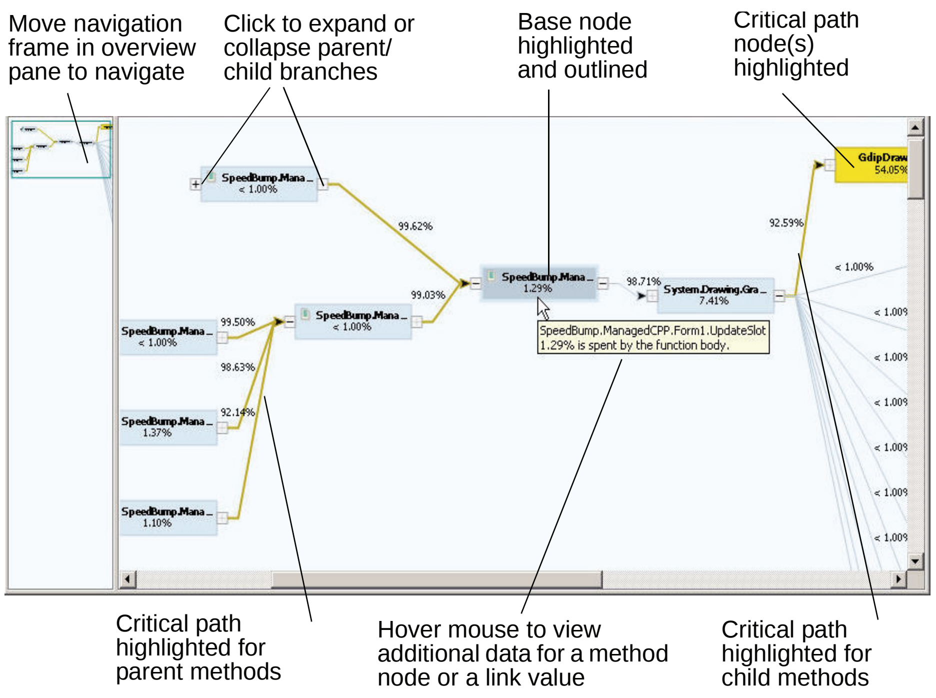

Move navigation frame in overview pane to navigate
Click to expand or collapse parent/ child branches
Base node highlighted and outlined
Critical path node(s) highlighted
SpeedBump.Mana... < 1.00%
99.62%
SpeedBump.Mana... 1.29%
98.71%
System.Drawing.Gra... 7.41%
92.59%
GdipDraw 54.05%
SpeedBump.Mana... < 1.00%
99.03%
SpeedBump.Mana... < 1.00%
99.50%
98.63%
SpeedBump.Mana... 1.37%
92.14%
SpeedBump.Mana... 1.10%
SpeedBump.ManagedCPP.Form1.UpdateSlot
1.29% is spent by the function body.
< 1.00%
Critical path highlighted for parent methods
Hover mouse to view additional data for a method node or a link value
Critical path highlighted for child methods

Figure 6-9. Call graph

The Call graph consists of two frames:

- The left frame shows an overview of the Call graph. This is useful to see the entire Call graph if the Call graph has too many nodes to be shown in the right frame without scrolling. As you expand or collapse nodes in the right frame, the overview automatically refreshes to display the current view. Alternatively, move the navigation frame around in the overview to change the portion of the graph displayed in the right frame. You can close the overview by right-clicking anywhere in the right frame and deselecting the **Show Overview** option.

- The right frame shows the base method node and all the methods either called by it or that call it. Use the expand/collapse boxes to show or hide the nodes to the right or left of the selected node.

  The percentage value shown in each node represents the percentage of time the node is using. The value shown on the lines to each child node represent the time that child path is using, as a percentage of the total time being used by its parent node.

Critical Paths

When you display a Call graph, DevPartner computes the critical path for the selected method and all of its children. The critical path is the sequence of method calls that accounted for the largest percentage of time attributable to the method and all of its child methods.

Navigating the Call Graph

You can drag the nodes to different locations on the window and the Call graph lines are automatically redrawn for you. This is useful if the screen is cluttered with too many methods or if you want to reduce the amount of screen taken up by the initial display of the base, parent, and child nodes.

By default, only the child nodes are shown in expanded form. The parent(s) of the base node are not shown. Click on the plus icon on the left side of the base node to display the parent node for the base node. To display the full path, expand the left icon for each parent node until you reach the first method executed by the program (typically named `Program Start`).

You can select nodes either individually or in a group. To select multiple nodes, select one node, then while pressing the Ctrl or Shift key, select the other nodes you want. You can then drag them as a group.

Viewing Source Code

To view the source code for the base node, right-click on the base node and select the **Go to Method** option on the context menu. You can only view the source code for the base node.

Child-side Analysis

Analyze the child (right) side of the Call graph to understand what to optimize.

Expand the child nodes to analyze whether the base method or a child method is responsible for the most time.

- If the base node has several parallel branches, look for branches that have the largest values on the link to the first child method. Optimizing branches with higher values is likely to provide more benefit in terms of performance.
- If the base method itself shows a high value, consider optimizing the base method.
- If a child branch is a large contributor to the time spent by the base method, look for child nodes on that branch with high percentage values.

Parent-side Analysis

Analyze the parent (left) side of the Call graph to determine if the base node branch is worth optimizing, or number of times the base node is called can be reduced or eliminated.

Expand the parent nodes to the left of the base node. In particular, examine the base node’s contribution to the time spent in its parent branches. This helps determine if optimizing the base node or its child methods is worthwhile. If the base method is a large contributor to several parents, or to an important parent in terms of overall program execution, it is probably worth considering as a target for optimization.

Values on the links between the base node and its parents are independent, not additive. Each percentage value represents the base node’s contribution to the time spent by that parent.

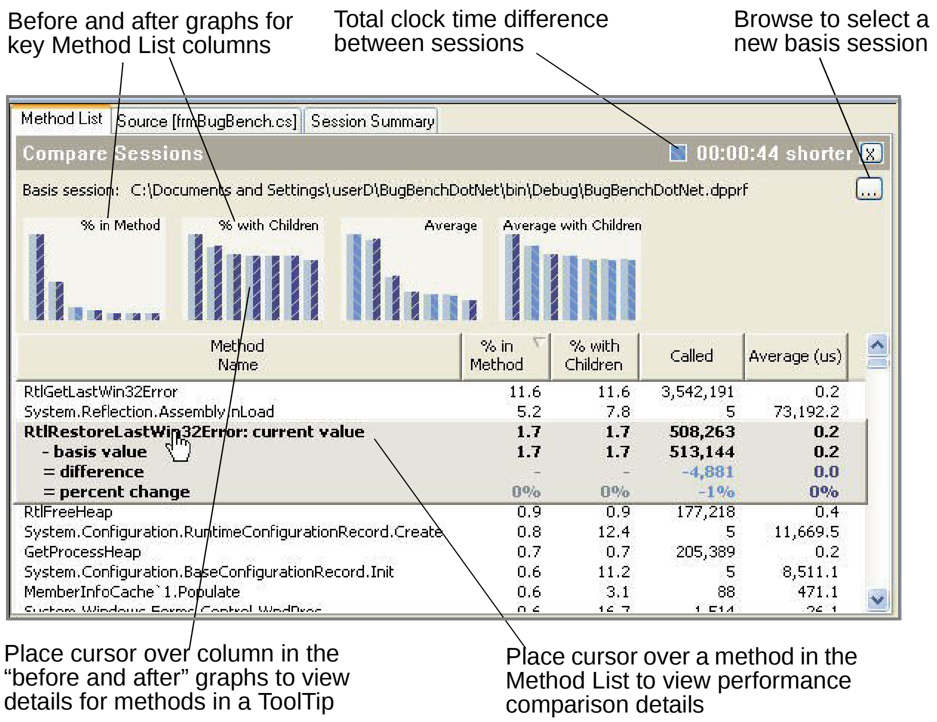
- If the base node has several parents, and one or more values on the links to the base node is high, the base node may be a candidate for optimization.
- If the values on the links to the parents are very small, optimizing the base node branch has little impact on parent method performance.
- To determine if the base node is the best choice to optimize, view a new Call graph with the parent selected as the base node. This shows the importance of the original base node to the parent node’s performance, relative to other children of that parent method.

When analyzing either the parent or child side of the Call graph, you can right-click a node and use the context menu to view the source code for the method to see if you can determine why it is using so much time.

Comparing Sessions

To fine tune a program's performance, you first need to locate where execution spends the most time so you can make adjustments to the costliest code fragments. Then you want to compare how those adjustments affect performance.

DevPartner gives you the ability to compare the results of one performance session with those of another so you can see the impact of optimizations you make on individual methods and on application performance as a whole.


Before and after graphs for key Method List columns
Total clock time difference between sessions
Browse to select a new basis session
Method List
Source [frmBugBench.cs]
Session Summary
Compare Sessions
00:00:44 shorter
% in Method
% with Children
Average
Average with Children
Place cursor over column in the “before and after” graphs to view details for methods in a ToolTip
Place cursor over a method in the Method List to view performance comparison details

Basis session: C:\Documents and Settings\userD\BugBenchDotNet\bin\Debug\BugBenchDotNet.dpprf

Method Name	% in Method	% with Children	Called	Average (us)
RtlGetLastWin32Error	11.6	11.6	3,542,191	0.2
System.Reflection.Assembly.nLoad	5.2	7.8	5	73,192.2
RtlRestoreLastWin32Error: current value	**1.7**	**1.7**	**508,263**	**0.2**
- basis value	**1.7**	**1.7**	**513,144**	**0.2**
= difference	-	-	**-4,881**	**0.0**
= percent change	**0%**	**0%**	**-1%**	**0%**
RtlFreeHeap	0.9	0.9	177,218	0.4
System.Configuration.RuntimeConfigurationRecord.Create	0.8	12.4	5	11,669.5
GetProcessHeap	0.7	0.7	205,389	0.2
System.Configuration.BaseConfigurationRecord.Init	0.6	11.2	5	8,511.1
MemberInfoCache`1.Populate	0.6	3.1	88	471.1

Figure 6-10. Comparing performance sessions

You invoke session comparison by toggling the **Compare** command with a session window open. The **Compare** command is available:

- As a tool bar button
- As a menu item on a performance session window context menu

When you first invoke the **Compare** command, you are prompted to choose a session file to be the basis session. DevPartner defaults to tracking the currently active session as the current session. Once you choose a basis session, the session window transforms to include a frame that you can place over any method in the method list. The frame displays a comparison between the basis and current session versions of that method.

When choosing sessions for comparison, try to ensure that the sessions compared were run in as near an identical fashion as possible. For example, do not compare a session started in the debugger with one run outside the debugger. To create more exact comparisons, consider using the session control file or the Session Control API to control data collection.

The upper right of the comparison window displays the overall time difference between the current session and the basis session. The graphic to left of the time display indicates whether the current session took more or less time than the basis session. This is useful for a quick comparison of sessions in which the application was exercised identically.

The comparison shows the current value, basis value, difference, and percent difference between the two versions of the chosen method. You can use DevPartner performance filters to alter the views of your session data.

At the top of the session window, four bar charts show a graphical view of the same information for the top methods in the current session.

You can copy the information in the session comparison data box by invoking the **Copy Comparison** command from the context menu within the **Method List**. This command copies the data onto the clipboard.

When you finish comparing, press **Esc**, or click the **Compare** icon.

Interpreting Session Comparison Results

A session comparison shows the **current value**, **basis value**, **difference**, and **% difference** between a method in the current session and the same method in the basis session. DevPartner uses color to help you see at a glance whether the value in the current session is larger or smaller than that in the basis session. When the values for difference and percent difference are dark blue, the values for the current session were better (faster) than those of the basis session. Light blue means that the performance values were slower in the current session.

Once you have determined what results your code changes accomplished between sessions for any given method, search other methods in the session to uncover any side effects of your initial code changes. Even though an individual method's performance improved, the larger program's performance may have degraded. In performance tuning, no tool can substitute for thorough knowledge of the structure of your code.

When examining session comparison results, be aware of the following:

- A percentage is a ratio of two numbers. Percentages are additive only when computed relative to the same total value.
- If one percentage value decreases, all other percentage values must increase. In a complex program this may be difficult to notice, since the percentage increase must be averaged across all the other methods in the program.
- To interpret a subprogram’s timing, you must understand that subprogram’s role in the enclosing program.
- Performance measurements have no meaning outside the context of the program that produced them. It is not possible to generalize about the effects of program changes without understanding the program’s operation.

Once you are satisfied with the changes to the costliest method in your program, you can turn your attention to other expensive methods.

Exporting Performance Data

You can export performance data in XML format or in CSV format. Exporting data in XML or CSV format facilitates use of your own or third-party software to analyze the data, integrate the data with data produced by other tools, and archive the data in a data warehouse.

- You can export DevPartner performance session files (with the **`.dpprf`** extension) to XML format. When a saved performance session file is open, the **Export DevPartner Data** command is available on the **File** menu. Refer to “Exporting Analysis Data to XML” on page 311 for information about exporting in XML format.

  You can also export data from the command line, as described in “Exporting Analysis Data to XML from the Command Line” on page 312.

- You can export Method List data to a comma-delimited CSV text file. Click the **Method Lis**t tab to make it active, display the columns you want to export, right-click in the Method List and choose **Export Method List** from the context menu. You can open the comma-delimited text file in Microsoft Excel or another spreadsheet application.

Controlling Data Collection

DevPartner gives you three ways to control when performance data is collected during the use of your application:

- You can use the session control toolbar to interactively control data collection as your program runs.
- You can use a session control file to assign session control actions to specific methods in your application modules.
- You can use the Session Control API to control data collection in your program.

Using the session control toolbar or Session Control API allows you to control data collection anywhere within a method. Using a session control file allows you to control collection only at the entrance to or exit from a method.

Using a session control file and using the Session Control API is described in “Analysis Session Controls” on page 303.

Analyzing from the Command Line

To automate data collection or run analysis sessions from the command line, use **`DPAnalysis.exe`**, the DevPartner command-line executable. For information on using **`DPAnalysis.exe`**, refer to “Starting Analysis from the Command Line” on page 285.

Using the Performance Analysis Viewer

DevPartner Studio provides a lightweight Performance Analysis Viewer for analyzing performance session files independently of Visual Studio. To launch the viewer, do any of the following:

- On the Start menu, select **Programs > Micro Focus > DevPartner Studio > Performance Analysis Viewer**.
- Double-click a **`.dpprf`** session file in Windows Explorer.
- Run a performance analysis session using **`DPAnalysis.exe`** on the command line. DevPartner displays the session data in the Performance Analysis Viewer.

What You Can Do in the Performance Analysis Viewer

With a session file open, you can view, sort, save, or print performance session data. In addition, you can:

- View the source code for a method
- Sort the data on the Method List tab
- View the Call graph for a method
- Compare session data
- Export the contents of the file as XML
- Export the contents of the **Method List** in CSV format

What you Cannot Do in the Performance Analysis Viewer

- Instrument an unmanaged application for performance
- Start a performance session
- Add files to a Visual Studio solution

Session files generated from the command line are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

Performance Analysis Tips for .NET Applications

The following are strategies you can use to make the performance analysis process more productive.

- **Analyze source code**

  Use the **Top 20 Source Methods** filter to isolate application hotspots.

  To avoid collecting data for all system (non-source) files, check **Exclude system images** on the **DevPartner Exclusions - Performance** options page. Once you optimize your source code, turn off this option so you can examine how your application uses system code, especially the .NET Framework.

  Use the Call graph to examine the most expensive methods to understand the costs associated with child methods called.

  Compare the effect of different algorithms or logic changes by running multiple performance sessions.

- **Understand Framework costs**

  Use **% with Children** on the **Method List** or **Source** tab to see how much time you are spending in the .NET Framework.

Drill into the .NET Framework by examining child methods in the Call graph to understand which calls are expensive and why.

Rework the application to do less work or to call the .NET Framework less often.

- **Understand start-up costs**

  Use the **Clear** session control before collecting performance data. The .NET Framework performs many one-time initializations. To prevent these from skewing performance results, warm up the application by exercising all the features you want to profile, then **Clear** the data. Next, run a test that exercises the same features to get a more accurate performance picture.

- **Understand what you want to measure**

  Consider how your application behaves before you begin collecting performance data. For example, if you are profiling a Web service or ASP.NET application, think about how Web caching affects results. If your test run inputs the same data repeatedly, your application fetches pages from the cache, skewing the performance data. In such a case, you could take pains to insure variable input data, or simpler, edit the **`machine.config`** file to turn off caching while you test. Comment out the line that reads:

  ```
  <add name=”OutputCache” type=System.Web.Caching.Output-
  CacheModule”/>
  ```

- **Measure performance of mixed-mode applications**

  You may choose to write parts of a .NET application in unmanaged C/C++. DevPartner allows you to collect performance data for both managed and unmanaged portions of an application in a single run, provided the unmanaged code is in a separate file and you instrument the code before collecting data. Thus, you can compare the effectiveness of unmanaged and managed code in the context of the total application by comparing performance sessions.

- **Collect complete data for distributed applications**

Tip: Use the process list on the **Session Control** toolbar to take performance snapshots of each process in a distributed multi-process application.

  When you analyze performance for a Web application, a multi-tier client/server application, or an application that uses Web services, include all remote application components in the analysis. Use a DevPartner installation to configure performance data collection on remote systems. If your application uses unmanaged C/C++ components, instrument the components for performance analysis before collecting data. Recommendations regarding start-up costs, .NET Framework costs, and awareness of application behavior apply equally to collecting data for server-side components.

- **Understand the limitations of micro-profiling**

  Once you identify a bottleneck in your application, you may find it convenient to create a smaller sample of code that duplicates the problem area in the main application. You improve performance in that sample by iterative performance comparisons and then move the code back in to the main application. Is your application going to be faster? Maybe. But you cannot know until you rerun your original performance tests.

- **Simulate actual running conditions**

  Application memory footprint, multi-threading, thread priorities, process security, network latency, server load, and other contingencies can affect the way your code runs in ways that performance testing of a single component may not reveal. You have not measured application performance until you have simulated as closely as possible the conditions under which your application is going to be used.

Submitting Data to Visual Studio Team System

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected item. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected item. For a coverage analysis session file, access work item submission in for a method selected in the Method List tab. When submitting a work item, DevPartner populates the **Work Item** form with data from the visible columns in the **Methods List** tab. To change the method data you submit in the **Work Item**, change the columns displayed in the **Method List**. DevPartner Studio supports Microsoft Visual Studio Team System if the Team Explorer client is installed and a Team Foundation Server connection is available. Refer to “Visual Studio Team System Support” on page 20 for general information about Team System support.

In a performance analysis session file, you can submit data for a method selected in the **Method List** tab in a DevPartner performance analysis session file as a **Work Item** to Visual Studio Team System.

When you submit a work item, DevPartner populates the **Work Item** form with data from the visible columns in the **Methods List** tab. To change the method data you submit in the **Work Item**, change the columns displayed in the Method List.

Chapter 7

In-Depth Performance Analysis

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with Performance Expert. The second section provides reference information for an in-depth understanding of the DevPartner Studio Performance Expert component.

Refer to the DevPartner Studio online help for additional task-oriented information about Performance Expert.

What is Performance Expert?

DevPartner Studio contains many features designed to assist application development, including a performance analyzer that helps you locate bottlenecks in your code. Performance Expert takes performance analysis a step further for managed 32-bit and 64-bit Visual Studio applications by providing deeper analysis of the following hard-to-solve problems:

- CPU/thread usage
- File/disk I/O
- Network I/O
- Synchronization wait time

Note: Performance Expert analyzes managed code only, and is therefore not supported in the DevPartner for Visual C++ BoundsChecker Suite.

Performance Expert analyzes your application at run-time and locates the problem methods in your code. It then allows you to view details about individual lines in the method, or to examine parent-child calling relationships to help you determine the best way to fix the problem. When you decide on an approach, Performance Expert enables you to jump directly to the relevant lines in your source code, so you can quickly fix problems.

Because Performance Expert is integrated into Visual Studio, you can use it to test applications as you develop them. You can also run Performance Expert sessions from the command line, or as part of an automated test scenario, by using the DevPartner command-line executable **`DPAnalysis.exe`** with traditional command-line switches or an XML configuration file. For information, see "Starting Analysis from the Command Line" on page 285.

Performance Expert is designed for use by software designers, software developers, and quality assurance (QA) engineers. It can also be used by development management staff to identify problems in an ongoing project.

Performance Expert and Performance Analysis

Think of Performance Analysis as a complement to traditional performance profiling. First, run your application with performance analysis to get a baseline view of performance. Next, run an identical session with Performance Expert to better understand the nature of difficult problems, especially problems that involve disk or network I/O, or synchronization issues. When you have fixed the problem, run the application again with performance analysis and use the performance analysis **Session Comparison** feature to verify the improvement. For information on comparing performance analysis sessions, see “Comparing Sessions” on page 200. For more information on using Performance Expert in conjunction with performance analysis, see “Using Performance Expert with Performance Analysis” on page 240.

Using Performance Expert Out of the Box

The following Ready, Set, Go procedure introduces you to using Performance Expert.

To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information about the subject described in a shaded box, read the additional text following the box.

Analyzing an application with Performance Expert does not require elevated system privileges. The system privileges you use to create and debug your application are sufficient for Performance Expert to analyze the application.

Ready: Consider What You Want to Analyze

When determining the type of application being analyzed. think about what steps, if any, you need to take before beginning a Performance Expert session.

Performance Expert collects data only from managed applications. To collect Performance Expert data for your application, the solution must contain at least one managed code project (for example, C#, Visual Basic, or managed C++). It must also include a startup project. If the solution includes multiple startup projects, DevPartner prompts you to choose a startup project for the session.

The following procedure assumes:

- You are testing a single process, managed application.
- You can build and run your application.
- Your solution contains at least one managed code project.
- Your solution includes a startup project.

See “DevPartner Studio Supported Project Types” on page 267 for a comprehensive list of supported project types for DevPartner memory analysis.

Performance Expert monitors a single process when run from Visual Studio, or when run with **`DPAnalysis.exe`** using traditional command-line switches. Although you can collect Performance Expert data from more than one process or service in a session by using **`DPAnalysis.exe`** with an XML configuration file, it is usually best to target a single process in a Performance Expert session. If your application runs in more than one process, rerun the application, targeting the second process. For more information about using **`DPAnalysis.exe`**, see “Automating Data Collection” on page 231.

You can use Performance Expert to improve performance of any managed Visual Studio application, including:

- ASP.NET Web applications
- ASP.NET Web services applications
- .NET Remoting server applications
- Windows Forms client applications
- Serviced components, e.g. COM+

Decide what data you are interested in collecting before beginning your Performance Expert session. Think about how your application performs. Does it slow down when you use certain features? If so, exercise that feature when you run your application with Performance Expert. Has traditional performance analysis indicated that excessive time is being spent in methods that read or write data, or access network resources? Performance Expert can provide additional information about disk and network I/O, so target that feature in a Performance Expert session.

If your application includes a local client process and a remote server process, are you interested in data from both processes? If so, you must first install DevPartner Studio and a DevPartner Remote Server license on the remote computer to collect the server data. Before collecting server-side data, be aware that some IIS setup might be required.

Set: Properties and Options

Once you have decided what code you want included in the Performance Expert session, you can set several properties and options to focus your data collection.

For this procedure, you can use the default DevPartner properties and options. No additional set-up is required.

Using Solution Properties or Project Properties, you can choose a startup project for the session or exclude certain projects from the session, if your solution contains multiple projects. Using DevPartner options, you can change display options or create a Session Control file to manage data collection. Setting up your analysis session is described in “Setting Properties and Options” on page 221.

Go: Collect Performance Expert Data

After considering what you want to analyze and setting the appropriate properties and options, you are ready to collect Performance Expert data.

DevPartner supports the Visual Studio launch model. When you click the Performance Expert icon or choose **Start without debugging with Performance Expert** on the **DevPartner** menu, DevPartner rebuilds the solution, launches the startup project for your application, and begins to collect Performance Expert data.

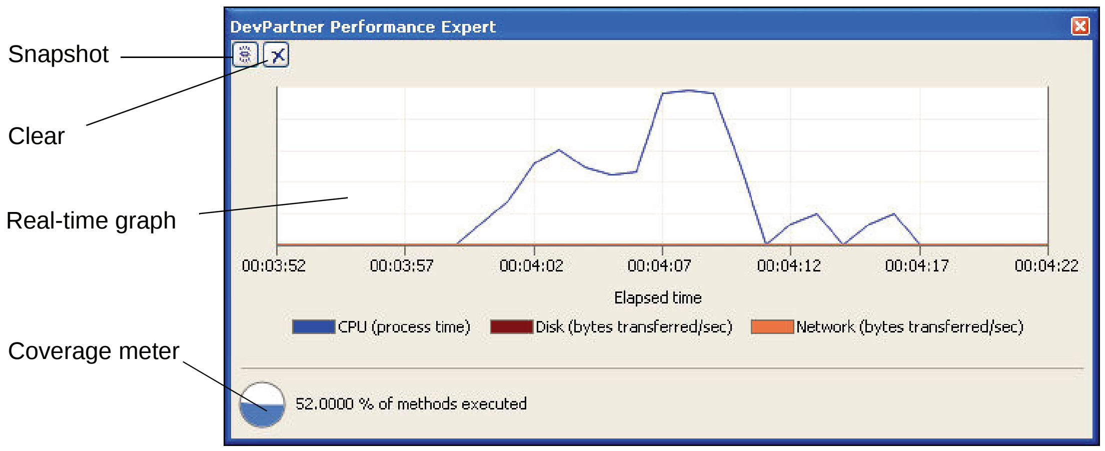


Figure 7-1. Controlling Data Collection with the Performance Expert window

1 In the **Performance Expert** window, click the **Clear** session control at the upper left to clear startup and initialization data and focus data collection on the problem feature.

2 Exercise the slow portion of your application.

3 Watch the **Performance Expert** window as you exercise your application. The graph displays a line for CPU process time, and if present, lines for disk and network activity. A spike in any of these lines may indicate a potential trouble spot.

4 If you see something interesting, click the **Snapshot** session control. DevPartner generates a Performance Expert session file and displays it in Visual Studio.

Using the Performance Expert Window

Using the Real-Time Graph

The Performance Expert real-time graph presents the last 30 seconds of activity as you run your application. The graph always draws a line reflecting CPU use. If your application does disk or network reads or writes, the graph includes separate lines for disk I/O and network I/O.

Use the real-time graph to monitor application activity. If you see something interesting, for example, a spike in activity in the graph, you can use the **Snapshot** button to take a snapshot of activity to that point. Conversely, if nothing of interest has happened, use the Clear button to clear data collected to that point.

Using Clear and Snapshot

The **Clear** and **Snapshot** buttons are located above the real-time graph, at the upper left of the Performance Expert window. Use these session controls to perform the following actions:

- **Clear -** Clears data collected to that point, or since the last clear action. Use Clear to focus data collection and minimize the size of the session results file.

- **Snapshot -** Creates a session results file that contains data collected up to that point, or since the last clear action. Data collection continues. You can take multiple snapshots as your application runs.

Using the Coverage Meter

The **Coverage Meter** is located below the real-time graph, at the lower left of the session control window. The coverage meter displays the percentage of your application methods that have been executed up to that point in the session. Use the coverage meter to ensure that you have tested all of your code under the Performance Expert. You can also use the coverage meter in conjunction with the session control actions to help focus data collection on certain parts of your application.

Note: Generally, run Performance Expert sessions without debugging. Results from non-debug sessions are easier to interpret and do not include the processing overhead caused by the debugger. If you run your application in the debugger, some timing values might be larger than expected, especially if breakpoints were hit during the session. Expect tracing and other debug-only functionality to figure highly in such session files.

5 When you finish collecting data, close your application.

When you close your application, DevPartner generates a final Performance Expert session file. If you want to capture all the session data in a single session file, it is not necessary to use the **Snapshot** session control. Simply close the application.

Analyze the Data

When you take a data snapshot, or when you finish collecting data and quit your application, DevPartner produces a Performance Expert session file. The initial view of the data DevPartner has collected for your application appears in the form of a results summary.

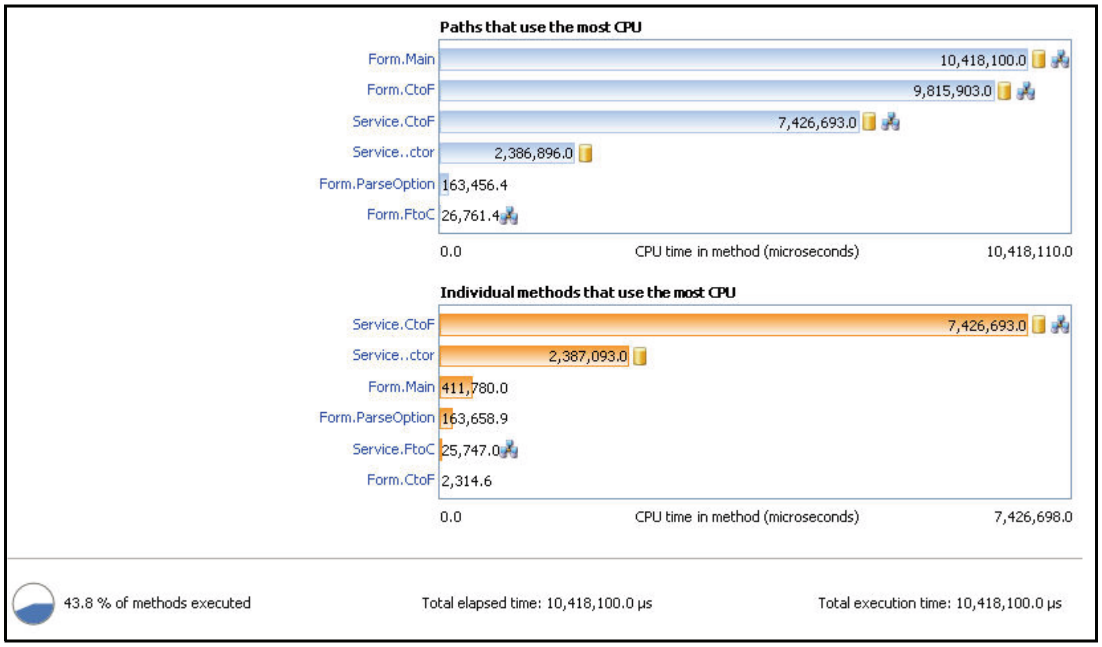


Figure 7-2. The Performance Expert Results Summary

The results summary contains two bar graphs, reflecting two ways to analyze the data to solve application problems:

Tip: An entry point method is a source code method that was not called by another source code method, i.e., an entry point into source code execution.

- **Paths that use the most CPU** displays entry point methods for the top paths, or chains of method calls, that consumed the most CPU cycles in the session. Path analysis enables you to quickly identify the most expensive paths of method execution. You can:
  - Fix the child methods responsible for poor performance
  - Modify other methods in the calling sequence so they call the expensive child methods less often
- **Individual methods that use the most CPU** displays the top methods in terms of CPU cycles consumed. Method analysis enables you to quickly identify individual problem methods so you can fix them.

Notice the icons at the ends of the bars in Figure 7-2 on page 211. These icons indicate that a method caused disk or network activity.

Deciding Where to Start

To begin evaluating the session data, compare the two bar graphs on the results summary.

- Is the top path in the **Paths that use the most CPU** chart significantly longer than the other paths in the chart?
- Does the top method in the **Individual methods that use the most CPU** chart stand out from the other methods in the chart?
- Does the time value for a method seem excessive for what the method does?
- Does the same method appear as expensive on both charts?

If the answer is yes to any of these questions, investigate that method.

Before you analyze the data, learn to navigate the data views that you can access from the results summary.

1. In the results summary, click on the top path in the **Paths that use the most CPU** chart. The **Path analysis** window opens.

   Notice that the **Path analysis** window includes **Call Graph** and **Call Tree** tabs, and below, **Source** and **Call Stacks** tabs.

2. Click **Back to Summary** to return to the results summary.
3. In the results summary, click the top method in the **Individual methods that use the most CPU** chart. The Methods window opens.

   Notice that the Methods window includes a list of the methods executed in the session, and below, **Source** and **Call Stacks** tabs.

4. Click **Back to Summary** to return to the results summary.

Analyzing Paths that Use the Most CPU

If you drill down from the **Paths that used the most CPU** graph, you can view Call graph and Call tree presentations of the session data. The Call graph shows the child methods called by the entry point method, with the relative contributions of each to the time spent in the path. The Call tree presents a tree view of the same data but adds additional data about each method in the form of user-configurable data columns.

If you choose to examine a method in the **Individual methods that use the most CPU** graph, DevPartner presents a **Methods** table with user-configurable data columns to assist your troubleshooting. To switch between the **Path analysis** and **Methods** table views, click **Back to Summary** in any details view.

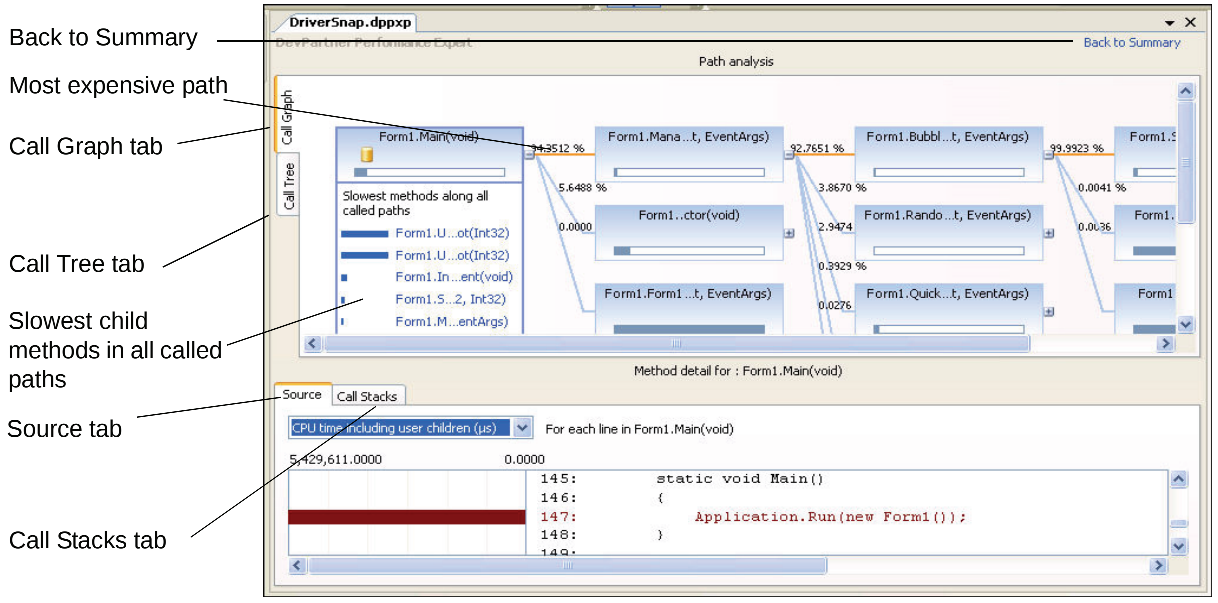
The calculation of the Performance Expert session data differs between the **Paths that use the most CPU** and the **Individual methods that use the most CPU** views. In the **Individual methods that use the most CPU** view, DevPartner excludes measurements for source code child methods in computing data for CPU time, disk or network I/O, and synchronization lock wait time. Excluding source code child methods focuses attention on methods that, in themselves, consume large amounts of CPU time. In contrast, DevPartner includes the impact of source code child methods to their parent methods in the **Paths that use the most CPU** view in order to highlight the most expensive paths of execution.

All computations in both views include time or throughput attributable to system or .NET Framework methods called by your source code methods. Managed applications typically spend significant time executing .NET Framework code. Performance Expert charges the system data to the lines in your source code that made the calls in order to focus attention on how your code interacts with the .NET Framework, that is, on the parts of the application that you can modify.

In this procedure, **Path analysis is used first** to analyze the relative contribution of child methods called in the most expensive paths of execution.

5 In the results summary, click on a method in the **Paths that used the most CPU** chart to drill down to the **Path analysis** view. If the Call graph is not visible, click the **Call Graph** tab, at the left.

DevPartner highlights the **critical** or most expensive path of execution. Start your troubleshooting here.



```
145:        static void Main()
146:        {
147:            Application.Run(new Form1());
148:        }
```

Figure 7-3. Identifying Expensive Execution Paths in the Path Analysis window

In the Call graph:

6 To investigate a path, click the plus sign on a node to expand the path to the right.

7 Click on any method to see the list of the slowest child methods it called, regardless of path. This list exposes slow methods that may not be part of the critical path.

8 To determine the relative contributions of different paths spawned by the same method, compare the percentage values on the lines that connect the selected method to each of its child paths. Investigate the most expensive (highest percentage) paths first.

9 Hover over the horizontal bar at the bottom of each node with the mouse pointer to see the percentage of time spent in the method versus the time spent executing child methods. For an example, see Figure 7-4 on page 215.

If most of the time is attributable to child methods, continue to investigate the path. If most of the time was local to the method, focus your efforts on that method.

The Call graph helps you quickly locate expensive methods in the calling sequence so you can focus your tuning efforts. In addition to showing the impact of child methods, the nodes in the Call graph provide insight into what your methods do. For more information on using the Call graph, see “The Call Graph” on page 227.

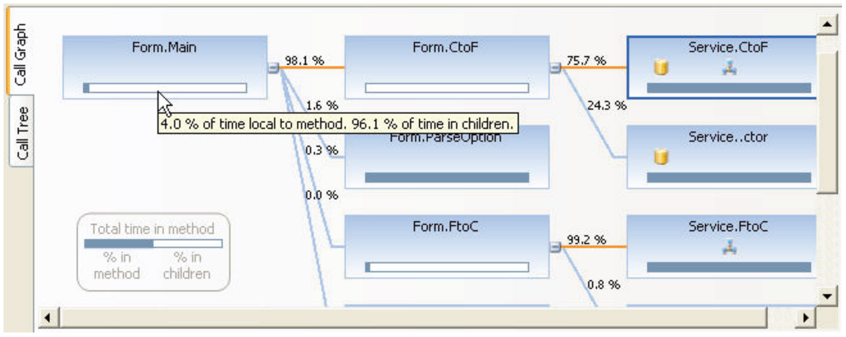


Figure 7-4. Assessing the Impact of child methods

10 Does a node you plan to investigate contain one or more of the following icons?

- Indicates disk activity
- Indicates network activity
- Indicates synchronization wait time

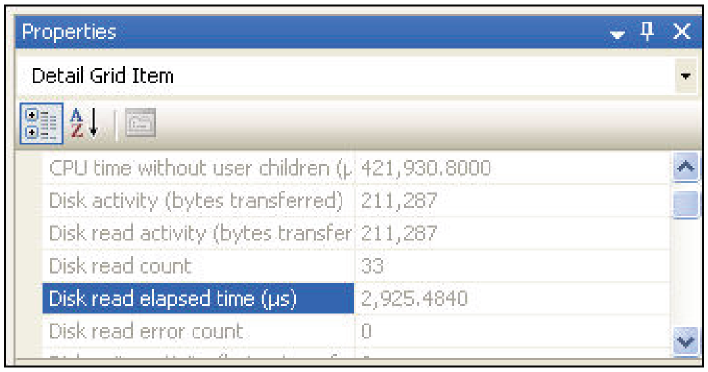
11 If so, hover over the icon with the mouse pointer to view the magnitude of the activity. If you think that the magnitude of the activity merits further investigation, switch to the **Call Tree** tab for more diagnostic help.

12 To view the **Call Tree**, click the **Call Tree** tab on the left side of the session file window.

Tip: The term "user children" refers to your own application source code methods, as opposed to system code or .NET Framework methods also called by your application code.

The Call tree provides information similar to the Call graph**,** but in the form of a tree view. The most expensive paths are indicated by position in the sort order of the table. The default sort column is **CPU time including user children**.

As you saw above, the Call graph provides information about the relative contribution of child methods to their parent methods. In contrast, the Call tree offers more detailed data about what the methods in your application actually do. This data is presented in the form of sortable, user-configurable data columns. You can add these data columns to the Call tree view by right-clicking in any column header and selecting **Choose Columns...** from the context menu. Before adding data columns, you can preview the data they contain in the Properties window in Visual Studio. To display the Properties window, choose **View** > **Properties Window**.


Properties

Detail Grid Item

CPU time without user children (µ	421,930.8000
Disk activity (bytes transferred)	211,287
Disk read activity (bytes transfer	211,287
Disk read count	33
Disk read elapsed time (µs)	2,925.4840
Disk read error count	0

Figure 7-5. Viewing Method data in the Properties window

In the Call tree:

13 To determine the relative contributions of different paths spawned by the same method, compare the values in the **CPU time including user children** column for each of the child paths. When sorted by this column (the default sort), the most expensive paths appear at the top of the tree view.

14 Use the Call tree in conjunction with the Call graph. For example, if an expensive node in the Call graph includes the network I/O icon, switch to the **Call Tree** and add the network-related data columns to the view.

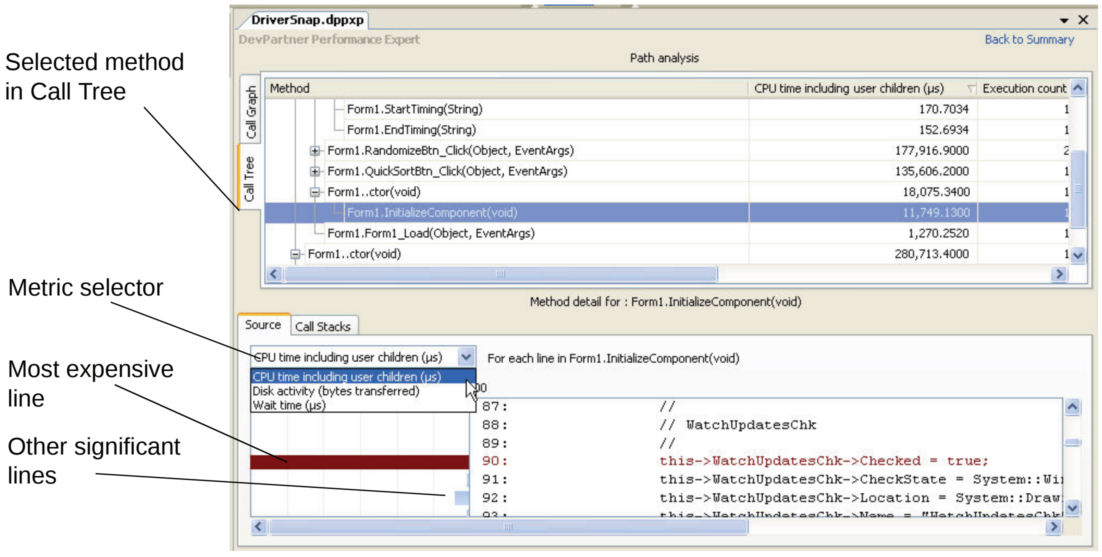
To add data columns to the Call tree view, right-click any column header and select **Choose Columns...** from the context menu.

These data columns show you the number of network reads or writes, how much time was spent reading or writing data across the network, the amount of data read or written, and the number of read or write errors.

If the node in the Call graph included the disk I/O or wait time icon, add those data columns to the Call tree view. In this way, you can quickly pinpoint the reason the problem node is so expensive.

Both the Call graph and the Call tree windows include a **Source** tab and **Call Stacks** tab in the lower part of the window.

The **Source** tab enables you to view source code for your application's methods, with metrics that indicate the expense of the lines that were executed during the session. Use it to view expensive lines of code in context, and to quickly locate lines that would be good candidates for improvement. The **Source** tab includes a metric selector, as shown in Figure 7-6 on page 217. The default metric in the **Path analysis** view is **CPU time including user children**. Additional metrics, including disk I/O, network I/O, and wait time, may be available depending on what the method does. Selecting a new metric in the selector updates the source pane so you can locate the most expensive line for that metric in your source code.


Selected method in Call Tree
Metric selector
Most expensive line
Other significant lines
DriverSnap.dppxp
DevPartner Performance Expert
Back to Summary
Path analysis
Call Graph
Call Tree
Method detail for : Form1.InitializeComponent(void)
Source
Call Stacks
CPU time including user children (µs)
CPU time including user children (µs)
Disk activity (bytes transferred)
Wait time (µs)
For each line in Form1.InitializeComponent(void)

Method	CPU time including user children (µs)	Execution count
Form1.StartTiming(String)	170.7034	1
Form1.EndTiming(String)	152.6934	1
Form1.RandomizeBtn_Click(Object, EventArgs)	177,916.9000	2
Form1.QuickSortBtn_Click(Object, EventArgs)	135,606.2000	1
Form1..ctor(void)	18,075.3400	1
Form1.InitializeComponent(void)	11,749.1300	1
Form1.Form1_Load(Object, EventArgs)	1,270.2520	1
Form1..ctor(void)	280,713.4000	1

```
87:            //
88:            // WatchUpdatesChk
89:            //
90:            this->WatchUpdatesChk->Checked = true;
91:            this->WatchUpdatesChk->CheckState = System::Wi
92:            this->WatchUpdatesChk->Location = System::Draw
```

Figure 7-6. Locating the most expensive lines in the Source tab

Use the **Source** tab in conjunction with the Call graph and Call tree.

15 Select a method of interest in the **Call Tree** tab. (If you have returned to the **Call Graph** tab, you can select a method node.) Select the **Source** tab. Notice that the most expensive line (measured by **CPU time including user children**) is highlighted in dark red. Scroll through the **Source** pane and notice that other expensive lines are highlighted in blue.

16 Did the method you selected have disk, network, or wait time activity? To quickly locate such methods in the Call tree, look for methods with high values in those data columns. (In the Call graph, look for the disk, network, or wait time icon in the method node.)

17 Expand the metric selector (see Figure 7-6 on page 217) at the upper left of the **Source** tab. If the selected method include disk I/O, network I/O, or wait time, the metric appears in the list. Select a new metric and scroll the source display to locate the most expensive line for that metric. An expensive method may present multiple opportunities for improvement.

18 Locate the line you want to fix in the **Source** tab. Double-click the line to open the source file in Visual Studio for editing.

The **Call Stacks** tab enables you to view different instances or usages of the expensive methods of your application. Each call stack is unique. In some cases you may see call stacks that contain the same sequence of method calls. However, some of the calls were made from different lines in at least one method.

Notice that as you select different methods in the Call graph or Call tree, the **Source** tab scrolls to the most expensive line in each method. Similarly, the **Call Stack** tab updates when you select a different method

For example, in Figure 7-6 on page 217, a child method is selected in the Call tree. If you plan to address the performance issue by fixing the child method itself, look at the **Source** tab. On the **Source** tab, you would see that the most expensive line in the method is highlighted. If the method did disk I/O or network I/O, or had significant wait time, use the metric selector to locate the most expensive lines for the selected metric. Once you decide what you want to fix, double-click the source line to edit it in Visual Studio.

On the other hand, if you plan to address the performance issue by changing the way your application calls the child method, switch to the **Call Stacks** tab.

Use the **Call Stacks** tab in conjunction with the **Call Graph** or **Call Tree**. The **Call Stacks** tab shows you all of the paths that called the selected method, so you can evaluate changes to the method in the context of all the ways the method is used in your application. Use the **Call Stacks** tab to quickly locate the most expensive instances (usages) of any method.

19 Select a method of interest in the **Call Tree** tab. (If you have returned to the **Call Graph** tab, you can select a method there.) Select the **Call Stacks** tab. Notice that DevPartner highlights the line that called the selected method.

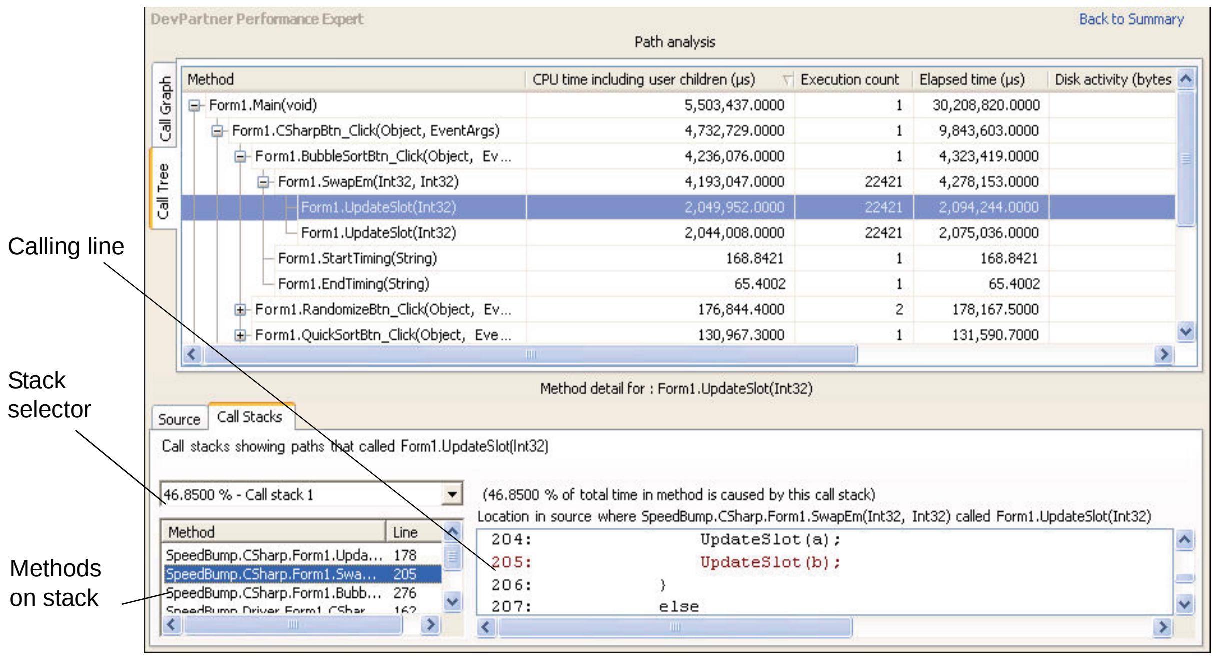
20 Expand the stack selector at the upper left of the **Call Stacks** tab. Use the stack selector to locate the most expensive usages of the method.

21 Locate the line you want to fix in the **Call Stacks** tab. Double-click the line to open the source file in Visual Studio for editing.

22 If you cannot directly fix an expensive method, modify your code to call the method less often, or not at all.

On the **Call Stacks** tab, you can examine all the calling sequences or paths that called the method you selected in the Call tree. In Figure 7-7 on page 219, note that the stack selector shows the percentage of time attributable to each call stack, so you can quickly locate the most expensive execution path. When you select a call stack, DevPartner shows all of the methods that make up the stack, with the number of the line in each method that called the next method on the stack. Selecting any method in the call stack updates the source pane to highlight the line that called the next child method. Double-click the calling source line to edit it in Visual Studio.

Even if you plan to fix a slow child method rather than change the way it is called, examine the **Call Stacks** tab for the method. It is a good idea to understand all the ways your application uses a method before you change it. Performance Expert makes it easy to do so.



Path analysis

Method	CPU time including user children (µs)	Execution count	Elapsed time (µs)	Disk activity (bytes
Form1.Main(void)	5,503,437.0000	1	30,208,820.0000	
Form1.CSharpBtn_Click(Object, EventArgs)	4,732,729.0000	1	9,843,603.0000	
Form1.BubbleSortBtn_Click(Object, Ev...	4,236,076.0000	1	4,323,419.0000	
Form1.SwapEm(Int32, Int32)	4,193,047.0000	22421	4,278,153.0000	
Form1.UpdateSlot(Int32)	2,049,952.0000	22421	2,094,244.0000	
Form1.UpdateSlot(Int32)	2,044,008.0000	22421	2,075,036.0000	
Form1.StartTiming(String)	168.8421	1	168.8421	
Form1.EndTiming(String)	65.4002	1	65.4002	
Form1.RandomizeBtn_Click(Object, Ev...	176,844.4000	2	178,167.5000	
Form1.QuickSortBtn_Click(Object, Eve...	130,967.3000	1	131,590.7000	

Method detail for : Form1.UpdateSlot(Int32)

Call stacks showing paths that called Form1.UpdateSlot(Int32)

46.8500 % - Call stack 1

(46.8500 % of total time in method is caused by this call stack)

Location in source where SpeedBump.CSharp.Form1.SwapEm(Int32, Int32) called Form1.UpdateSlot(Int32)

Method	Line
SpeedBump.CSharp.Form1.Upda...	178
SpeedBump.CSharp.Form1.Swa...	205
SpeedBump.CSharp.Form1.Bubb...	276
SpeedBump.Driver.Form1.CShar...	162

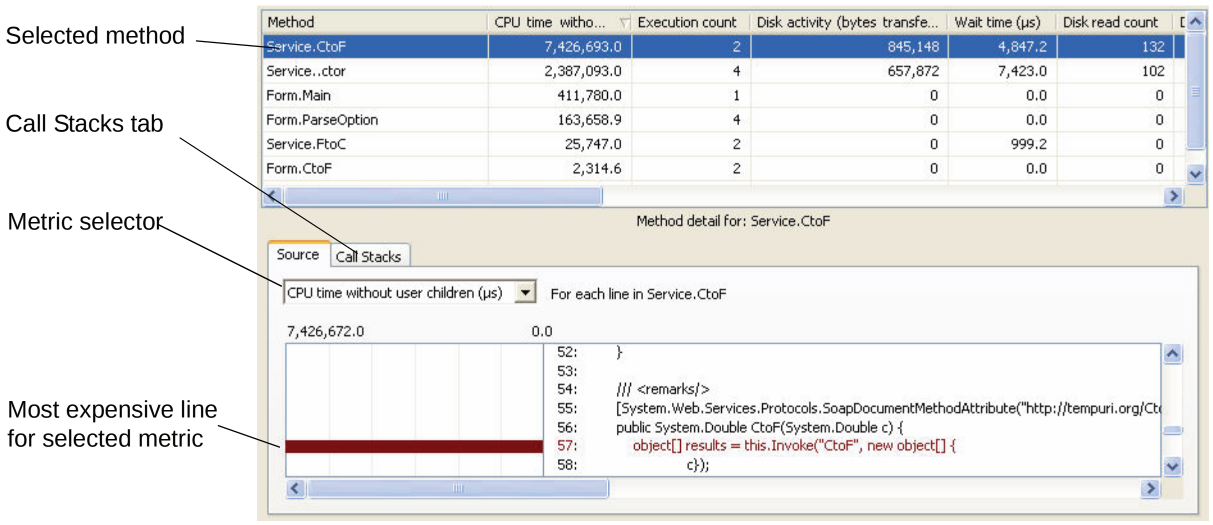
```
204:            UpdateSlot(a);
205:            UpdateSlot(b);
206:        }
207:        else
```

Figure 7-7. Identifying the most expensive calling paths that used the method

Analyzing Individual Methods that Use the Most CPU

As well as drilling into the session data from the **Paths that use the most CPU** bar chart on the results summary, Performance Expert data is analyzed by using the **Individual methods that use the most CPU** bar chart. For example, if the top method in this chart is consuming more time than it should, you can click the method in the chart to examine it immediately.

Clicking the method opens a Methods table that lists the methods that executed when you ran your application.



Method	CPU time witho...	Execution count	Disk activity (bytes transfe...	Wait time (µs)	Disk read count
Service.CtoF	7,426,693.0	2	845,148	4,847.2	132
Service..ctor	2,387,093.0	4	657,872	7,423.0	102
Form.Main	411,780.0	1	0	0.0	0
Form.ParseOption	163,658.9	4	0	0.0	0
Service.FtoC	25,747.0	2	0	999.2	0
Form.CtoF	2,314.6	2	0	0.0	0

Method detail for: Service.CtoF

CPU time without user children (µs)   For each line in Service.CtoF

7,426,672.0   0.0

```
52:     }
53:
54:     /// <remarks/>
55:     [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Ct
56:     public System.Double CtoF(System.Double c) {
57:        object[] results = this.Invoke("CtoF", new object[] {
58:                    c});
```

Figure 7-8. Analyzing the impact of individual methods

23 To access the **Individual methods that use the most CPU** views, click **Back to Summary**.

24 In the results summary, click on a method in the **Individual methods that use the most** CPU chart to drill down to the **Methods** table.

DevPartner highlights the most expensive individual methods in your application. By default, methods are sorted by CPU time spent in the method, without user children, but including system calls.

25 To customize the column selection in the **Methods** table, right-click any column header and select **Choose Columns...** from the context menu.

Use the data columns to determine the most expensive aspects of method performance.

By default, the Methods table is sorted by **CPU time without user children**. This metric focuses on the performance of the method itself. In contrast, the **Paths that use the most CPU** views include user, or source code, child methods in the calculation.

Use the **Source** tab in conjunction with the Methods table. When you select a method, the **Source** tab guides you directly to the most expensive line in the method and displays the relative cost of other lines. The most expensive line appears in dark red. Other lines that contribute to time spent in the method appear in light blue.

26 Use the metric selector on the **Source** tab to locate the most expensive lines for each available metric. A problem method may present multiple opportunities for improvement.

Use the **Call Stacks** tab in conjunction with the **Methods** table. The **Call Stacks** tab shows you all of the paths that called the selected method, so you can evaluate changes to the method in the context of all the ways the method is used in your application. Use the **Call Stacks** tab to quickly locate the most expensive instances (usages) of any method.

27 Locate the line you want to fix in the **Source** tab or in the **Call Stacks** tab. Double-click the slow line and open the source in Visual Studio for editing.

28 If you cannot directly fix an expensive method, modify your code to call the method less often, or not at all.

Although the percent of time calculation in the **Individual methods that use the most CPU** and Methods tables excludes time spent in source code child methods, it includes time spent in system child methods. You have probably noticed that managed applications spend a good deal of time in the .NET Framework. Including system children in the calculation focuses attention on methods in your source code that exhibit problems in the way they interact with system code, which can be especially critical in managed applications.

Saving Session Files

When you finish reviewing the Performance Expert data you can save the session file or discard it.

1 Select the unsaved session file in Visual Studio. Choose **File > Save <filename>.dppxp**.

2 If you close the session file window in Visual Studio before saving the session, DevPartner prompts you to save the open session file.

DevPartner saves session files as part of the active solution. They appear in the DevPartner Studio virtual folder in Solution Explorer. Performance Expert session files take the **`.dppxp`** extension.

By default, DevPartner physically saves the session files in your project's output folder. DevPartner automatically increments the file name based on the contents of the default folder (for example, **`MyApp.dppxp`**, **`MyApp1.dppxp`**, and so on). If you save session files to a location other than the default folder, you must manage the file naming.

For projects that do not have an output folder, such as a Visual Studio 2005 Web site project, DevPartner physically saves the files to the project folder.

Session files generated outside of Visual Studio are not automatically added to the project's solution. You can manually add externally generated session files to an open solution in Visual Studio.

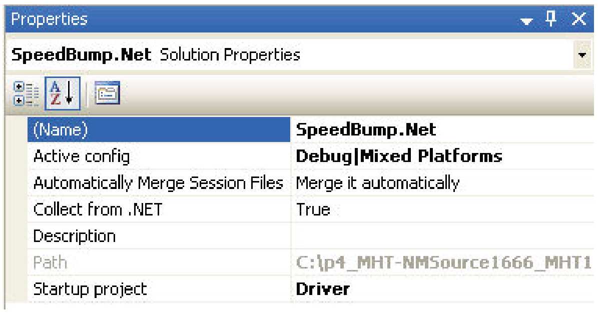
This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a performance analysis session, continue reading the rest of this chapter for additional information, or refer to the DevPartner online help for task-based information.

Setting Properties and Options

Before beginning a Performance Expert session, it is often useful to fine-tune data collection to include or omit certain types of information. Use Solution Properties, Project Properties, and DevPartner options to better focus your analysis session.

Solution Properties

To view properties that affect Performance Expert at the solution level, select the solution in the Solution Explorer and press F4 to view the Properties window.



Properties

SpeedBump.Net Solution Properties

(Name)	**SpeedBump.Net**
Active config	**Debug|Mixed Platforms**
Automatically Merge Session Files	Merge it automatically
Collect from .NET	True
Description	
Path	C:\p4_MHT-NMSource1666_MHT1
Startup project	**Driver**

Figure 7-9. Solution properties

The following Solution properties may affect Performance Expert:

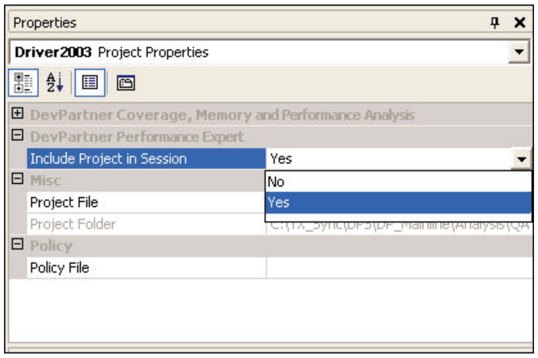
- **Collect from .NET -** Running your managed application with Performance Expert overrides this property if it is set to **False**. Performance Expert always collects data from managed applications.
- **Startup project -** If your solution includes multiple projects, you can change the startup project. The **Project** properties for the startup project govern data collection for all projects active in the session.

Note that your solution must include a startup project. If the solution contains multiple startup projects, DevPartner prompts you to choose a startup project for the session before analysis begins.

Only projects for which the **Action** in the **Common Properties > Startup Projects** page of the solution properties is set to **Start** are included in the prompt dialog. If the desired startup project does not appear in the prompt, open the solution properties page and set the **Action** for the project to **Start**. If you choose a new startup project for a subsequent session, review the properties for the new startup project to ensure the data collection options are correct.

Project Properties

To review project level properties, select a project in the Solution Explorer and review the properties that can be set for projects within the solution.



Properties

Driver2003 Project Properties

DevPartner Coverage, Memory and Performance Analysis	
DevPartner Performance Expert	
Include Project in Session	Yes
Misc	
Project File	
Project Folder	C:\TX_Sync\DPS\DP_Mainline\Analysis\QA
Policy	
Policy File	

Figure 7-10. Project properties for Performance Expert

The following project-level property affects Performance Expert:

- **Include Project in Session -** To exclude a project from Performance Expert data collection, select **No**.

Options

To review DevPartner option settings for Performance Expert sessions, choose **DevPartner > Options > Analysis**.

- The **Display** option allows you to set the precision, scale, and units used when displaying your data.
- The **Session Control File** option allows you to create a set of rules and actions to control the data that DevPartner collects as your application or module runs. Refer to "Creating a Session Control File Within Visual Studio" on page 303 for more information about session control files.

Other Visual Studio options, such as the **Environment > Fonts and Colors** options, also affect DevPartner features.

Finding Application Problems with Performance Expert

Performance Expert helps you identify problems in managed Visual Studio applications in the following critical areas:

- CPU/thread use (including wait and synchronization issues)
- File and disk I/O
- Network I/O
- Synchronization wait time

When run from Visual Studio, Performance Expert analyzes a single process at a time. It reports data for any managed threads executing in the selected process. To analyze an additional process, select the second process and rerun Performance Expert. Performance Expert can also analyze a distributed application that spans multiple computers. For information about remote data collection, see "Collecting Data from Distributed Applications" on page 233.

DevPartner supports the Visual Studio launch model. When you click the Performance Expert icon or choose **Start without debugging with Performance Expert** on the **DevPartner** menu, DevPartner immediately launches the startup project for your application and begins to collect Performance Expert data.

In order to collect Performance Expert data for your application, the solution must contain at least one managed code project (for example, C#, Visual Basic, or managed C++). It must also include a startup project. For more information, see "Setting Properties and Options" on page 221

If You Get a Security Exception

If you see a security exception message when you attempt to collect data for a managed application, it means that your security policy prevented DevPartner instrumentation of your code. By default, assemblies must have the `SkipVerification` permission to be profiled. If you remove this permission from the permission set of the policy under which the code executes, or add imperative security declarations to the assembly that cause this permission to be revoked, the assembly can not be profiled.

To remedy this condition, enable secure profiling in one of two ways.

- Set the following global environment variable and retry profiling the application:

  ```
  NM_NO_FAST_INSTR=1
  ```

  This solution allows you to work around this issue, although it does exact a slight performance penalty.

- Change the policy for the assembly using the .NET Framework Configuration tool MMC snap-in, or by temporarily removing any imperative security declarations in the assembly.

See the *.NET Framework Developers Guide* in the Visual Studio online help for more information on security policy in Visual Studio.

Accounting for Child Methods

The calculation of the Performance Expert session data differs between the **Paths that uses the most CPU** and the **Individual methods that use the most CPU** views. DevPartner excludes measurements for source code child methods in computing data for CPU time, disk or network I/O, and synchronization lock wait time in the Individual method analysis views. In contrast, DevPartner includes the impact of source code child methods to their parent methods in the Path analysis views.

All computations in both views include time or throughput attributable to system or .NET Framework methods called by your source code methods. Managed applications typically spend a lot of time executing Framework code. Performance Expert charges the system data to the lines in your source code that made the calls in order to focus attention on how your code interacts with the Framework, that is, on the parts of the application that you can modify.

For more tips on collecting and analyzing the session data, see “Usage Scenarios” below.

Usage Scenarios

The typical methodology for resolving performance issues consists of the following steps.

1. Locate the slowest line in a problem method and optimize it.
2. If you cannot optimize the line, remove it or execute it less often.

In the simplest cases, you may be able to locate the slowest line in a method (e.g., by using Performance Analysis) and either optimize it or call it less often. However, in real world application development, many problems have more complex causes. You may be able to identify the slowest method, only to find that a combination of lines within the method is slowing execution. In such a case, additional targeted data can help you analyze the problem quickly.

For example, if the slowest part of your application does a lot of network I/O, the following metrics would likely help you understand the nature of the problem:

- Total number of network reads and writes
- Number of bytes read or written
- Number of read or write errors
- Elapsed time for read or write operations

If your application did a lot of disk I/O, you would want to see metrics that reflected read/write volume and the efficiency of those operations. Performance Expert reports exactly this kind of data.

You can use Performance Expert to analyze CPU and thread performance, disk I/O, network I/O, and synchronization wait time. The following examples illustrates ways in which you can use Performance Expert to improve application performance.

Identifiable Performance Problem

Scenario: *Usability testers have reported that specific operations in your application are too slow. As a developer, you want to locate the parts of your source code that are responsible for the slow operations taking so long to complete and fix them.*

Assume that the slowest part of your application has run under Performance Expert as described in "Go: Collect Performance Expert Data" on page 209. When you examine the session file, you immediately see the method that took the longest time to execute at the top of the **Individual methods that use the most CPU** graph. However, in a complex application, a single slow method may affect performance less than a sequence of moderately slow methods. The slowest calling sequences appear in the **Paths that use the most CPU** graph. Do some methods appear in both graphs? If so, these methods definitely deserve scrutiny.

You also notice that some of the methods in the graphs are marked with icons that indicate disk I/O or network I/O activity in the method. These indicators tell you something about the kind of processing done by these methods.

Disk activity

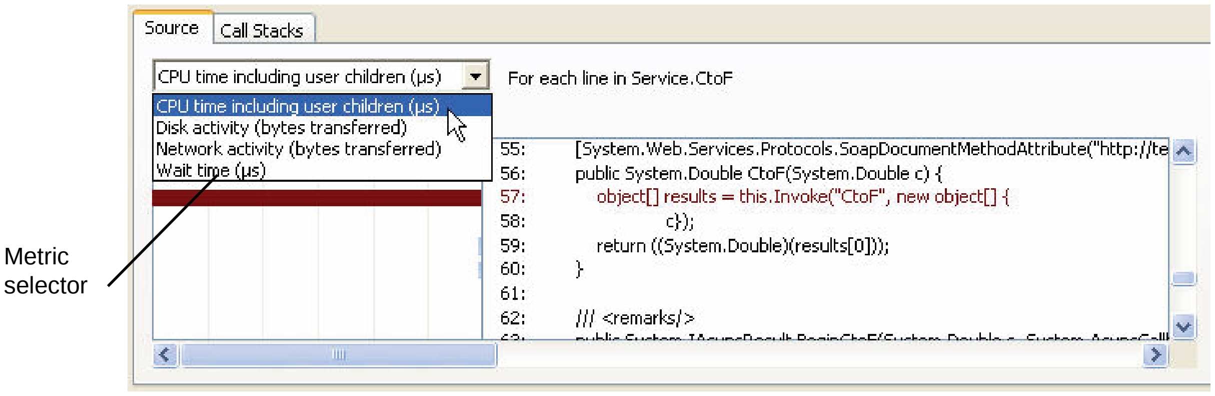
Network activity

At the bottom of the results summary, Performance Expert displays the **Total elapsed time** and **Total execution time.** If the execution time is very small relative to elapsed time, and you have exercised the application in such a way that you are reasonably sure the difference is not simply due to waiting on user input, check to see if some methods in your application are spending more time waiting for locks than they should.

Assume that you first decide to examine the top method in the **Individual methods that use the most CPU** graph. Many factors can affect CPU utilization: processor-intensive computations, disk I/O, network I/O, or inefficiently used synchronization objects. Similarly, you know that wait time can have multiple causes: the resource your method is waiting for could be shared within the same process, or with an external process. But how do you quickly determine what is going on in your application?

Click the top method in the **Individual methods that use the most CPU** graph to open the **Methods** detail view for the method. Notice the data in the columns in the **Methods** table. This information should help you determine what the method is doing. If the method was marked with the disk activity icon in the graph, right-click in the table and use the **Choose Columns...** dialog box to add all of the disk-related columns to the table. You might find that the method is producing read or write errors, or is using a large amount of time to write small amounts of data, and is being executed many times.

The **Source** tab in the lower half of the Methods window shows you the source code for any method you select in the table. When you click on a method in the table, the source automatically scrolls to the line that consumed the most CPU time and indicates the time attributable to that line. The view also indicates graphically other lines that used CPU time.


Metric selector
Source
Call Stacks
CPU time including user children (μs)
CPU time including user children (μs)
Disk activity (bytes transferred)
Network activity (bytes transferred)
Wait time (μs)
For each line in Service.CtoF

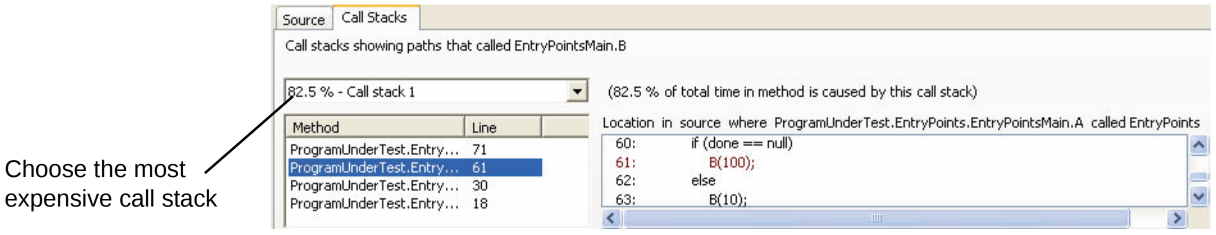
```
55:     [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://te
56:     public System.Double CtoF(System.Double c) {
57:         object[] results = this.Invoke("CtoF", new object[] {
58:                     c});
59:         return ((System.Double)(results[0]));
60:     }
61:
62:     /// <remarks/>
```

Figure 7-11. Locating problem lines in the Source tab

If the method performed disk or network I/O, or had wait time, expanding the metric selector at the upper left lists those selections, so you can immediately locate the most significant line in the method for that metric. For example, Choose **Disk activity** from the drop-down list to immediately go to the line that transferred the most bytes, and to see relative disk activity for other lines in the method. If the method involves **Wait time,** check that view too. Notice which lines are associated with long wait times. In each view, DevPartner selects the most expensive line by default. Comparing these views of the lines in the method shows you where to focus your efforts much more quickly than traditional debugging techniques.

When you have located an appropriate line to fix, double-click on it to jump to that line in your source code in Visual Studio.

If a way to fix the problem is not obvious, click the **Call Stacks** tab to see all the ways the method was used as your application executed. Is the problem method called by more than one path? If so, examine the call stacks that are responsible for the most time in the method.


Choose the most expensive call stack
Source
Call Stacks
Call stacks showing paths that called EntryPointsMain.B
82.5 % - Call stack 1
(82.5 % of total time in method is caused by this call stack)
Location in source where ProgramUnderTest.EntryPoints.EntryPointsMain.A called EntryPoints

Method	Line
ProgramUnderTest.Entry...	71
ProgramUnderTest.Entry...	61
ProgramUnderTest.Entry...	30
ProgramUnderTest.Entry...	18

```
60:         if (done == null)
61:             B(100);
62:         else
63:             B(10);
```

Figure 7-12. Finding the most expensive call stack

Tip: Performance Expert records a unique parent branch if any method (or calling line in the same method) in the call stack is different.

Look first at the parent path responsible for the highest percentage of calls. Try to modify your code to eliminate the calls, or call less frequently. The **Call Stacks** tab includes a view of your source code. When you select a method in the stack, the source automatically scrolls to the line where the call to the next method in the stack was made. A double-click opens the line in Visual Studio, so you can quickly modify the calling sequence if necessary. Once you have made the changes to your code, run the application again with Performance Expert to verify the improvement.

Scaling Problem in an Application

Scenario: *Your new Web application runs fine when you test it on your computer. But when you allow additional users to access the application, it is too slow. You have a looming deadline. How do you quickly determine what is wrong?*

You can collect Performance Expert data while stressing your application with a load-testing tool. To do so, start and stop your application with a command line tool or script. DevPartner provides a command line utility called **`DPAnalysis.exe`** for this purpose. For information on running a Performance Expert session from a command line, see "Automating Data Collection" on page 231. For example, you could do something like the following:

1. Start the application under Performance Expert with **`DPAnalysis.exe`**.
2. Run the load-testing application.
3. Stop the application.
4. Examine the Performance Expert session data.

Assume that when you look at the session file, no single method in the **Individual methods that use the most CPU** graph stands out as the likely culprit. It is a complex application, and it is probable that several methods contribute to the sluggish performance. Start analysis with the **Paths that use the most CPU** graph in the results summary. This graph shows a list of methods, but in this case each method represents an **entry point**. An entry point method is not called by another source code method. In other words, it is an entry point into the execution of code that you wrote. Most important, it marks the beginning of an execution path that you can change, either by modifying the methods, or the way they are called. The entry point method that corresponds to the most expensive path of execution in your application appears at the top of the graph. Click on the method to open the **Path analysis** view.

The Call Graph

When you open the Call graph from the Results Summary, DevPartner places the most expensive paths at the top of the Call graph, and highlights the most expensive child path whenever a path branches. As you examine the data, investigate the most expensive child paths first. To investigate a path, expand the nodes to the right.

Tip: The percentages on lines connecting a method to the child methods it called are additive; those on lines connecting the chain of methods in a single path are not.

To determine the relative contributions of different paths spawned by the same method, compare the percentage values on the lines that connect the selected method to each of its child paths. The value on each link represents the percent of time in the parent method attributable to child methods called in that path. Thus, in Figure 7-13 on page 228, the method `Form.Main` called `Form.CtoF`, `Form.ParseOption`, and `Form.FtoC`. The value on the line that links `Form.Main` to `Form.CtoF` is 98.1%, while the remaining 1.9% is spread among the other called paths. This means that the path `Form.Main` calls `Form.CtoF` accounted for 98.1% of the CPU time spent in `Form.Main` that was attributable to the execution of child methods. Start your troubleshooting with this path.

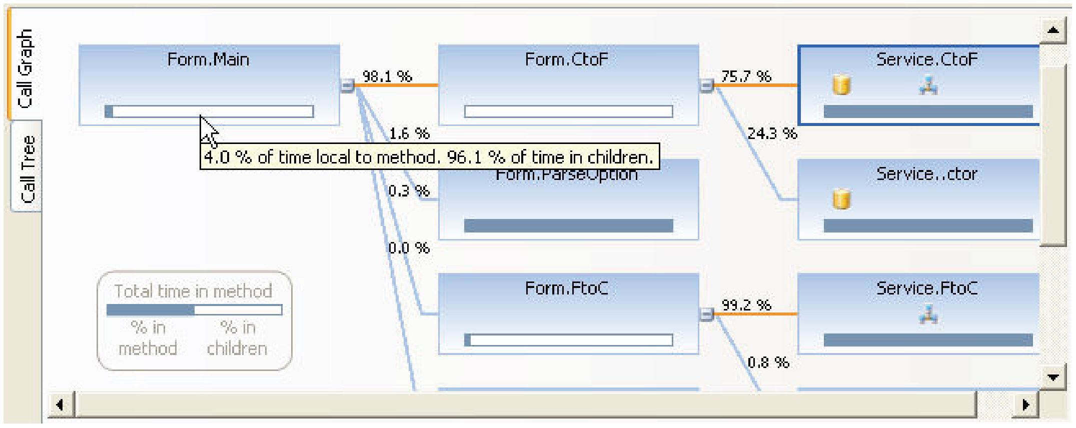


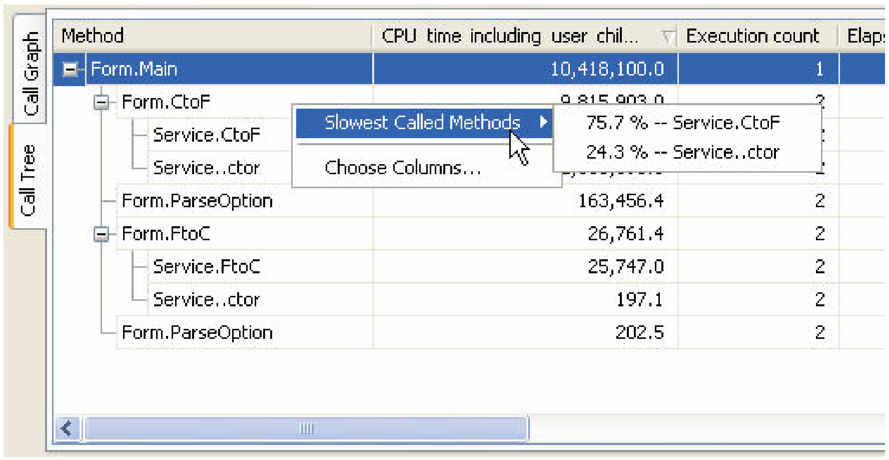
Figure 7-13. Understanding the impact of child methods

As you investigate the called path, notice the horizontal bar at the bottom of each node. The bar shows the relative percentages of time in the method due to the method body compared to the child methods it called. Hover over the bar with the mouse to see the actual percentages. Use this bar to guide your tuning efforts. For example, if 4% of time is spent in the method body, and 96% of time is attributable to child methods, continue to investigate the most expensive called paths to locate the child methods that are affecting performance. Fix those methods or change your code so they can be called less often. If, on the other hand, 96% of the time was spent in the method body, focus your efforts there.

Also notice whether an expensive node contains the disk activity, network activity, or wait time icons. Hover over the icon with the mouse to view the magnitude of the activity. If a node contains one or more of these icons, consider switching to the Call tree view and adding the appropriate data columns for more help in diagnosing the problem.

The Call Tree

The default sort of the Call tree table is by **CPU time including user children**. To gain an idea of where the bulk of the time is being spent, scan the values in the other columns. Doing so determines whether wait time, disk or network I/O, or CPU-intensive processing is the major factor. If you need more detail, you can add additional columns, such as disk or network reads, writes, and errors, to the display.



Method	CPU time including user chil...	Execution count	Elaps
Form.Main	10,418,100.0	1	
Form.CtoF	9,815,903.0	2	
Service.CtoF			
Service..ctor			
Form.ParseOption	163,456.4	2	
Form.FtoC	26,761.4	2	
Service.FtoC	25,747.0	2	
Service..ctor	197.1	2	
Form.ParseOption	202.5	2	

Figure 7-14. Displaying additional data for the selected method in the call tree

For example, if an expensive method in the Call graph shows `network I/O`, select it, switch to the Call tree, and add all of the network-related data columns to the table. To add columns, right-click in the Call tree table and select **Choose columns...** from the context menu. See the Performance Expert online help for a full explanation of the data reported in each column.

Tip: The term "user" in "user children" or "user methods" refers to your source code methods.

Whether you are using the Call graph or Call tree, the session file window includes the **Source** and **Call Stacks** tabs. These tabs function as they do in the Methods table, except that the data is calculated to include data attributable to user, or source code, child methods. Use the **Source** tab to immediately locate the most expensive line in any method you select in the Call graph or Call tree. Use the **Call Stacks** tab to see the relative impact of other paths that called the method and to locate the line that called the selected method in the stack. Double-click a line of code in either the **Source** or **Call Stacks** tab to jump to that line in Visual Studio for editing.

Performance Slow but No Specific Issue

Suppose your application is generally sluggish, but you cannot identify a specific issue. Performance tuning is an iterative process. You can still use the techniques described above to try to improve performance.

- Run the application under Performance Expert.
- Go through the **Paths that use the most CPU** and try to optimize the most expensive branches for each critical path.
- Go down the list of **Individual methods that use the most CPU** in the same way and try to optimize the top methods in the list.
- Retest to verify improvement.

Collecting Data from Web Applications

You can collect Performance Expert data for any managed application, including Web applications. When you run a Web application with Performance Expert, be aware of the following.

Managed Code Only

Unlike some other DevPartner features, Performance Expert collects data for managed applications exclusively. Therefore if your application uses Internet Explorer as the client, do not expect to see Internet Explorer data in the session file. DevPartner displays server-side data for your ASP.NET or Web service application.

web.config Requirements

For Performance Expert to successfully profile an ASP.NET application, the following two conditions must be met:

- The project must include a **`web.config`** file.
- The project must be configured for debugging. To do this, the **`web.config`** file must include a compilation element with the debug attribute set to true. For example:

```
<compilation debug=”true” />
```

Multiple Process Profiling

When run from the Visual Studio IDE or from the command line using the DevPartner command line switches, Performance Expert collects data for a single process or service per session. If your application runs in more than one process, or if you need to collect data for a service, such as IIS, as well as the process your target application runs in, you can use **`DPAnalysis.exe`** (a command line executable version of DevPartner analysis tools) and target an XML configuration file to manage the session. For more information see “Using DPAnalysis.exe with an XML Configuration File” on page 288.

Note: Although you can collect data (in separate session files) from two or more processes or services simultaneously by using **`DPAnalysis.exe`** with an XML configuration file, Performance Expert is generally best run on a single process at a time. Data collection overhead for multiple processes can affect interaction of the processes, as well as slowing the applications and inflating elapsed time values. If you collect Performance Expert data for multiple processes simultaneously, large timing values for disk I/O, network I/O, or synchronization wait time may reflect inflation by profiling overhead. Rerun the session targeting a single process to confirm that the timing values are large enough to merit investigation.

Single Process Profiling on IIS 6.0

On IIS 6.0, Performance Expert collects data for only one worker process. On IIS there is one worker process per application pool. Therefore, if you run a Web service and a Web service client on your system, and both execute in the same application pool, Performance Expert gathers data for both, even if you started the service under Performance Expert and started the client in a separate instance of Visual Studio without Performance Expert. If you change the application so the client executes in a different application pool, Performance Expert gathers data only for the application (in this case, the service) started with Performance Expert.

No Remote Session File for Components Running Under DLLHOST

When running Performance Expert for a process that interacts with **`dllhost.exe`** on a remote system, a final session file is not generated on the remote system when **`dllhost.exe`** terminates.

Source Code on Remote Computers

DevPartner Studio assumes that the source file exists on the same computer as the open session file.

- If a **File** > **Open** dialog box appears when you attempt to view the source code, use it to browse to the correct location on the remote computer.
- If you have collected data for a remote ASP.NET application, you may need to look up the value of the **Local Path** entry in the **Virtual Directory** tab of the IIS settings for the target Web site in order to browse to the source file.

Session Files Saved to Open Solution

DevPartner session files are saved with the current solution. Opening a Web project from IIS directly, as opposed to opening the project through Visual Studio, may cause a different solution file to be used. DevPartner session files created in the first solution would not be visible in the second solution.

Automating Data Collection

Performance Expert supports command line execution through an executable called **`DPAnalysis.exe`**. This file is located in your **`\Program Files\Micro Focus\DevPartner Studio\Analysis\`** folder.

Note: For installs on 64-bit versions of Windows, DevPartner Studio is located at: **`\Program Files (x86)\Micro Focus\`DevPartner Studio`\Analysis\.`**

You can run an application under Performance Expert from a command prompt, or create batch files to automate data collection. You can launch the Performance Expert session in two ways:

- Specify the target and arguments in standard MS-DOS command line syntax
- Specify an XML configuration file that contains the targets and arguments for the session

Using Command-line Switches

In the example in the section "Scaling Problem in an Application" on page 227. Quality Assurance engineers monitor scalability (or any other aspects of the application) on a daily basis by setting up an automated test (or suite of tests) to be run on the application every night. To automate the tests, set up a batch file to

1. Start the application under Performance Expert
2. Start the load-testing application and any other tests you want to run
3. Stop the application when the tests are complete

DevPartner automatically generates the session log file when the application exits.

The command line syntax to launch the session is:

```
DPAnalysis.exe /Exp /E /O /W /H [/P or /S] target {target arguments}
```

`/Exp` sets analysis type to Performance Expert

`/E` enables data collection for the specified process/service

`/O` specifies the session file output folder and/or name

`/W` specifies the working folder for the process

`/H` specifies the host computer on which the target runs

`/P` or `/S` specifies that the target is a process or a service; use only one

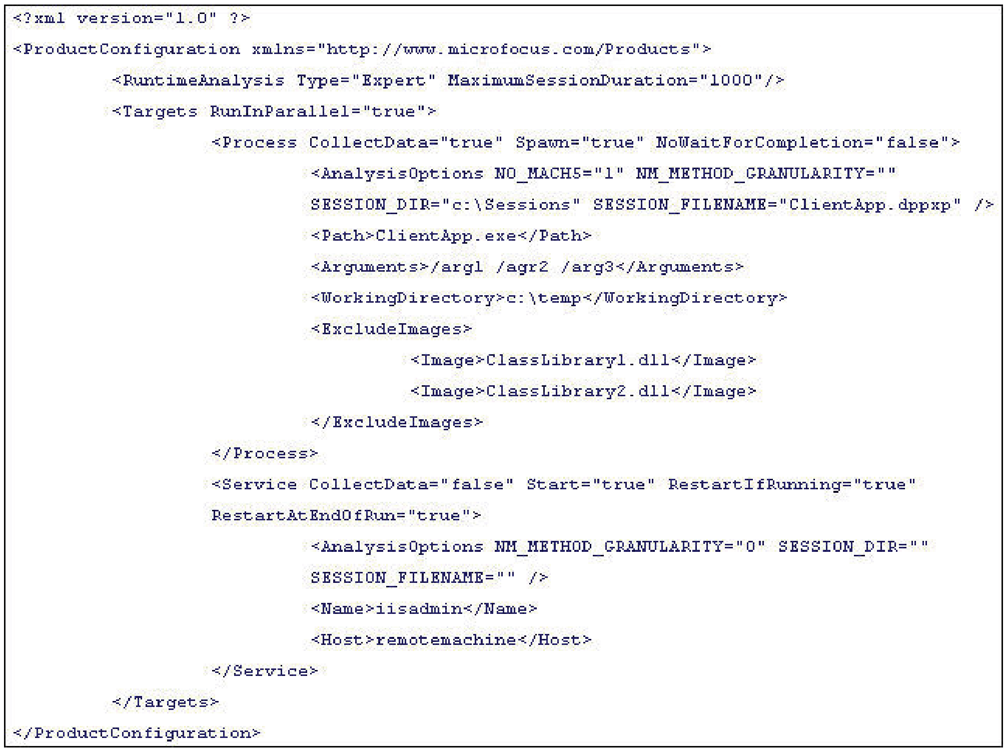
There is one restriction on the order in which the switches must appear: The `/P` or `/S` switch must occur last. Everything that follows either switch is interpreted as an argument to the process or service.

Using an XML Configuration File

To use an XML configuration file, the command line is even simpler.

```
dpanalyis.exe /C [path]configuration_file.xml.
```

The configuration file contains the necessary parameters for any type of DevPartner analysis, including some options that are not available using command line switches. For example, if you want to exclude application components from a Performance Expert session, you must use the `ExcludeImages` element in the configuration file.



```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.microfocus.com/Products">
        <RuntimeAnalysis Type="Expert" MaximumSessionDuration="1000"/>
        <Targets RunInParallel="true">
                <Process CollectData="true" Spawn="true" NoWaitForCompletion="false">
                        <AnalysisOptions NO_MACH5="1" NM_METHOD_GRANULARITY=""
                        SESSION_DIR="c:\Sessions" SESSION_FILENAME="ClientApp.dppxp" />
                        <Path>ClientApp.exe</Path>
                        <Arguments>/arg1 /agr2 /arg3</Arguments>
                        <WorkingDirectory>c:\temp</WorkingDirectory>
                        <ExcludeImages>
                                <Image>ClassLibrary1.dll</Image>
                                <Image>ClassLibrary2.dll</Image>
                        </ExcludeImages>
                </Process>
                <Service CollectData="false" Start="true" RestartIfRunning="true"
                RestartAtEndOfRun="true">
                        <AnalysisOptions NM_METHOD_GRANULARITY="0" SESSION_DIR=""
                        SESSION_FILENAME="" />
                        <Name>iisadmin</Name>
                        <Host>remotemachine</Host>
                </Service>
        </Targets>
</ProductConfiguration>
```

Figure 7-15. Specifying session details in the XML configuration file

To collect data for a process that runs on a remote computer, you must specify a folder and file name. Use the `SESSION_FILENAME` and `SESSION_DIR` elements in the Analysis options in the configuration file.

For detailed information about using the configuration file to manage data collection, see "Using DPAnalysis.exe with an XML Configuration File" on page 288.

QA engineers scan the session log file the following morning. If performance numbers deteriorate, QA sends the session log to the appropriate developers. This way, QA tracks the health of the application throughout the development cycle. If a problem appears, the development team has the session log file to use in quickly determining the nature of the problem. In addition, the development team knows that the problem was caused by a code change from the previous day, greatly reducing the amount of code it has to review to fix the problem.

For detailed information on using **`DPAnalysis.exe`**, see Appendix B, "Starting Analysis from the Command Line".

Collecting Data from Distributed Applications

DevPartner can collect Performance Expert data from distributed application components that run on remote systems, provided the remote systems are properly licensed for remote data collection. Before you launch a remote session, be aware that a Performance Expert session monitors a single process per run when run from Visual Studio or with **`DPAnalysis.exe`** from the command line using traditional command line syntax. Although the XML configuration file allows you to target more than one process or service in a single run of the application, it is usually best to target a single process in a Performance Expert session. If your application runs in multiple processes, simply rerun the application targeting the second process. Driving the application with a script or batch file ensures that you exercise the application identically in both sessions. For an overview, see "Automating Data Collection" on page 231.

If necessary, you can collect the data (in a separate session file) for the second process or service in a single run of the application if you use **`DPAnalysis.exe`** with the XML configuration file option. Although you can collect data from two or more processes or services simultaneously, be aware that data collection overhead for multiple processes can affect interaction of the processes, as well as slowing the applications and inflating elapsed time values. If you collect Performance Expert data for multiple processes simultaneously, large timing values for disk I/O, network I/O, or synchronization wait time may reflect inflation by profiling overhead. Rerun the session targeting a single process to confirm that the timing values are large enough to merit investigation.

Note: Any name and MAC address conflicts on the network can affect remote profiling functionality, especially when using cloned virtual machines.

Enabling Remote Data Collection with DPAnalysis.exe

`DPAnalysis.exe` cannot be used to spawn remote processes. It can only be used to enable data collection for processes on remote computers. For example, with the following command line:

```
DPAnalysis.exe /host remotemachine /p c:\MyDir\target.exe
```

`DPAnalysis.exe` sets up profiling for **`target.exe`** but does not attempt to start it on the remote computer. Profiling begins when **`target.exe`** starts on the remote computer by whatever means.

This is not the case for remote services, which can be started remotely. For example:

```
DPAnalysis.exe /host remotemachine /s servicename
```

This command enables profiling and attempts to start **`servicename`** on **`remotemachine`**

Optionally, you can use the XML configuration file to specify the parameters in the command line examples above. For detailed information about **`DPAnalysis.exe`**, see Appendix B, "Starting Analysis from the Command Line".

Note: Never use the Force Profiling switch `/f` for **`DPAnalysis.exe`** when remote profiling a moxed-mode or managed application. Doing so can potentially cause initialization issues.

Saving Session Files on Remote Computers

Session files for all four types of analysis (coverage, memory, performance, and Performance Expert) are saved on the remote computer in remote profiling scenarios. A folder and session file name must be provided on the command line or in the XML configuration file for remote processes or services. The folder specified must already exist on the remote computer. If no folder or file name is provided, a Save As dialog box appears on the remote computer.

Viewing the session file

Copy the session file to a computer with DevPartner Studio installed, such as the computer where the profiling was initiated and the client file is saved.

On the command line or in the XML configuration file, specify a mapped drive on the remote computer to save the session files to another computer with DevPartner Studio installed, such as the computer where the profiling was initiated.

Collecting Data with Terminal Services or Remote Desktop

DevPartner Studio supports Windows Terminal Services. See “Using Terminal Services and Remote Desktop” on page 21 for more on using DevPartner Studio with Terminal Services.

Remote Profiling and Windows XP Service Pack 3 (SP3) or Earlier

Windows XP SP 3 increased security levels for remote applications. The new security settings can prevent DevPartner from collecting data on some server-side application components when profiled from Visual Studio. To collect data from application components on a remote computer, modify the security settings on all Windows XP SP3 or earlier operating systems (both the remote computer and the client computer where profiling is initiated) in the session.

The procedures that follow describe three ways to alter these security settings to allow remote profiling.

Adding DevPartner Control Service to the Windows Firewall Exclusion List

If the Windows Firewall service is enabled, add the DevPartner Control Service to the Firewall's exclusion list. Follow these steps:

1 From the **Start** menu, select **Control Panel**.

2 From the **Control Panel**, select **Windows Firewall**, then select the **Exceptions** tab.

3 On the **Exceptions** tab, click **Add Program**.

4 In the **Add a Program** dialog box, click **Browse**, then navigate to **`NCS.exe.`** The default location for this executable is:

```
C:\Program Files\Micro Focus\DevPartner Studio\Analysis\NCS.exe
```

Note: For installs on 64-bit versions of Windows, this executable is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\Analysis\NCS.exe.`**

5 Click Open in the **Browse** dialog box to select **`NCS.exe`**, then click **OK** to close the **Add a Program** dialog box.

6 On the **General** tab of the **Windows Firewall** control panel, clear the **Don't allow exceptions** check box.

Modifying Security Settings on Both Remote (Server) and Local (Client) Computers

Follow these steps to modify the security settings:

1 In the **Control Panel**, open **Administrative Tools > Local Security Policy > Local Policies > Security Options**.

2 Open the Properties page for **DCOM: Machine Access Restrictions in Security Descriptor Definition Language (SDDL) syntax**.

3 Select **Edit Security**.

4 Add an **Anonymous Logon** user, if one does not already exist.

5 Give the **Anonymous Logon** user both **Local** and **Remote** access.

If Visual Studio is running when the settings are changed, you must restart Visual Studio for the new settings to take effect.

Relaxing COM Security on the Client Computer

To relax COM security, follow these steps on the client computer where profiling is initiated:

1 From the **Start** menu, select **Control Panel**.

2 From the **Control Panel**, select **Administrative Tools**; from the **Administrative Tools** window, open **Component Services**.

3 In the **Component Services** window, navigate to **My Computer**, right-click on **My Computer** and select **Properties**.

4 On **My Computer Properties**, select the **COM Security** tab.

5 Under **Launch and Activation Permissions** on the **COM Security** tab, click **Edit Limits** and make these changes:

6 Click **Add** and enter `NETWORK`.

7 Make sure that the **Allow** check box is selected for **Local Launch**, **Remote Launch**, **Local Activation**, and **Remote Activation**.

8 Under **Launch and Activation Permissions** on the **COM Security** tab, click **Edit Default** and make these changes:

9 Click **Add** and enter `NETWORK`.

10 Make sure that the **Allow** check box is selected for **Local Launch**, **Remote Launch**, **Local Activation**, and **Remote Activation**.

Remote Profiling and Windows Vista Service Pack 1 (SP1) or Later

Supported Windows Vista SP1 and later operating systems (including Windows 7) feature increased security for remote applications. These security settings can prevent DevPartner from collecting data on some server-side application components when profiled from Visual Studio.

Windows Firewall inbound and outbound program and port rules must be set up to allow remote profiling on Windows Vista SP1 and later operating systems. Component access and local security policy must also be set up to allow remote profiling.

Anti-virus programs such as Symantec Endpoint Protection can define inbound and outbound rules. Rules defined in an anti-virus or other firewall management program take precedence over rules defined by the Windows firewall utility. If your anti-virus program can define inbound/outbound firewall rules, define the firewall rules described in the sections below using the anti-virus program rather than the Windows Firewall console.

If your anti-virus program does not define the firewall rules, use the following procedures to allow remote application profiling. These procedures modify security on all Windows Vista SP1 and later operating systems (both the remote computer and the client computer where profiling is initiated) in a session. You must be logged on as a user with administrative priveleges to perform these procedures.

Adding Windows Firewall Inbound/Outbound Program Rules on Remote (Server) and Local (Client) Computers

For computers using Windows Vista or later operating system (OS), you must add an inbound/outbound rule to the Windows Firewall if the Windows Firewall service is enabled. You must be logged on as a user with administrative privileges to perform the following steps. Perform these steps on both the remote and local computers.

1 Click the Windows **Start** button and select **Control Panel**.

2 From the Control Panel, select **Windows Firewall**.

3 In the Windows Firewall utility, select **Advanced Settings**. The Windows Firewall with Advanced Security window appears.

4 In the left hand pane select **Inbound Rules**, then in the right hand Actions pane, select **New Rule**. The New Inbound Rule Wizard appears.

5 Select the **Program** radio button and click **Next**.

6 Select the **This Program with** radio button and click **Browse**.

7 In the **Open** dialog box, navigate to the location of the DevPartner Analysis folder and select **`NCS.exe`**.

   For DevPartner installations on supported 32-bit operating systems, this file is located in **`\Program Files\Micro Focus\DevPartner Studio\Analysis\`**.

   For DevPartner installations on supported 64-bit operating systems, this file is located in in **`\Program Files (x86)\Micro Focus\DevPartner Studio\Analysis\.`**

8 Select the **Allow the connection** radio button. Click **Next**.

9 Select all check boxes. Click **Next**.

10 Type descriptive name and description. Click **Finish** to add the rule.

11 In the left hand pane select **Outbound Rules**, then in the right hand Actions pane, select **New Rule**. The New Outbound Rule Wizard appears.

12 Repeat steps 5-10 to add the outbound rule.

Adding Windows Firewall Inbound/Outbound Port Rules on the Remote (Server) Computers (Windows Vista OS or Later)

For computers using Windows Vista or later operating system (OS), you must add an outbound rule to the Windows Firewall if the Windows Firewall service is enabled. You must be logged on as a user with administrative privileges to perform the following steps. Perform these steps only on the remote computer.

1 Click the Windows **Start** button and select **Control Panel**.

2 In the Control Panel, select **Windows Firewall**.

3 In the Windows Firewall utility, select **Advanced Settings**. The **Windows Firewall with Advanced Security** window appears.

4 In the left hand pane, select **Inbound Rules**, then in the right hand Actions pane, select **New Rule**. The New Inbound Rule Wizard appears.

5 Select the **Port** radio button and click **Next**.

6 Select the **TCP** radio button and the **Specific local ports** radio button.

7 In the Specific local ports text box, type `37493` for the port number. Click **Next**.

8 Select the **Allow the connection** radio button. Click **Next**.

9 Select all check boxes. Click **Next**.

10 Type descriptive name and description. Click **Finish** to add the rule.

11 In the left hand pane, select **Outbound** Rules, then in the right hand Actions pane, select **New Rule**. The New Outbound Rule Wizard appears.

12 Releat steps 5-10 to add the outbound rule.

Note: Make sure the firewall rules necessary for remote profiling take priority over existing firewall rules if the firewall has the means to prioritize rules. The topmost rule in the list takes the highest priority.

Adding Access Permissions on Both Remote (Server) and Local (Client) Computers

For computers using Windows Vista SP1 or later operating system, perform the following steps on both the remote and local computers to allow component access for remote profiling.

1 Click the Windows **Start** button and type `DCOMCNFG.EXE` in the **Search** box.

2 Click the **`DCOMCNFG.EXE`** program entry from the **Search** box result to start Component Services.

3 On the left pane tree view, navigate to **Console Root > Component Services > Computers > My Computer**. Right-click on **My Computer** and select **Properties** from the context menu.

4 In the **My Computer Properties** dialog box, select the **COM Security** tab.

5 Under **Access Permissions**, select the **Edit Limits...** button. The Access Permissions dialog box appears. Click **Add**.

6 In the **Select Users, Computers, Service Accounts, or Groups** dialog box, type `Authenticated Users` in the **Enter the object names to select** box. Click **OK** to add the name and return to the **Access Permissions** dialog box.

7 For Windows 2008 and 2008R2 operating systems only, do the following:

   a Type a semicolon (`;`) after the `Authenticated Users` entry and type `Distributed COM Users`.

   b Click **OK** to add the names to the **Group or user names** list.

   c Select the `Distributed COM Users` entry and select the **Allow** check box for both Local Access and Remote access. Click **OK**.

8 In the **Access Permissions** dialog box, click the `Authenticated Users` entry and select the **Allow** check box for both Local Access and Remote access. Click **OK**.

9 For Windows 2008 and 2008R2 operating systems only, you must manually add launch and activation permissions for the `Distributed COM Users` entry. Do the following:

   a Under **Launch and Activation Permissions**, select the **Edit Limits...** button. The Launch and Activation Permission dialog box appears.

   b Click the `Distributed COM Users` entry and select the **Allow** check box for Local Launch, Remote Launch, Local Activation, and Remote Activation. Click **OK.**

10 Click **OK** to apply changes and to close the **My Computer Properties** dialog box.

11 Restart the computer to allow the changes to take effect.

Setting Up Local Security Policy on Both Remote (Server) and Local (Client) Computers

For computers using Windows Vista SP1 or later operating system, perform the following steps on both the remote and local computers to set up the local security policy for remote profiling. You must be logged on as a user with administrative privileges perform these steps.

1. Click the Windows **Start** button and select **Control Panel**.
2. In the Control Panel, select **Administrative Tools**.
3. In Administrative tools, select **Local Security Policy**. The Local Security Policy dialog box appears.
4. On the left pane tree view, navigate to **Local Policies > Security Options**.
5. In the right pane, right-click **Network Access: Let Everyone permissions apply to anonymous users** and select **Properties**.
6. Select the **Enabled** radio button and click **OK**.
7. Restart your computer for the changes to be in effect.

Firewalls and Remote Data Collection

To collect session data from remote computers, DevPartner connects to a previously installed service whenever DevPartner runs, either within Visual Studio or via **`DPAnalysis.exe`**. This service listens for interprocess communication traffic at the internet address `0.0.0.0 port 37493`. This service connection may trigger some firewall alarms. You can configure your firewall to trust this address to discontinue these alarms. If your firewall is set to maximum security levels, it may prevent DevPartner remote data collection. Reconfigure your firewall to enable data exchange at the address `0.0.0.0 port 37493`.

If ever the Local and Remote machines are running in a virtual environment (e.g. VMWare Workstation), make sure that the security application (e.g. Symantec Endpoint Protection) of the Host machine allows communication on the IPs used by both Local, Remote machine and the virtual network.

Exporting DevPartner Data to XML Format

You can export Performance Expert data to an XML format. Exporting data in XML format allows you to more easily use your own or third-party software to analyze the data, integrate the data with data produced by other tools, and archive the data in a data warehouse.

You can export Performance Expert session files (with the **`.dppxp`** extension) to XML format. When a saved Performance Expert session file is open, the **Export DevPartner Data** command is available on the **File** menu.

You can also export XML data from the command line, as described in “Exporting Analysis Data to XML” on page 311.

In the DevPartner Studio installation folder, the file **`DevPartnerPerformanceExpert`***`xx`***`.xsd`** describes the XML schema that is used by Performance Expert to export session files.

Using Performance Expert with Performance Analysis

Performance tuning is an iterative process. Use Performance Expert in conjunction with DevPartner Studio Performance Analysis. First, run your application with performance analysis and save the session file to capture a baseline view of performance. Then use Performance Expert to troubleshoot difficult problems, especially problems that involve disk or network I/O, or synchronization issues. When you have fixed a problem, run the application in a performance analysis session and use the performance analysis Session Comparison feature to verify the improvement. For example:

1. Run your application with performance analysis.
2. Notice the methods that appear to be slowing performance.
3. If a way to fix the problem methods is not immediately obvious, run an identical session with Performance Expert.
4. Check to see if the problem methods appear in the **Paths that use the most CPU** or the **Individual methods that use the most CPU** graphs.
5. Click the method in the **Paths that use the most CPU** graph to open the Call graph. The Call graph shows the method in context and indicates whether the method itself or its child methods are responsible for the performance issue.
6. Notice whether the problem method is marked with the disk, network, or wait time icons.

   If, for example, the method indicates network activity, switch to the **Call Tree** tab and use the context menu to add the network-related data columns to the view. The additional data can help you determine whether the problem is due to read activity, write activity, or to read or write errors. If you drilled into the data from the **Individual methods that use the most CPU** graph, you can add the data columns to the Methods table.
7. Use the **Call Stacks** tab to see how many ways the problem method was called, and which call stack was the most expensive.
8. Use the **Source** tab to locate the offending lines of code and jump to the source file to edit in Visual Studio.

Once you have fixed the problem, run the application in a second performance analysis session. Using the previous performance analysis session file as a baseline, compare the sessions with the performance analysis Session Compare feature to verify the improvement.

Performance Expert and performance analysis are complementary, but there are differences in the way they compute timing data. If you run a performance analysis session that includes system images and a Performance Expert session on the same application, you may notice that the performance analysis **Top 20 Source Methods** and the Performance Expert **Individual methods that use the most CPU** do not contain exactly the same methods, or that the methods do not appear in the same order.

In a performance analysis session, the percent of time spent in a method (**% in Method** column) is computed without user or system child methods. In a Performance Expert session, the percent of time spent in the methods that appear in the **Individual methods that use the most CPU** and **Methods** tables includes time spent in system children.

If you have done any performance profiling, you may have noticed that managed applications spend a lot of time executing methods in the .NET Framework. Including system children in the Performance Expert results focuses attention less on methods that take a long time to execute in themselves, and more on methods in your source code that exhibit problems in the way they interact with system code. You cannot do anything about time spent in system code once it begins to execute, but you can change how and when your code calls system code. Performance Expert helps you quickly identify these problem areas.

You cannot compare a performance analysis session file directly to a Performance Expert session file. You can only compare performance analysis session files.

Performance Expert in the Development Cycle

Use Performance Expert throughout the software development cycle. Many members of the engineering team can benefit from using Performance Expert at several points in the software project life cycle.

Software Designers

Software designers must often develop prototypes that meet specific requirements, for example, in response time or scalability. Before producing the final design, the designer must identify the operations and, if possible, the methods, that are preventing the prototype from meeting the performance requirements. Ideally, the designer would like to be able to identify a few methods that, if fixed, would give a dramatic performance boost.

Software designers can use Performance Expert during the design and prototype phase to improve the speed and efficiency of their code. As the design progresses, regular testing helps to ensure that the prototype code meets minimal performance requirements. When the prototype is handed off to the development team, developers can feel comfortable reusing sections of the prototype, knowing that it has been tested for several critical performance issues.

Software Developers

Software developers should use Performance Expert frequently during development. Consider running Performance Expert in addition to unit tests prior to code check-ins. Just as the unit tests ensure that the component does what it is supposed to do without breaking other components, Performance Expert provides early warning of potential performance issues before the component is fully integrated into the application and therefore more difficult to fix.

The software development team builds the application based on the designer’s prototype and specification. As soon as the application (or application components) can be tested and run, developers can integrate Performance Expert into their automated testing routines in order to identify potential CPU usage, file I/O, or network I/O issues as they are coding and debugging. Developers can review the Performance Expert session log each morning to see if the previous day’s coding has introduced any new performance issues and address issues immediately. When coding is complete, the development team submits the final Performance Expert session log to document that performance goals have been met.

Quality Assurance Engineers

Quality Assurance teams can use Performance Expert to continuously monitor application performance. QA can easily integrate Performance Expert into automated test suites to obtain a daily reading of application performance in critical areas. When problems appear, QA teams can send the session log to the development team or attach the log to a bug report in a defect tracking system.

Designated engineers can review critical metrics in the session log files on a daily basis. If the session log suggests a problem, the QA engineer can send the log file to the responsible developer so the problem can be addressed immediately.

Thus, all members of the software development team can benefit from running Performance Expert, from the design phase to final quality assurance testing. There is even a benefit for product management. At each critical milestone, Performance Expert session logs, coupled with before-and-after performance analysis session files, can be used to document that the product meets performance expectations.

Submitting Data to Visual Studio Team System

With Visual Studio 2008 or earlier, submit data as a Work Item through Visual Studio Team System of the type **Bug** for a selected item. In Visual Studio 2010 or later, submit data as a Work Item through Team Foundation Server of the type **Issue**, **Bug**, or **Defect** for a selected item. For a coverage analysis session file, access work item submission in for a method selected in the Method List tab. When submitting a work item, DevPartner populates the **Work Item** form with data from the visible columns in the **Methods List** tab. To change the method data you submit in the **Work Item**, change the columns displayed in the **Method List**. DevPartner Studio supports Microsoft Visual Studio Team System if the Team Explorer client is installed and a Team Foundation Server connection is available.

You can submit method-level data from a Performance Expert session file as a Visual Studio Team System **Work Item** of the type bug. The **Submit Work Item** command is available on the context menu for a method selected in the following Performance Expert views:

- The **Methods** table in the Methods detail view
- The Call tree in the Path analysis view

When you submit a work item, DevPartner populates the **Work Item** form with data from the visible columns in the **Methods** table or **Call Tree** view. To change the method data you submit in the **Work Item**, change the columns displayed in the method view.

For more information about DevPartner Studio integration with Visual Studio Team System, see “Visual Studio Team System Support” on page 20.

Chapter 8

System Comparison

This chapter contains two sections. The first section provides a quick-start procedure to get first-time users up and running with System Comparison. The second section provides reference information for an in-depth understanding of DevPartner's System Comparison feature.

Refer to the System Comparison online help for additional task-oriented information about comparing systems.

What is System Comparison?

System Comparison compares two computer systems, or compares the current state of a computer with a previous state, allowing you to determine why your application:

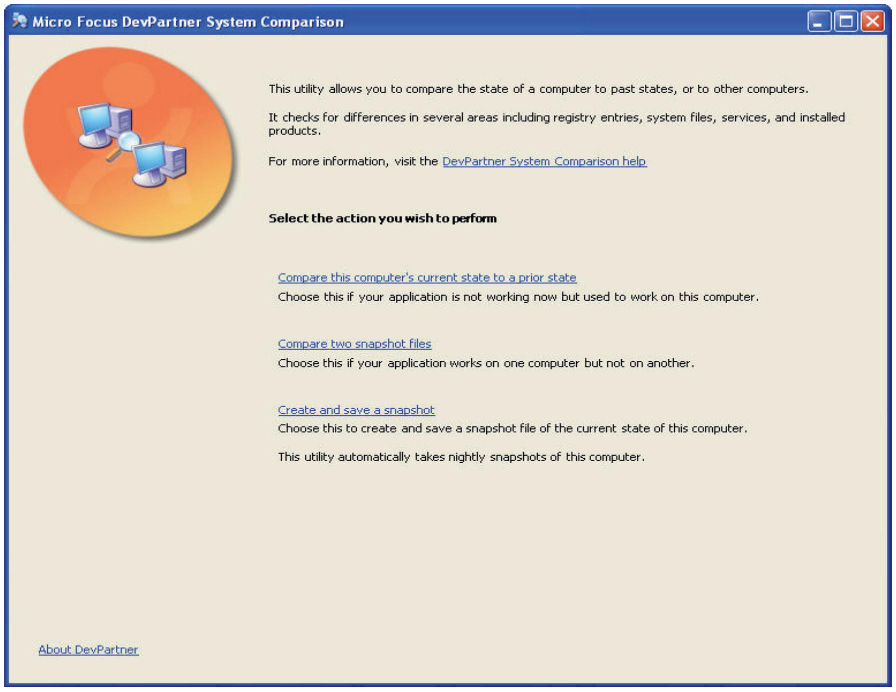
- Works on one computer but not on another
- Works differently on different computers
- No longer works on a computer on which it previously worked

To compare systems, System Comparison creates XML files, called snapshot files, that contain information about a computer system, such as its installed products, system files, drivers, and many other system characteristics. It then compares snapshot files and reports the differences between them.

Unlike other DevPartner components, System Comparison is not integrated into the Visual Studio environment. It runs as a standalone utility to minimize its impact on target systems.

System Comparison consists of:

- a service, which takes nightly snapshots of a system,
- a user interface, which enables you to take snapshots manually and to compare snapshots to find differences
- a command line interface
- a Software Development Kit (SDK). The SDK allows software developers to gather additional comparison information and to embed snapshot functionality in deployed applications.



This utility allows you to compare the state of a computer to past states, or to other computers.

It checks for differences in several areas including registry entries, system files, services, and installed products.

For more information, visit the DevPartner System Comparison help

Select the action you wish to perform

Compare this computer's current state to a prior state
Choose this if your application is not working now but used to work on this computer.

Compare two snapshot files
Choose this if your application works on one computer but not on another.

Create and save a snapshot
Choose this to create and save a snapshot file of the current state of this computer.

This utility automatically takes nightly snapshots of this computer.

Figure 8-1. The System Comparison user interface

Using System Comparison Out of the Box

The following Ready, Set, Go procedure introduces you to using System Comparison.

> To get up and running quickly, follow the steps presented in shaded boxes. If you would like more information, read the additional text following the box.

Analyzing a system with System Comparison does not require elevated system privileges. The system privileges you use to create files and work with applications on your system are sufficient for DevPartner to analyze the system.

In the following procedure, make a minor change to your computer, then compare your computer's current state with its previous state.

Ready: Consider What You Want to Compare

Before running a system comparison, understand the goal of the comparison.

The following procedure assumes:

- You have installed System Comparison.
- The System Comparison service is running and has taken a snapshot.

  When System Comparison is installed, the service is started automatically and takes its first snapshot within a few minutes of starting. The service is listed as `DevPartner Differ` in your system's Services list.
- You can compare different states of one computer.

Identifying exactly what you want to compare helps ensures that you set up the comparison appropriately. For example, your goal might be one of the following, some of which might include additional set-up steps:

- To check how installation or removal of a product impacts computer services, settings, registry keys, or files. (Checking registry key or files requires the additional set-up of modifying an XML file.)
- To determine if system changes may have caused a product to stop working on a system on which it previously worked.
- To determine the extent of the impact of product changes (for example, any impact on automated tests).
- To check that a new development system has all of the tools that were available on a previous development system.
- To determine why a product does not work, or works differently, on a certain system.
- To troubleshoot a product after it has been deployed to an end-user site.

Set: Prepare for System Comparison

Once you have decided on the goal of the comparison, you might have to perform some set-up tasks.

For this procedure, you can use the default System Comparison options. No additional set-up is required.

Some examples of situations that require set-up tasks include the following:

- If you want to compare registry keys or specific files, set-up tasks would include modifying the **`RegistrySections.xml`** or **`FileSections.xml`** files, as described on page 253 and page 254
- If you want to compare data that is not gathered by default, set-up tasks would include writing a custom plug-in, as described on page 260. Categories of data gathered by default are described in Table 8-1 on page 250.

- If you want to compare two systems, set-up tasks would include installing System Comparison on the second computer, taking a snapshot, and making that snapshot file available for comparison, as described on page page 256.

Go: Make a Change and Create a Snapshot

You are now ready to begin a system comparison. In this procedure you change your computer and compare its current state with its previous state.

To demonstrate how system differences are reported, make some changes to your computer system before creating a snapshot.

1 Navigate to the **Control Panel** > **Administrative Tools** > **Services** window, and stop or start several services that do not impact your work environment. For example, you might stop the Automatic Updates service. (Take note of the services you modify so you can restart them later.)

2 From the Start menu, select **Programs** > **Micro Focus** > **DevPartner System Comparison**.

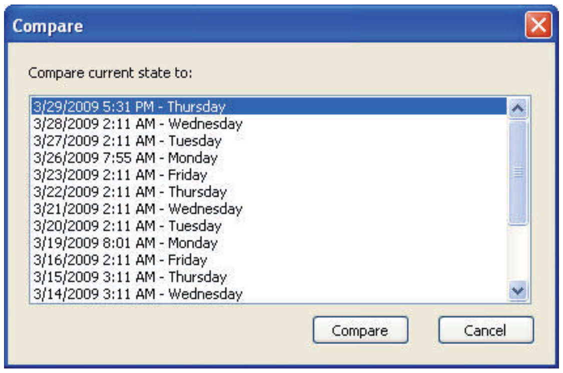
3 In the System Comparison window, click **Compare this computer's current state to a prior state**.

A list of snapshot files displays. The System Comparison service (described on page 248) automatically takes a daily snapshot of the state of the computer, and the dates and times of these files are listed.

Note: There may likely already be at least one file in the list. If there are no files listed, check that the System Comparison service is running. The service is identified as `DevPartner Differ` in the services list.

4 From the list, select the date and time of the snapshot to use as the basis of the comparison and click **Compare**.

System Comparison displays a Results window. The content of the Results window is described in Analyze Results.

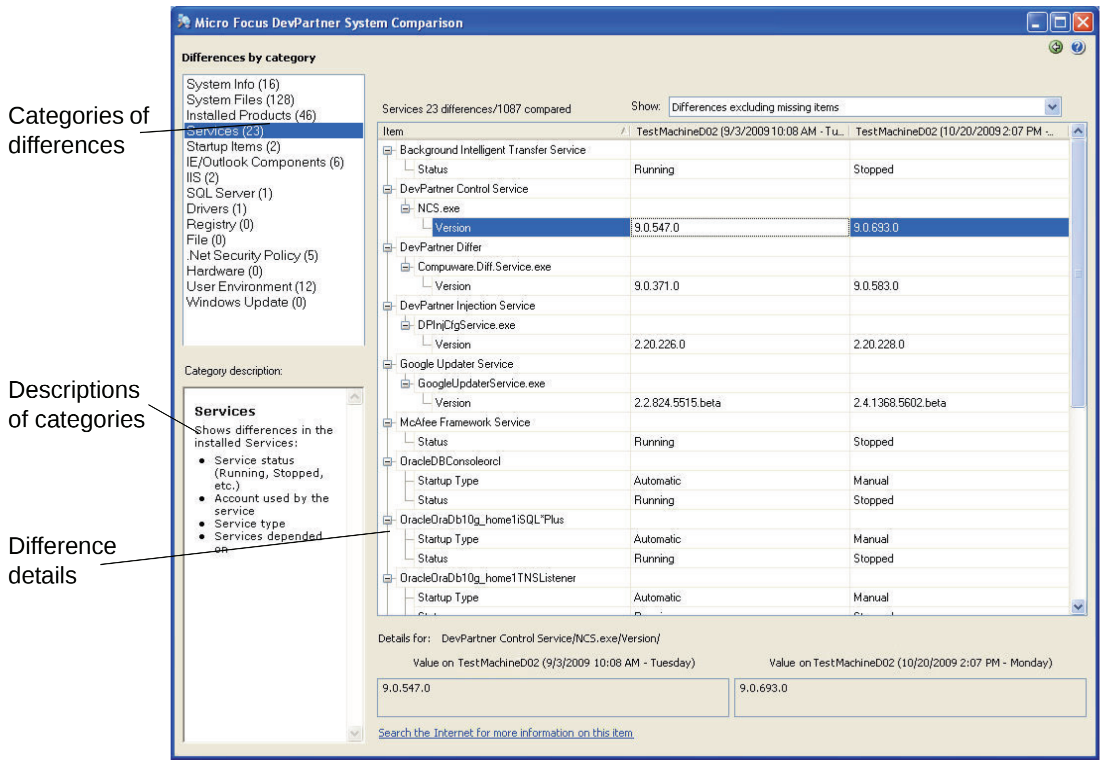


```
3/29/2009 5:31 PM - Thursday
3/28/2009 2:11 AM - Wednesday
3/27/2009 2:11 AM - Tuesday
3/26/2009 7:55 AM - Monday
3/23/2009 2:11 AM - Friday
3/22/2009 2:11 AM - Thursday
3/21/2009 2:11 AM - Wednesday
3/20/2009 2:11 AM - Tuesday
3/19/2009 8:01 AM - Monday
3/16/2009 2:11 AM - Friday
3/15/2009 3:11 AM - Thursday
3/14/2009 3:11 AM - Wednesday
```

Figure 8-2. List of Snapshot files

Analyze Results

When System Comparison compares two snapshots, it displays the differences between them and all items in a results window, as shown in Figure 8-3. The Results window from this Ready, Set, Go procedure might contain far less information than the results shown in the figure.



Micro Focus DevPartner System Comparison

Differences by category

System Info (16)
System Files (128)
Installed Products (46)
Services (23)
Startup Items (2)
IE/Outlook Components (6)
IIS (2)
SQL Server (1)
Drivers (1)
Registry (0)
File (0)
.Net Security Policy (5)
Hardware (0)
User Environment (12)
Windows Update (0)

Category description:

Services
Shows differences in the installed Services:
- Service status (Running, Stopped, etc.)
- Account used by the service
- Service type
- Services depended on

Services 23 differences/1087 compared

Show: Differences excluding missing items

Item	TestMachineD02 (9/3/2009 10:08 AM - Tu...	TestMachineD02 (10/20/2009 2:07 PM -...
Background Intelligent Transfer Service		
Status	Running	Stopped
DevPartner Control Service		
NCS.exe		
Version	9.0.547.0	9.0.693.0
DevPartner Differ		
Compuware.Diff.Service.exe		
Version	9.0.371.0	9.0.583.0
DevPartner Injection Service		
DPInjCfgService.exe		
Version	2.20.226.0	2.20.228.0
Google Updater Service		
GoogleUpdaterService.exe		
Version	2.2.824.5515.beta	2.4.1368.5602.beta
McAfee Framework Service		
Status	Running	Stopped
OracleDBConsoleorcl		
Startup Type	Automatic	Manual
Status	Running	Stopped
OracleOraDb10g_home1iSQL*Plus		
Startup Type	Automatic	Manual
Status	Running	Stopped
OracleOraDb10g_home1TNSListener		
Startup Type	Automatic	Manual

Details for: DevPartner Control Service/NCS.exe/Version/

Value on TestMachineD02 (9/3/2009 10:08 AM - Tuesday)	Value on TestMachineD02 (10/20/2009 2:07 PM - Monday)
9.0.547.0	9.0.693.0

Search the Internet for more information on this item

Figure 8-3. Results window

The upper left pane lists the categories that were compared and the number of differences found in each category. The first category with a non-zero number of differences is selected when the window initially opens.

The bottom left pane displays a description of the selected category.

The right pane displays the details of the differences found in the selected category.

1 Click on several categories to display their descriptions.

2 Click on the **Services** category to display differences in the **Difference details** pane.

In the **Difference details** pane, the name of the item appears in the first column. The second and third columns list the information from the snapshots, with the name of the computer and full timestamp of the comparison shown in the header row.

Items not in that snapshot are listed as "[missing]." Items on a computer are indicated by a check mark or the word "installed."

3 The two columns at the bottom of the details pane list the actual data from the first and second snapshots for the selected item.

4 Near the bottom of the screen is the link **Search the Internet for more information on this item**. Click the link to launch an Internet search for items related to the currently selected difference (for example, "windows system environment variables").

5 In the upper right corner of the Difference details pane, click on the Show list. Use these options to filter the differences shown.

6 When you are done reviewing differences, click the back button located in the upper right corner of the window, to return to the main System Comparison window.

The Results window shows differences and lists all items that are the same in both snapshots, depending on which option you display in the Show list.

Note that System Comparison considers version numbers when evaluating differences. It considers components with different version numbers to be different components. If a component exists in two snapshots but the version number of the component is different, the component is listed as missing.

To compare the current state with a different previous state, select a different snapshot from the **Difference details for current state compared to:** list in the results window.

When you have finished experimenting with System Comparison, remember to restart the services you stopped earlier.

This concludes the Ready, Set, Go section of this chapter. Now that you have a basic understanding of the mechanics of running a system comparison, continue reading the rest of this chapter for additional information, or refer to the System Comparison online help for task-based information.

The System Comparison Service

The System Comparison service, named `DevPartner Differ`, automatically takes a snapshot of the state of your computer at 2:10 a.m. daily if the computer is running. If the computer is powered off, it takes the snapshot five minutes after the next start-up. When you install System Comparison, it takes a snapshot a few minutes after the System Comparison service starts.

The nightly snapshot service collects snapshots for 21 nights, then begins deleting the oldest ones. You can change the number of retained snapshots by modifying the value in the System Comparison utility’s settings file, as described in “Changing the Number of Retained Snapshots” on page 249. The size of snapshot files varies depending on the amount of data collected. A typical file size is less than one megabyte.

The System Comparison service runs at minimum priority, but it does consume some system resources for several minutes while it runs. If you prefer, you can set the System Comparison service startup type to manual, but you lose automatic snapshot creation.

Changing Automatic Snapshot Settings

Both the timing of the automatic snapshot taken by the System Comparison service and the number of snapshots retained are determined by the values in the System Comparison utility settings file. The settings file (**`MicroFocus.Diff.Settings.xml`**) is located in the **`Program Files\Micro Focus\DevPartner Studio\System Comparison\bin`** folder.

For installs on 64-bit versions of Windows, the settings file is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\bin.`**

Changing the Number of Retained Snapshots

System Comparison retains 21 automatic snapshot files by default, after which the oldest files are deleted. To change the number of retained snapshot files, modify the `SnapshotsTo-Keep` key in the settings file. For example, the following key would change the number of retained snapshots to 30:

```
<add key="SnapshotsToKeep" value="30" />
```

Changing the Snapshot Hour and Minute

The System Comparison service takes an automatic snapshot of your computer at 2:10 a.m. daily. (If the computer is powered off, it takes the snapshot five minutes after the next start-up.) To change this default time, specify an alternate time in the Settings file using the `SnapshotHour0To23` and `SnapshotMinute0To59` keys. For example, the following keys would change the automatic snapshot time to 3:42 a.m.

```
<add key="SnapshotHour0To23" value="3" />
```

```
<add key="SnapshotMinute0To59" value="42" />
```

Valid settings for the hour are 0 to 23. Valid settings for the minute are 0 to 59.

You must restart the service for the new settings to take effect. If an automatic snapshot has already been taken for the day, the new setting takes effect the next day. System Comparison takes only one automatic snapshot per day.

Categories of Differences

When taking a snapshot, the System Comparison utility records the existence, version, and status of the items listed in the following table.

You can add additional categories to customize data acquisition by writing a System Comparison Plug-in, as described in “Writing a Plug-in” on page 260.

Table 8-1. Categories of Differences

Category	Differences Detected
System Info	• Operating system • .NET Framework • Global Assembly Cache • The Java Runtime • System Environment variables • File system case sensitivity
System Files	• Operating system files in **`\\System32`** and **`\\SysWOW64`** when present. • Windows File Protection Cache in **`\\System32\dllcache`** - This folder contains operating system files that are used to maintain Windows if an operating system file is damaged. If a file is damaged or missing, it is automatically replaced from this folder without any intervention. • Side-by-side assemblies in **`\\WinSxS`**
Installed Products	The products detected. If the version number is available, it is shown. The information is read from the Add/Remove Programs section of the registry.
Services	Differences in the installed services: • Service status (Running, Stopped, etc.) • Account used by the service • Service type • Services depended on
Startup Items	Startup differences. This information is read from the following: • The **`Win.ini`** file found in the Windows folder. • The following registry key: **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`** • If possible, version information is included from the program file.

Table 8-1. Categories of Differences

Category	Differences Detected
IE/Outlook Components	Internet Explorer and Outlook differences: • Active Setup shows updated or missing Outlook / Internet Explorer components extracted from the registry key **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components`** • Browser Helper objects extracted from the registry key **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects`** • MIME mappings (mapping between MIME type and which application handles the MIME) extracted from the registry key: **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\MimeFeature`** objects. • Internet Explorer extensions extracted from the registry key **`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Extensions`** • Internet settings extracted from the registry key **`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\InternetSettings`**
IIS	Differences in your Microsoft IIS installation, as available from the IIS metabase, including differences in all installed Web applications and their settings, such as: • Web server differences • SMTP server differences • FTP server differences
SQL Server	Differences in your Microsoft SQL installation: • Microsoft SQL settings from the registry. • Data about Microsoft SQL services and related services. • Settings in the `syscurconfigs` and `sysconfigures` tables in the master database for all installed instances. The System Comparison utility attempts to connect to SQL Server using integrated security. If SQL Server is not running, differences in the master database are not collected. **Note:** For master database differences to be collected, the account under which you are running must have sufficient privilege to access these two tables.
Drivers	Differences of all Drivers found: • Installed drivers • Status of drivers

Table 8-1. Categories of Differences

Category	Differences Detected

Registry	Differences in specific sections of the registry. By default, no registry sections are collected, but differences in the following registry sections can be collected:
	`HKEY_CLASSES_ROOT`
	`HKEY_LOCAL_MACHINE`
	You can customize the sections of the registry to collect by editing the **`RegistrySections.xml`** file, found in the **`Program Files\Micro Focus\DevPartner Studio\System Comparison\data`** folder.
	Note: For installs on 64-bit versions of Windows, the file is located at: **`\Program Files (x86)\Micro Focus \DevPartner Studio\System Comparison \data.`**
	You must have sufficient privilege to collect registry key data.

Files	Differences in the contents of folders and file properties from specific paths. By default, no files are included in the collection. You can customize the paths to collect by editing the **`FileSections.xml`** file, found in the **`Program Files\Micro Focus\DevPartner Studio\System Comparison\data`** folder. **Note:** For installs on 64-bit versions of Windows, the file is located at: **`\Program Files (x86)\Micro Focus`** **\DevPartner Studio`\System Comparison`** **`\data.`**
.NET Security Policy	Determines security policy differences on two separate system configurations, or security policy changes in time on the same computer. • Enterprise • Machine • User
Hardware	• System (Manufacturer, Model, Number of Processors, and System Type) • Memory (in MegaBytes) • Detailed information per processor (Description, Clock Speed, Role, and Status)
User Environment	Differences in user environments that may affect program runs. These are dependent on which user took the snapshot. • Environment Variables • Accessibility Settings • International Settings
Windows Update	Differences on the state of the Windows Update service. This information may be useful to determine if a suspected update may have changed underlying components.

Comparing Registry Keys

Registry settings are often of interest when comparing systems, but since a system might have thousands of registry keys it is useful to narrow the scope of keys to be compared. The file **`RegistrySections.xml`**, located in the data folder of your installation path (**`Program Files\Micro Focus\DevPartner Studio\System Comparison\data`** by default) specifies the sections of the registry to be compared.

For installs on 64-bit versions of Windows, the file is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\data.`**

By default, no registry keys are included in your snapshots.

Note: If using this file with the System Comparison utility's Snapshot Application Program Interface (API), this file must be in a **`\data`** folder one level above the application's executable file. For example, if the executable is in **`...\App\bin\MyApp.exe`** then this file must be **`...App\data\RegistrySections.xml`**.

You can compare registry entries in `HKEY_LOCAL_MACHINE` and `HKEY_CLASSES_ROOT`. Comparing other registry keys is not supported.

You can specify as many sections as you need.

You must have sufficient privilege to collect registry key data.

Syntax

```
<Section categoryName="XXX">YYY</Section>
```

Parameters

XXX	A category name displayed in the user interface. This attribute is optional. When not specified the registry key is used as the category name.
YYY	A registry key from which to start collecting recursively. The key does not specify the prefix `HKEY_LOCAL_MACHINE` or `HKEY_CLASSES_ROOT`. For example, to collect all of **`KEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc`** use the following: **`<Section categoryName="Microsoft RPC">SOFTWARE\Microsoft\Rpc</Section>`** To collect all of `LOCAL_MACHINE` or `CLASSES_ROOT` keys you can specify the special character '\'. For example, **`<Section categoryName="All">\</Section>`**. Be aware, though, that collecting all registry keys is time consuming. For keys of type `REG_BINARY`, only the first 20 bytes of each key are collected.

Example

The following is a sample **`RegistrySections.xml`** file.

```
<RegistrySections>
<LocalMachine>
<!-- This is an example that would collect all registry keys under RPC
-->
```

```
<Section categoryName="Microsoft RPC">SOFTWARE\Microsoft\Rpc</
Section>

</LocalMachine>
<ClassesRoot>
<!-- This is an example that would collect everything under Class-
esRoot, which would be many megabytes of data -->

<Section categoryName="All">\</Section>
<Section categoryName="Shell Extensions">*\shellex</Section>

</ClassesRoot>
</RegistrySections>
```

Comparing Specific Files

By default, differences in individual files are not collected. Comparing specific files is often of interest when comparing systems, but it is useful to narrow the scope of files to be compared. Use the file **`FileSections.xml`**, located in the data folder of your installation path (**`Program Files\Micro Focus\DevPartner Studio\System Comparison\data`** by default) to specify files to be compared.

For installs on 64-bit versions of Windows, the file is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\data.`**

Required: If using this file with the System Comparison utility's Snapshot Application Program Interface (API), this file must be in a **`\data`** folder one level above the application's executable file. For example, if the executable is in **`...\App\bin\MyApp.exe`** then this file must be **`...App\data\FileSections.xml`**.

Each category of file to be included in the comparison is specified in a separate section of **`FileSections.xml`**. You can specify as many sections as necessary.

Syntax

```
<Section [categoryName="XXX"] [filterPattern="{*,?}"]
[attributes="{yes,no}"] [programAttributes="{yes,no}"] [recurseSubDi-
rectories="{yes,no}"]>YYY</Section>
```

Parameters

`categoryName`	An optional attribute. XXX is a name that displays as a sub-category. When this attribute is not present the category name is the folder path by default.
`filterPattern`	An optional attribute. It specifies a file filter using the wildcard characters * (zero or more characters) and ? (exactly one character). When this attribute is not present, it is equivalent to `filter=”*.*”`
`attributes`	An optional XML attribute. When not present, it is equivalent to `attributes=”yes”`. If this attribute is equal to “yes”, the utility collects the following: flag read only encrypted file length modified date The Company and Product attributes are not collected, nor are boolean file attributes such as read-only or debug, unless they are set.
`programAttributes`	An optional attribute. When not present, it is equivalent to `programAttributes=”yes”`. If this attribute is equal to “yes” and the file name extension is any of “**`.exe`**”, “**`.dll`**”, “**`.ocx`**”, “**`*.cpl`**” , the utility collects the following program version information: version language Setting programAttributes to “no” is useful, for example, in a Quality Assurance environment where one wants to verify if any files have been deleted or added during the installation of a product but you expect that some files properties (like program version) to change at each installation.
`recurseSubDirectories`	An optional XML attribute. When this attribute is not present, it is equivalent to `recurseSubDirectories=”yes”` If this attribute is equal to “yes”, the utility collects file information for all folders recursively.
`YYY`	The path from which to start collecting file information recursively.

Example

The following is an example **`FileSections.xml`** file.

```
-->
- <FileSections>
- <!-- These are examples of file sections: -->

<Section categoryName="My Product">c:\somedir\somesubdir</Section>
```

```
<Section categoryName="My bat files" attributes="yes" filter-
Pattern="*.bat" programAttributes="no" recurseSubDirec-
tories="no">c:\diff</Section>

<Section categoryName="My Test Files" attributes="yes" programAt-
tributes="yes" recurseSubDirectories="yes">D:\test</Section>

</FileSections>
```

Installing Without DevPartner Studio

System Comparison installs separately from DevPartner Studio. This is useful when comparing two different computers to see why an application behaves differently on different systems. When comparing systems to find an discrepancy between them, it is important to minimize the changes made to those systems. Installing System Comparison alone, without the overhead of Visual Studio or the rest of the DevPartner features, makes it easier to focus on important differences between the systems being compared.

To install System Comparison, from the DevPartner Studio installation set-up screen select **Install DevPartner System Comparison** and follow the installation procedure.

System Comparison is included in the DevPartner license agreement, therefore using System Comparison consumes a DevPartner license. Refer to the *DevPartner Studio Installation Guide* for a detailed discussion of license issues, but note the following:

- If you have a node-locked (single-seat) license or a concurrent license, using System Comparison consumes one license while it is performing a comparison. Starting the Comparison service and taking snapshots with the service does not consume a license.
- If you are running DevPartner Studio under a 14-day evaluation period, the 14 days begins when you use the System Comparison user interface to perform a comparison. It does not begin when the Comparison service is installed, started, and takes a snapshot.

Running the Comparison Utility from the Command Line

You can automate data collection and comparison using the two command line interfaces, **`CommandLine.exe`** and **`CommandLineDiff.exe`**.

- **`MicroFocus.Diff.CommandLine.exe`** takes a snapshot of the current condition of your computer system. By default, it stores the snapshot in the last folder used to store snapshots, but you can specify an alternate folder as a parameter to the command line.

  Examples:

  ```
  C:\Program Files\Micro Focus\DevPartner Studio\System
  Comparison\bin>MicroFocus.Diff.CommandLine.exe

  C:\Program Files\Micro Focus\DevPartner Studio\System
  Comparison\bin>MicroFocus.Diff.CommandLine.exe c:\MySnaps
  ```

  For installs on 64-bit versions of Windows, the default installation folder is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\`**.

- **`MicroFocus.Diff.CommandLineDiff.exe`** compares the values in two existing snapshot files and writes the resulting differences to an output file.

  If running on WIndows 7 or Windows Vista, ensure that you have sufficient privileges to write to the output folder.

  Required parameters are `computers` (it is a placeholder) and the names of the files to be compared. Optionally, you can specify the folder in which the output file is written.

  Examples:

  ```
  C:\Program Files\Micro Focus\DevPartner Studio\System
  Comparison\bin>MicroFocus.Diff.CommandLineDiff.exe computers
  SnapFile1 SnapFile2
  ```

  ```
  C:\Program Files\Micro Focus\DevPartner Studio\System
  Comparison\bin>MicroFocus.Diff.CommandLineDiff.exe computers
  SnapFile1 SnapFile2 C:\MyResults
  ```

  The output file is an XML file that can be read programmatically to check the results of the comparison. You cannot open this output file with the System Comparison utility's user interface.

The command line programs are located in the System Comparison utility's **`\bin`** folder (**`\Program Files\Micro Focus\DevPartner Studio\System Comparison\bin`** by default).

Software Development Kit

System Comparison includes a Software Development Kit (SDK) that provides functionality for software developers, including:

- The ability to use the Snapshot Application Program Interface (API) to embed function calls in an application to trigger a snapshot after the application is deployed.

  The Snapshot API enables an application developer to control snapshot capability from within a deployed application. Should problems occur after the application is deployed, embedded API calls can trigger a snapshot that can assist with diagnosing the problem.

- The ability to write a System Comparison Plug-in to specify additional information to be gathered during a snapshot.

  The categories of information gathered by the System Comparison utility (described in Table 8-1 on page 250) are sufficient for most comparisons. If you require additional information to adequately compare systems, you can customize the System Comparison utility by writing a data acquisition plug-in.

The API and plug-in functionality are described in the following sections.

System Comparison Snapshot API

The System Comparison Snapshot API enables you to control snapshot capability from within a deployed application. When using the Snapshot API, you can specify:

- where the snapshot is stored
- what to do with messages or errors
- how progress status is to be reported
- where plug-ins are located, if custom plug-ins are used

Snapshot API information is located in two folder under the System Comparison installation folder:

- The **`System Comparison\redistributable`** folder contains the assemblies customers are licensed to include in their application's installation.

  The Snapshot API assemblies may be redistributed in accordance with the Micro Focus Software License Agreement terms and conditions. Licensing software is not required to take a snapshot. To be viewed and compared, the snapshot must be sent to a licensed DevPartner system.

- The **`\System Comparison\sdk\SnapshotAPI`** folder contains a sample application (**`SampleSnapshotAPI.cs`**) showing use of the API in an application.

  Review **`SampleSnapshotAPI.cs`** for an understanding of how the API can be used.

The Snapshot API is accessible from VB.NET, C#, and managed C++ and can be used with applications built with .NET Framework 1.1 and 2.0.

The following describes the classes and methods implemented for the Snapshot API, as illustrated in the **`SampleSnapshotAPI.cs`** application.

Note: If using the Snapshot API, your application's path must include a **`\data`** folder one level above the application's executable file. The **`RegistrySections.xml`** and **`FileSections.xml`** files must reside in the **`\data`** folder, even if those files are not being used. For example, if your executable is in **`...\App\bin\MyApp.exe`**, then **`...App\data\RegistrySections.xml`** and **`...App\data\FileSections.xml`** must exist.

Taking a Snapshot

`MicroFocus.Diff.Collector` implements the class `SnapshotAPI.` You can use the class `SnapshotAPI` to take a snapshot.

Public SnapshotAPI (ILoggable logger, IProgressStatus progressStatusInter-estedParty, String pluginsSubDirectoryName)	**Logger**: An instance of a class responsible for handling events and errors. Optionally, pass null, but implementing a logger is recommended as it eases troubleshooting. **progressStatusInterestedParty**: An instance of a class responsible for handling progress status messages. It may be important in your application to provide feedback to the user, since the snap-shot operation may be lengthy. Optionally, pass null. **pluiginsSubDirectoryName**: The name of a sub-folder of your executable's folder that contains data acquisition assemblies. If you do not have any plug-ins, you can specify an empty existing folder or pass null.

The `SnapshotAPI class` implements three methods:

public string TakeSnapshot (String snapshotDirectory)	Takes the snapshot and stores it in the specified folder.
public int GetNumberSteps ()	Provides the total number of steps the progress status object receives. You can then design your progress status accordingly. (See page 260 for information on the `ProgressStatus` inter-face.)
public void Dispose ()	The snapshot object is a disposable object.

The following example illustrates the most fundamental snapshot functionality:

```
using ( SnapshotAPI snapshoter = new SnapshotAPI( null, null, null ) )
{
string snapFile = snapshoter.TakeSnapshot ( userSnapshotDirectory );}
```

This would take a snapshot but would not be very useful in a production setting, as errors, messages, and progress would not be tracked.

Logging Messages

You can control reporting of errors and messages returned during the snapshot process using `MicroFocus.Diff.LoggableInterface`. In your application, create a logging mechanism to implement this interface to direct the messages to an appropriate output device. The sample application, for example, implements a `ConsoleLogger` class that logs messages to the console.

This interface consists of two methods:

`void Log( string message )`	Call this method to log a normal status message.
`void LogError( string message )`	Call this method to log an error message.

Reporting Progress

You can report and display the progress of the snapshot by implementing the `MicroFocus.Diff.ProgressStatus` interface. This interface consists of three methods:

`void OneStep ()`	The method to call back to notify the interested party to increment the progress display by one step.
`void MultiSteps ( int nbrSteps )`	The method to call back to notify the interested party to increment the progress display by several steps.
`void UpdateStatus (String newStatus )`	The method to call back to notify the interested party to process a new status. The newStatus string represents the new status, which would typically be displayed as part of a progress status UI element.

Writing a Plug-in

The categories of information gathered by the System Comparison utility (described in Table 8-1 on page 250) are sufficient for most comparisons. If you require additional information to adequately compare systems, you can customize the System Comparison utility by writing a data acquisition plug-in. This section defines a data acquisition plug-in, demonstrates how plug-ins work using supplied samples, and explains how you can create your own data acquisition plug-in.

Required: Existing plug-ins created with versions of DevPartner Studio earlier than 9.0 must be rebuilt before they can be used in DevPartner Studio 9.0 or later.

What is a Plug-in?

A plug-in is a .Net assembly that contains one or more types that implement the interface `MicroFocus.Diff.PluginInterface.IPluggableDataExtractor`. A plug-in defines a high level category of data to be gathered for comparisons. A plug-in extracts data and hands it to its caller in an XML format by adding XML elements to a base element in a hierarchical manner.

Plug-ins reside in **`bin/plugins`** in the product installation folder. Every .NET assembly in that folder is loaded by the Comparison service and every type that implements the interface `IPluggableDataExtractor` is instantiated and placed in the list of plug-ins to call when data is extracted.

To familiarize you with the mechanics of writing a plug-in, System Comparison includes two sample files:

- A sample plug-in, **`SamplePlugin.cs`**, which demonstrates the structure of a simple plug-in. This sample does not collect significant data, but always shows a timestamp difference in the second data point of the first sub-category. For details about the methods implemented in a plug-in, refer to the file **`IPluggableDataExtractor.cs`** in **`\SDK\Plugin`**.
- A program to exercise the sample plug-in, **`TestDriver.cs`**, which you can use to become familiar with the mechanics of the sample plug-in. You can then use it to exercise your customized plug-ins. Once your plug-ins are retrieving the information you expect, you can then place them in the System Comparison **`bin/plugins`** folder to exercise them with the System Comparison user interface or command line interface.

Plug-in Sample Step By Step Instructions

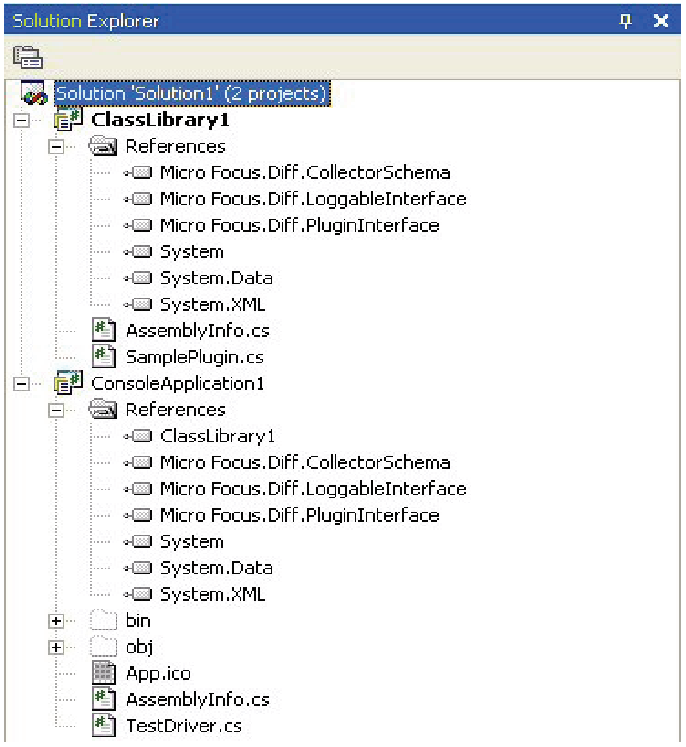
To become familiar with the use of plug-ins, use the **`TestDriver.cs`** and **`SamplePlugin.cs`** sample files. Both files are located in the **`\sdk\Plugin`** folder (**`C:\Program Files\Micro Focus\DevPartner Studio\System Comparison\sdk\Plugin`** by default).

Note: For installs on 64-bit versions of Windows, the default folder is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\sdk\Plugin.`**

Tip: Plug-ins can be created with any currently supported version of Visual Studio**.** Use the version of Visual Studio that matches the version of .NET Framework containing features you want to use.

To build and test the sample files:

1. Create a solution using Visual Studio.
2. In this solution, add two C# projects:
   - `ClassLibrary1` (type class library): This project is used to develop your plug-in.
   - `ConsoleApplication1` (type console): This project is used to debug the plug-in.



```
Solution 'Solution1' (2 projects)
  ClassLibrary1
    References
      Micro Focus.Diff.CollectorSchema
      Micro Focus.Diff.LoggableInterface
      Micro Focus.Diff.PluginInterface
      System
      System.Data
      System.XML
    AssemblyInfo.cs
    SamplePlugin.cs
  ConsoleApplication1
    References
      ClassLibrary1
      Micro Focus.Diff.CollectorSchema
      Micro Focus.Diff.LoggableInterface
      Micro Focus.Diff.PluginInterface
      System
      System.Data
      System.XML
    bin
    obj
    App.ico
    AssemblyInfo.cs
    TestDriver.cs
```

3 In the ClassLibrary1 project:

- **a** Add **`SamplePlugin.cs`** (located in **`C:\Program Files\Micro Focus\DevPartner Studio\System Comparison\sdk\Plugin`** by default).

  For installs on 64-bit versions of Windows, the default folder is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\System Comparison\.`**

- **b** Add a reference to the following assemblies (from the redistributable folder, **`C:\Program Files\Micro Focus\DevPartner Studio\System Comparison\redistributable`** by default):

  `MicroFocus.Diff.PluginInterface.dll`

  `MicroFocus.Diff.LoggabcomputerleInterface.dll`

  `MicroFocus.Diff.CollectorSchema.dll`

4 In the `ConsoleApplication1` project:

- **a** Delete **`Class1.cs`** from the solution explorer, if it exists.

- **b** Add the **`TestDriver.cs`** file to the project (located in **`C:\Program Files\Micro Focus\DevPartner Studio\System Comparison\sdk\Plugin`** by default).

- **c** Add reference to the following assemblies (from the redistributable folder):

  `MicroFocus.Diff.PluginInterface.dll`

  `MicroFocus.Diff.LoggableInterface.dll`

  `MicroFocus.Diff.CollectorSchema.dll`

- **d** Add a reference to `ClassLibrary1`.

   e Set this project as the Startup project.

5 Build and run the solution. You can run in debug mode and step through the sample to understand the basic functioning of a plug-in. An XML output file containing the sample plug-in data generates. The file is called **`pluginOutput.xml`** and is in the folder from which you ran the test driver.

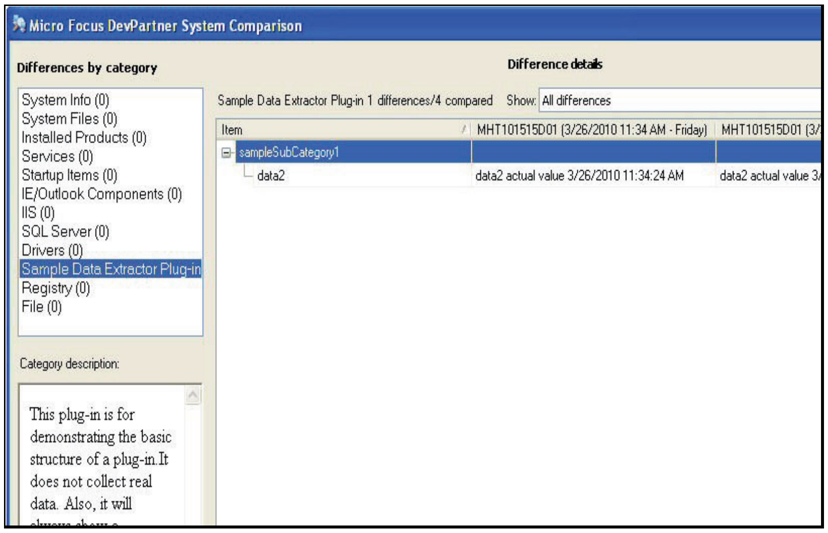
```
-<testPlugin>

- <c n="Sample Data Extractor Plug-in">
- <c n="sampleSubCategory1">
<s n="data1">data1 actual value</s>
<s n="data2">data2 actual value 4/3/2006 10:42:34 AM</s>
</c>
- <c n="sampleSubCategory2">
<s n="data1">data1 actual value</s>
<s n="data2">data2 actual value</s>
</c>
</c>

</testPlugin>
```

After exercising the sample plug-in with `TestDriver` you can use it with the System Comparison utility's user interface or command line interface:

1 Copy **`ClassLibrary1.dll`** to the plugin subfolder.

2 Use the System Comparison user interface or command line interface to take a snapshot, then take a second snapshot.

3 Compare the two snapshots. Since the `SamplePlugin` collects timestamp data, the two snapshots shows this difference.



System Info (0)
System Files (0)
Installed Products (0)
Services (0)
Startup Items (0)
IE/Outlook Components (0)
IIS (0)
SQL Server (0)
Drivers (0)
Sample Data Extractor Plug-in
Registry (0)
File (0)

Category description:

This plug-in is for demonstrating the basic structure of a plug-in.It does not collect real data. Also, it will

Sample Data Extractor Plug-in 1 differences/4 compared   Show: All differences

Item	MHT101515D01 (3/26/2010 11:34 AM - Friday)	MHT101515D01 (3/
sampleSubCategory1		
data2	data2 actual value 3/26/2010 11:34:24 AM	data2 actual value 3/

Figure 8-4. Results window for the sample plug-in

Creating and Testing Your Plug-in

Once you are familiar with the mechanics of plug-ins you can begin to design your customized plug-in to gather data that interests you.

When designing your plug-in, pay particular attention to the hierarchy of data collected. Be sure that the hierarchy is designed to provide insight into the values in which you are interested. When a non-matching value is found in a data hierarchy, the rest of the data in that hierarchy is not compared. (Refer to "Modifying a Deployed Plug-in" on page 264 for information about changing the data hierarchy in subsequent versions of a plug-in.)

You can exercise your plug-in with `TestDriver` to simplify troubleshooting. Once you are satisfied with the plug-in output, you can test it with the System Comparison command line interface:

1. Copy your plug-in to the product installation **`plugins`** folder. (You do not have to copy the **`TestDriver.exe`** file, which was used only to test your plug-in.)
2. Run the command line program (**`<product dir>\bin\MicroFocus.Diff.CommandLine.exe`**) on two computers that have differences in the area your plug-in is collecting.
3. Compare the two snapshots with the System Comparison user interface. You should see your differences.
4. Restart the System Comparison service so it includes the data specified by your plug-in when it creates snapshots.

Review the **`DifferEvent.log`** in your temporary folder (see the temp environment variable for the exact location) to troubleshoot any problems that occur. An event is logged if your plug-in is found and instantiated. Subsequent errors that occur during load, unload, or get data calls also generate events in the log.

Also, any error you log via the `ILoggable traceLogger` parameter of the `IPluggableDataExtractor.GetData` call writes to this file. See **`IPluggableExtractor.cs`**.

Modifying a Deployed Plug-in

After deploying a plug-in, you may decide to modify the data being collected. When older snapshot files are compared with snapshots created with the new version of your plug-in, the data collected does not match. The System Comparison utility identifies the mismatch as a difference, which could lead to confusion.

You can control how a mismatch is handled through use of major and minor version numbers. When the major version number of a plug-in differs between snapshots, the System Comparison utility reports that the "Plug-in schemas are incompatible." If the minor version numbers differ, the System Comparison utility identifies the status of the new data as being "unknown" in the old snapshot.

If you change a plug-in to delete data or change the hierarchy of data collected, changing the major version number is recommended. If you change the plug-in to add data, changing the minor version is generally sufficient.

To become familiar with this mechanism, you can experiment with changing the version number in the `SamplePlugin`, which is initially set to 1.0:

```
public PluginSchemaVersion PluginVersion
{
get { return new PluginSchemaVersion( 1, 0 );}
}
```

Note: If you need to replace or remove a plug-in, you first need to stop the System Comparison service and exit the System Comparison user interface to prevent the file from been locked by the operating system.

Highlights of the Plug-in Schema

To familiarize yourself with the plug-in schema, you can check any snapshot. A snapshot contains data from the plug-in. For details, refer to the file **`diff-plugin-schema.xsd`** in **`\sdk\Plugin`**. The following is an annotated sample XML fragment showing some of the elements and attributes you can use.

`<c`	Outermost Category node is for your plug-in.
`n="MyApplication"`	The name of your plug-in. This text appears in the list of categories (on left in UI).
`descrip="text"`	This text is shown in the bottom left of the UI when your category is selected.
`schema="1">`	Set this to "1". Change it when new versions of your plug-in are incompatible with prior releases.
`<c`	All nested categories are shown in the main window of the UI.
`n="MyCategory"`	This is the name shown in the main window of the UI, and used for comparison.
`missing="text"`	Optional. This is the text you want shown when this category is missing from the other snapshot. It can be version information or something simple, like "installed".
`error="text">`	Optional. If your plug-in gets an error while fetching data for this category, you can include it here and it displays in the UI.
`<s`	All data is string data.
`n="MyData"`	This is the name shown in the main window of the UI, and used for comparison.
`search="t1 t2"`	Optional. Search terms for the "search" link in the UI. These are passed to Google.
`error="text">`	Optional. If your plug-in gets an error while fetching data, you can include it here and it displays in the UI.
`Actual Data Value`	This is your data from the registry or some other setting.
`</s>`	
`</c>`	
`</c>`	

About the Redistributable Assemblies

The current version of **`MicroFocus.Diff.PluginInterface.dll`**, **`MicroFocus.Diff.LoggableInterface.dll`** and **`MicroFocus.Diff.CollectorSchema.dll`** is 1.0.0.0. Customized plug-ins continue to work with future versions of the System Comparison utility as long as the version number of these assemblies does not change. If major modifications are made to the assemblies, the version number increments and you must rebuild your plug-ins against the new assemblies.

Appendix A
DevPartner Studio Supported Project Types

This chapter contains tables listing project types supported by each DevPartner Studio feature listed above.

Supported Project Types

DevPartner Studio works in many software development environments which include Visual Studio integrated development environments, managed and unmanaged project types, and programming languages.

Table A-1 Supported Visual Studio version and language reference

Integrated Development Environment	Managed or Unmanaged	Language
Visual Studio 2012 (VS2010)	Managed	Visual Basic Visual C++ Visual C#
	Unmanaged	Visual C++
Visual Studio 2010 (VS2010)	Managed	Visual Basic Visual C++ Visual C#
	Unmanaged	Visual C++
Visual Studio 2008 (VS2008)	Managed	Visual Basic Visual C++ Visual C#
	Unmanaged	Visual C++
Visual Studio 2005 (VS2005)	Managed	Visual Basic Visual C++ Visual C# Visual J#
	Unmanaged	Visual C++

The following pages describe supported IDEs, project types, and languages for each respective DevPartner Studio feature.

Error Detection Supported Project Types

Projects build into an x86 or x64 executable. DevPartner error detection supports the following project types

Table A-2 Error Detection Support for Managed Project Types

Visual Studio Version(s)	Project Types	Supported Languages
VS2012	ATL Project CLR Console Application CLR Empty Project Makefile Project MFC Application MFC ActiveX Control MFC DLL Win32 Console Application Win32 Projec	C++
	Activity Designer Library Activity Library ASP.NET AJAX Server Control [1] ASP.NET AJAX Server Control Extender [1] ASP.NET Dynamic Data Entities Web Application ASP.NET Empty Web Application ASP.NET MVC 3 Web Application ASP.NET MVC 4 Web Application ASP.NET Server Control [1] ASP.NET Web Forms Application Console Application Excel 2010 Workbook [3] Excel 2010 Template [3] Excel 2010 Workbook [3] Import Reusable SharePoint 2010 Workflow Import SharePoint 2010 Solution Package Portable Class Library Reports Application SharePoint 2010 Project SharePoint 2010 Silverlight Web Part SharePoint 2010 Visual Web Part Silverlight Application Silverlight Business Application Silverlight Class Library Silverlight Navigation Application	C#, VB

Table A-2 Error Detection Support for Managed Project Types

Visual Studio Version(s)	Project Types	Supported Languages
VS2012 (cont.)	Syndication Service Library WCF RIA Services Class Library WCF Service Application WCF Service Library WCF Workflow Service Application Windows Forms Control Library Windows Service Word 2010 Document [3] Word 2010 Template [3] Workflow Console Application WPF Application2 WPF Browser Application WPF Custom Control Library [2] WPF User Control Library [2]	
	Class Library Empty Project	C++, C#, VB
[1] JavaScript, Asynchronous XML [2] Generated code; .NET Framework 3.0 or later [3] Not supported in Visual Studio (Standalone only)		
VS2010	ATL Project CLR Console Application CLR Empty Project Makefile Project MFC ActiveX Control MFC Application MFC DLL Win32 Console Application Win32 Project	C++
	Activity Library Activity Designer Library ASP.NET AJAX Server Control [1] ASP.NET AJAX Server Control Extender [1] ASP.NET Dynamic Data Entities Web Application ASP.NET Empty Web Application ASP.NET MVC 2 Empty Web Application ASP.NET MVC 2 Web Application ASP.NET Server Control [1] ASP.NET Web Forms Application	C#, VB

Table A-2 Error Detection Support for Managed Project Types

Visual Studio Version(s)	Project Types	Supported Languages
VS2010 (cont.)	Console Application Crystal Reports Application Empty Workflow Project [3] Sequential Workflow Console Application [3] State Machine Workflow Service Library Syndication Service Library Visual Basic SQL CLR Database Project Visual C# SQL CLR Database Project Windows Forms Application Windows Forms Control Library Windows Service WPF Application [2] WPF Browser Application [2] WPF Custom Control Library [2] WPF User Control Library [2]	
	Class Library Empty Project Test Project[3]	C++, C#, VB
[1] JavaScript, Asynchronous XML [2] As a standard VB or C# [3] Generated code; .NET Framework 3.0 or later		
VS2008	MFC Application MFC DLL Win32 Console Application Win32 Project Win32 Smart Device Project (see note)	C++
	Crystal Reports Application Test Project Windows Control Library Windows Application Windows Forms Application Windows Forms Control Library WPF Application [1] WPF User Control Library [1] WPF Custom Control Library [1] WPF Browser Application [1]	C#, VB

Table A-2 Error Detection Support for Managed Project Types

Visual Studio Version(s)	Project Types	Supported Languages
VS2008 (cont.)	Console Application Windows Service	C++, C#, VB
[1] Generated code; .NET Framework 3.0 or later [2] VS 2008 SP1 or later		
VS2005	MFC Application MFC DLL MFC Active X Control MFC ISAPI Extension DLL Win32 Console Application Win32 Project Win32 Smart Device Project (see note) Windows Forms Control Library	C++
	Calculator Starter Kit	J#
	Crystal Reports Application Windows Control Library Windows Application	C#, J#, VB
	Console Application Windows Service	C++, C#, J#, VB
MFC - Microsoft Foundation Class		

Note: Win32 Smart Device project types (Smart Device project types where the Solution Platform is set to Win32) must be running on the development computer, not an emulator, for DevPartner error detection to support them.

Hosted projects are built into an x86 or x64 DLL, and need to be hosted in an EXE if you want to test them. DevPartner error detection supports the following project types only when hosted within another executable:

Table A-3 Supported when the project is hosted within another executable

Visual Studio Version(s)	Supported Project Types When Hosted Within Another Executable	Supported Languages
VS2012	ATL Project Extended Stored Procedure DLL MFC Active X Control MFC DLL Windows Forms Control Library	C++
	LightSwitch Application (Standalone only) Web Control Library	C#, VB
	Class Library Windows Control Library	C++, C#, VB
VS2010	ATL Project Extended Stored Procedure DLL MFC Active X Control MFC DLL Windows Forms Control Library	C++
	Web Control Library	C#, VB
	Class Library Windows Control Library	C++, C#, VB
VS2008	ATL Project ATL Server Project ATL Smart Device Project Extended Stored Procedure DLL MFC Active X Control MFC DLL MFC ISAPI Extension DLL MFC Smart Device ActiveX Control MFC Smart Device Application MFC Smart Device DLL Windows Forms Control Library	C++
	Web Control Library	C#, VB
	Class Library Windows Control Library	C++, C#, VB

Visual Studio Version(s)	Supported Project Types When Hosted Within Another Executable	Supported Languages
VS2005	ATL Project ATL Server Project ATL Smart Device Project Extended Stored Procedure DLL MFC Active X Control MFC DLL MFC ISAPI Extension DLL MFC Smart Device ActiveX Control MFC Smart Device Application MFC Smart Device DLL Windows Forms Control Library	C++
	Web Control Library	C#, J#, VB
	Class Library Windows Control Library	C++, C#, J#, VB
ATL - Active Template Library MFC - Microsoft Foundation Class		

Code Review Supported Project Types

The following table lists project types supported by DevPartner Studio code review.

Table A-4 Code Review Support for Managed Project Types

Visual Studio Version(s)	Managed Project Type	Supported Languages
VS2012	Activity Designer Library Activity Library ASP.NET AJAX Server Control [1] ASP.NET AJAX Server Control Extender [1] ASP.NET Dynamic Data Entities Web Application ASP.NET Empty Web Application ASP.NET MVC3 Web Application ASP.NET MVC4 Web Application ASP.NET Server Control [1] ASP.NET Web Forms Application Class Library Console Application Empty Project Excel 2010 Template Excel 2010 Workbook Import Reusable SharePoint 2010 Workflow LightSwitch Application Portable Class Library Reports Application SharePoint 2010 Project SharePoint 2010 Silverlight Web Part SharePoint 2010 Visual Web Part Silverlight Application Silverlight Business Application Silverlight Class Library Silverlight Navigation Application Syndication Service Library WCF RIA Services Class Library WCF Service Application WCF Service Library WCF Workflow Service Application Windows Forms Application Windows Forms Control Library Windows Service Word 2010 Document Word 2010 Template Workflow Console Application	C#, VB

Table A-4 Code Review Support for Managed Project Types

Visual Studio Version(s)	Managed Project Type	Supported Languages
VS2012 (cont.)	WPF Application WPF Browser Application WPF Custom Control Library [2] WPF User Control Library [2]	
[1] As an ASP .NET Web [2] Generated code; .NET Framework 3.0 or later		
VS2010	Activity Library Activity Designer Library ASP.NET MVC 2 Empty Web Application ASP.NET MVC 2 Web Application ASP.NET Dynamic Data Entities Web Application ASP.NET Dynamic Data Linq to SQL Web Application ASP.NET Empty Web Application ASP.NET Web Application ASP.NET Web Forms Application ASP.NET Web Service ASP.NET Web Service Application ASP.NET Web Site ASP.NET AJAX Server Control [1, 4] ASP.NET AJAX Server Control Extender [1] ASP.NET Server Control [1] Class Library Console Application Crystal Reports Application Empty Project Empty Workflow Project [2] Reports Application Sequential Workflow Console [2] Sequential Workflow Library[2] Sequential Workflow Service Library State Machine Workflow Console [2] State Machine Workflow Library[2] State Machine Workflow Service Library Syndication Service Library Test Project[2] WCF Service Application WCF Workflow Service Application	C#, VB

Table A-4 Code Review Support for Managed Project Types

Visual Studio Version(s)	Managed Project Type	Supported Languages
VS2010 (cont.)	Windows Application Windows Forms Application Windows Forms Control Library Windows Service Workflow Activity Library[2] Workflow Console Application WPF Forms Application [3] WPF Browser Application [3] WPF Custom Control Library [3] WPF User Control Library [3]	
[1] As an ASP .NET Web [2] As a standard VB or C# [3] Generated code; .NET Framework 3.0 or later		
VS2008	ASP.NET Web Application ASP.NET Web Service ASP.NET Web Site ASP.NET AJAX Server Control[1] ASP.NET AJAX Server Control Extender[1] ASP.NET Server Control[1] Class Library Console Application Crystal Reports Application Mobile Web Application Test Project[2] Empty Workflow Project[2] Sequential Workflow Console [2] Sequential Workflow Library[2] State Machine Workflow Console [2] State Machine Workflow Library[2] Workflow Activity Library[2] Web Control Library Windows Application Windows Control Library Windows Service	C#, VB

Table A-4 Code Review Support for Managed Project Types

Visual Studio Version(s)	Managed Project Type	Supported Languages
VS2008 (cont.)	WPF Application [3] WPF User Control Library [3] WPF Custom Control Library [3] WPF Browser Application [3] WCF Service Application	
[1] As an ASP .NET Web [2] As a standard VB or C# [3] Generated code; .NET Framework 3.0 or later		
VS2005	ASP.NET Web Application ASP.NET Web Service ASP.NET Web Site Class Library Console Application Crystal Reports Application Mobile Web Application Web Control Library Windows Application Windows Control Library Windows Service	C#, VB

Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert Supported Project Types

The following table lists DevPartner Studio analysis supported projects.

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2012	ATL Project CLR Console Application CLR Empty Project Makefile Project MFC ActiveX Control MFC Application MFC DLL Win32 Console Application Win32 Project	C++
	Activity Library Activity Designer Library ASP.NET AJAX Server Control [1, 4] ASP.NET AJAX Server Control Extender [1] ASP.NET Dynamic Data Entities Web Application ASP.NET Empty Web Application ASP.NET Server Control [1] ASP.NET Web Application ASP.NET Web Forms Application ASP.NET Web Service Application Console Application Empty Workflow Project [3] Excel 2010 Workbook [4, 7] Excel 2010 Template [4, 7] Excel 2010 Workbook[4, 7] LightSwitch Application (Visual Basic) [8] LightSwitch Application (Visual C#) [8] Reports Application SQL Server CLR Database Project WCF Workflow Service Application WCF Service Application3 Windows Forms Application Windows Forms Control Library Windows Service	C#, VB

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2012 (cont.)	Word 2010 Document [4, 7] Word 2010 Template [4, 7] Workflow Console Application WPF Application [2] WPF Browser Application [2] WPF Custom Control Library [2] WPF User Control Library [2]	C#, VB
	Class Library Empty Project	C++, C#, VB
	Class Library Console Application Empty Project Link Library [6] Syndication Service Library WCF Service Library Windows Application [6] Windows Forms Application Windows Forms Control Library Windows Service WPF Application WPF User Control Library	COBOL for .NET (Micro Focus Studio Enterprise Edition)[5]

[1] JavaScript, Asynchronous XML

[2] Generated code; .NET Framework 3.0 or later

[3] As a standard VB or C#

[4] Coverage Analysis and Performance Analysis only

[5] Visual Studio 2008 Service Pack 2 or later

[6] Native (Error Detection, Coverage Analysis, Performance Analysis work without instrumentation)

[7] Not supported in Visual Studio (Standalone only).

[8] When hosted inside an application. Not supported in Visual Studio (Standalone only).

MFC - Microsoft Foundation Class

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2010	ATL Project CLR Console Application CLR Empty Project Makefile Project MFC ActiveX Control MFC Application MFC DLL Win32 Console Application Win32 Project	C++
	Activity Library Activity Designer Library ASP.NET AJAX Server Control[1, 4] ASP.NET AJAX Server Control Extender[1] ASP.NET MVC 2 Empty Web Application ASP.NET MVC 2 Web Application ASP.NET Dynamic Data Entities Web Application ASP.NET Dynamic Data Linq to SQL Web Application ASP.NET Empty Web Application ASP.NET Server Control[1] ASP.NET Web Application ASP.NET Web Service Application Console Application Crystal Reports Application Empty Workflow Project[3] Reports Application Sequential Workflow Console Application [3] Sequential Workflow Library[3] Sequential Workflow Service Library State Machine Workflow Console Application [3] State Machine Workflow Library[3] State Machine Workflow Service Library Syndication Service Library	C#, VB

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2010 (cont.)	Visual Basic SQL CLR Database Project Visual C# SQL CLR Database Project WCF Workflow Service Application WCF Service Application [3] WCF Service Library Windows Forms Application Windows Forms Control Library Windows Service Workflow Activity Library [3] Workflow Console Application WPF Application [2] WPF Browser Application [2] WPF Custom Control Library [2] WPF User Control Library [2]	C#, VB
	Class Library Empty Project	C++, C#, VB
	Class Library Console Application Empty Project Link Library [6] Syndication Service Library WCF Service Library Windows Application [6] Windows Forms Application Windows Forms Control Library Windows Service WPF Application WPF User Control Library	COBOL for .NET (Micro Focus Studio Enterprise Edition)[5]

[1] JavaScript, Asynchronous XML

[2] Generated code; .NET Framework 3.0 or later

[3] As a standard VB or C#

[4] Coverage Analysis and Performance Analysis only

[5] Visual Studio 2008 Service Pack 2 or later

[6] Native (Error Detection, Coverage Analysis, Performance Analysis work without instrumentation)

MFC - Microsoft Foundation Class

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2008	ATL Project ATL Server Project ATL Server Web Service ASP.NET Web Service CLR Console Application CLR Empty Project Shared Add-in MFC ActiveX Control MFC Application MFC DLL Win32 Console Application Win32 Project	C++
	ASP.NET Web Site ASP.NET AJAX Server Control [1, 4] ASP.NET AJAX Server Control Extender [1] ASP.NET Server Control [1] ASP.NET Web Application ASP.NET Web Service Console Application Crystal Reports Application Reports Application SQL Server Project WPF Application [2] WPF Browser Application [2] WPF Custom Control Library [2] WPF User Control Library [2] WCF Service Application [3] Test Project [3]	C#, VB

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	Project Type	Supported Languages
VS2008 (cont.)	Empty Workflow Project [3] Sequential Workflow Console Application [3] Sequential Workflow Library [3] State Machine Workflow Console Application [3] State Machine Workflow Library [3] Workflow Activity Library [3] Visual Studio Add-in Visual Studio Integration Package Web Control Library Windows Application Windows Control Library	C#, VB
	Class Library Windows Forms Application Windows Forms Control Library Windows Service	C++, C#, VB
	Windows Forms Windows Forms Control Library Syndication Service Library WPF Application Class Library Console Application Windows Service WCF Service Library WPF User Control Library	COBOL for .NET (Micro Focus Studio Enterprise Edition)[5]

[1] JavaScript, Asynchronous XML

[2] XAML generated code; .NET Framework 3.0 or later

[3] As a standard VB or C#

[4] Coverage Analysis and Performance Analysis only

[5] Visual Studio 2008 Service Pack 2 or later

MFC - Microsoft Foundation Class

Table A-5 Coverage Analysis, Performance Analysis, Memory Analysis, and Performance Expert supported project types

Visual Studio Version(s)	**Project Type**	**Supported Languages**
VS2005	ATL Project ATL Server Project ATL Server Web Service ASP.NET Web Service CLR Console Application CLR Empty Project Shared Add-in MFC ActiveX Control MFC Application MFC DLL Win32 Console Application Win32 Project	C++
	ASP.NET Web Application ASP.NET Web Service SQL Server Project	C#, VB
	ASP.NET Web Site Console Application Crystal Reports Visual Studio Add-in Visual Studio Integration Package Web Control Library Windows Application Windows Control Library	C#, J#, VB
	Class Library Windows Forms Application Windows Forms Control Library Windows Service	C++, C#, J#, VB
ATL - Active Template Library MFC - Microsoft Foundation Class		

Note: DevPartner Studio coverage analysis and performance analysis support VBScript and JScript in both classic ASP and client-side Web script.

Appendix B
Starting Analysis from the Command Line

This appendix contains information about the **`DPAnalysis.exe`** command line tool which works for Coverage Analysis, Memory Analysis, Performance Analysis, and Performance Expert.

Introducing DPAnalysis.exe

In addition to collecting analysis data while running your program in Visual Studio, you can use **`DPAnalysis.exe`** to collect profiling information without launching Visual Studio. **`DPAnalysis.exe`** collects application data in conjunction with option switches or by pointing to an XML configuration file.

Running DPAnalysis.exe from the Command Line

You can use **`DPAnalysis.exe`** from the command line, using switches or an XML configuration file to direct the analysis session. The following command line example launches a performance analysis session for the application **`target.exe`** and saves the session file (**`.dpprf`**) to the **`c:\output`** folder:

```
DPAnalysis.exe /perf /output c:\output\target.dpprf /p target.exe
```

Using **`DPAnalysis.exe`** from the command line, you can enable data collection and spawn a single process or service. To spawn more than one process with **`DPAnalysis.exe`**, see "Using DPAnalysis.exe with an XML Configuration File" on page 288.

`DPAnalysis.exe` does not instrument unmanaged code. To collect performance or coverage analysis data for an unmanaged application, you must first instrument the application. See "Collecting Data for Unmanaged Code" on page 119 for coverage analysis and "Collecting Data from Unmanaged Code" on page 190 for performance analysis.

Use the following syntax and switches to run the four DevPartner Studio analysis components from the command line.

```
DPAnalysis.exe [/Perf|/Cov|/Mem|/Exp] [/E|/D|/R][/O outputfilename]
[/W workingdirectory] [/PROJ_DIR][/H hostmachine] [/NOWAIT]
[/NO_UI_MSG] [/N] [/F] [/A C:\temp1;C:\temp2]
[/NO_QUANTUM /NM_METHOD_GRANULARITY /EXCLUDE_SYSTEM_DLLS
/NM_ALLOW_INLINING /NO_OLEHOOKS
/NM_TRACK_SYSTEM_OBJECTS] {/P|/S} target.exe [target arguments]
```

Analysis Type Switches

Sets the run-time analysis type. The default is performance analysis.

`/Cov[erage]`	Sets analysis type to Coverage Analysis
`/Exp[ert]`	Sets analysis type to Performance Expert
`/Mem[ory]`	Sets analysis type to Memory Analysis
`/Perf[ormance]`	Sets analysis type to Performance Analysis

Data Collection Switches

Enables or disables data collection for a given target, but does not launch the target. These switches are optional.

`/E[nable]`	Enables data collection for the specified process or service.
`/D[isable]`	Disables data collection for the specified process or service.
`/R[epeat]`	Profiling occurs any time you run the specified process until you use the `/D` switch to disable profiling.

Other Switches

These switches are optional.

`/O[utput]`	Specify the session file output folder or folder and name.
`/W[orkingdir]`	Specify the working folder for the target process or service.
`/PROJ_DIR`	Specify the folder of the DevPartner Studio project, used to locate playlists, etc.
`/H[ost]`	Specify target's host computer.

`/NOWAIT`	If you use `/NOWAIT` with multiple targets in a batch file, **`DPAnalysis.exe`** launches **`process2`** immediately after **`process1`** starts.
	For example:
	`DPAnalysis.exe /Exp /NOWAIT /P c:\temp\process1.exe`
	`DPAnalysis.exe /Exp /NOWAIT /P c:\temp\process2.exe`
	If you omit the optional `/NOWAIT` switch, **`DPAnalysis.exe`** waits until **`process1`** exits to start **`process2`** (default behavior).

`/NO_UI_MSG`	Set this switch to "true" to suppress UI error messages. The default is “false”.
`/N[ewconsole]`	Run the process in its own command window. If using **`DPAnalysis.exe`** to analyze a console application requiring keyboard input, you must use the `/NewConsole` switch to open a console window to accept the input.

`/F[orce_profiling]`	Forces coverage analysis or performance analysis profiling of unmanaged code applications that have not been instrumented with Native C/C++ Instrumentation.
`/A[ssembly_dir]`	For Coverage Analysis only. A list of paths that contain referenced DLL/PDB files. Use this switch to specify any referenced but not loaded DLL/PDB files to display in a Coverage session file. Separate multiple paths with a semicolon (;).

Quoted Paths and the /O[utput] Switch

If you use a quoted path as the parameter for the output (`/o`) switch and you do not include the file name, you must end the path in one of the following ways:

`/o “c:\test directory”`	End with a quote.
`/o “c:\test directory\.”`	End with a back slash followed by a period.
`/o “c:\test directory/”`	End with a forward slash.

Analysis Options

These switches are optional.

`/NO_QUANTUM`	Disables excluding time spent on other threads.
`/NM_METHOD_GRANULARITY`	Sets data collection granularity to method-level. Line-level is default (performance analysis only).
`/EXCLUDE_SYSTEM_DLLS`	Excludes data collection for system dlls (performance analysis only).
`/NM_ALLOW_INLINING`	Enable run-time instrumentation of inline methods.
`/NO_OLEHOOKS`	Disable collection of COM.
`/NM_TRACK_SYSTEM_OBJECTS`	Track system object allocation (memory analysis only).

Target Switch

Required. Pick only one. Identifies target to follow as either a process or service. All arguments that follow the target name or path are arguments to the target.

`/P[rocess]`	Specify a target process (followed by arguments to process).
`/S[ervice]`	Specify a target service (followed by arguments to service).
`/C[onfig]`	Specify the configuration file and path.

Using DPAnalysis.exe with an XML Configuration File

To manage analysis sessions with an XML configuration file, run **`DPAnalysis.exe`** from the command line with the `/config` switch and a properly structured XML configuration file as its target. For example:

```
DPAnalysis.exe /config c:\temp\configuration_file.xml
```

By using a configuration file, you can profile and manage multiple processes or services. The ability to profile multiple processes can be especially useful for analyzing Web applications.

Starting a session with **`DPAnalysis.exe`** launches a Session Control toolbar for each profiled process on the system where you invoked **`DPAnalysis.exe`**. Use the appropriate instance of the toolbar to execute session control actions for each process.

The following is a sample configuration file:

```
<?xml version=”1.0” ?>

<ProductConfiguration xmlns=”http://www.microfocus.com/products”>

  <RuntimeAnalysis Type=”Performance”

   MaximumSessionDuration=”1000” NoUIMsg="true" />

  <Targets RunInParallel=”true”>

    <Process CollectData=”true” Spawn=”true”

     NoWaitForCompletion=”true”>

      <AnalysisOptions NO_QUANTUM=”1” NM_METHOD_GRANULARITY=”1”

       SESSION_DIR=”c:\temp” />

      <Path>ClientApp.exe</Path>

      <Arguments>/arg1 /agr2 /arg3</Arguments>

      <WorkingDirectory>c:\temp</WorkingDirectory>

      <ExcludeImages>

        <Image>ClassLibrary1.dll</Image>

        <Image>ClassLibrary2.dll</Image>

      </ExcludeImages>

    </Process>

    <Service CollectData=”true” Start=”true”

     RestartIfRunning=”true”

     RestartAtEndOfRun=”true”>

      <AnalysisOptions NM_METHOD_GRANULARITY=”0”

       EXCLUDE_SYSTEM_DLLS=”1” />
      <Name>IISadmin</Name>

      <Host>remotemachine</Host>

    </Service>

  </Targets>

</ProductConfiguration>
```

XML Configuration File Element Reference

The following information describes the elements of an XML Configuration file.

Runtime Analysis Element

```
<RuntimeAnalysis Type = “type of analysis”
MaximumSessionDuration = “number of seconds”
NoUIMsg = "allow or suppress UI error messages" />
```

Attributes

None.

Type

Required. Possible choices are: **`Performance`**, **`Coverage`**, **`Memory`**, or **`Expert`**. These choices specify the analysis types for all targets listed.

MaximumSessionDuration

Optional. If omitted, no default used. If specified, **`DPAnalysis.exe`** limits a session run to this number of seconds. For example, if you specify: `MaximiumSessionDuration=”60”` and then begin profiling a service (with `RestartAtEndOfRun=”true”` for the service), after 60 seconds, **`DPAnalysis.exe`** stops the service and then restarts the service.

NoUIMsg

Optional. If omitted, “false” is used by default. If set to “true”, **`DPAnalysis.exe`** suppresses all UI error messages that may appear during the duration of the session. Setting this to “true” is useful when sessions are run unattended or when running a large number of consecutive tests.

Element Information

Number of occurrences	One
Parent elements	ProductConfiguration
Contents	None

Remarks

Defines the type of analysis and maximum session time.

Example

The following example shows a construction using `RuntimeAnalysis` following a `ProductConfiguration` tag. In this example, the `Type` attribute specifies a performance analysis with a maximum duration of 1000 seconds and suppression of UI error messages.

```
<?xml version=”1.0” ?>
<ProductConfiguration xmlns=”http://www.microfocus.com/products”>
<RuntimeAnalysis Type=”Performance” MaximumSessionDuration=”1000”
NoUIMsg="true" />
```

Targets Element

```
<Targets RunInParallel=”true or false”>

    ...

  </Targets>
```

Attributes

RunInParallel: Optional. Specify **true** or **false**. Defaults to **true** if omitted. If you specify more than one target, defines how the targets are run. When `RunInParallel` is **true**, **`DPAnalysis`.exe** starts the target processes and services one right after the other; multiple targets run at the same time (parallel). Otherwise, **`DPAnalysis.exe`** starts target `N + 1` only after process `N` has started and stopped; targets run one at a time (serial).

Element Information

Number of occurrences	One
Parent elements	RuntimeAnalysis
Contents	Process, Service

Remarks

Required. Begins a block of one or more `<Process>` or `<Service>` entries. Target processes and services are started in the order they are listed in the configuration file.

Example

The following example shows a construction using Targets to specify analysis of one `<Service>` and two `<Process>` elements. Note that `RunInParallel` is **true** so that, for this example, the targets would run in parallel.

```
<Targets RunInParallel=”true”>

   <Service CollectData=”true” Start=”true”>

      <AnalysisOptions NM_METHOD_GRANULARITY=”0”

         EXCLUDE_SYSTEM_DLLS=”1” />

      <Name>ServiceApp</Name>

      <Host>remotemachine</Host>
   </Service>
   <Process CollectData=”true” Spawn=”true”
         NoWaitForCompletion=”true”>
      <AnalysisOptions NO_QUANTUM=”1”
         NM_METHOD_GRANULARITY=”1”
         SESSION_DIR=”c:\MyDir” />
      <Path>ClientApp.exe</Path>
      <WorkingDirectory>c:\temp</WorkingDirectory>
   </Process>
   <Process CollectData=”true” Spawn=”true”
         NoWaitForCompletion=”true”>
      <AnalysisOptions NO_QUANTUM=”1”
         NM_METHOD_GRANULARITY=”1”
         SESSION_DIR=”c:\MyDir” />
      <Path>TestApp02.exe</Path>
      <WorkingDirectory>c:\temp</WorkingDirectory>
   </Process>
</Targets>
```

Process Element

```
<Process
CollectData = “true or false”
Spawn = “true or false”
NoWaitForCompletion = “true or false”
NewConsole = “true or false”
RepeatInjection = “true or false”
>
...
</Process>
```

Attributes

Profiling occurs any time you run the specified process until you use the `/D` switch to disable profiling.

CollectData: Optional. Specify **true** or **false.** Defaults to **true** if omitted. Specifies whether profiling are enabled for the target process.

Spawn: Optional. Specify **true** or **false**. Defaults to **true** if omitted. Specifies if **`DPAnalysis.exe`** spawns the specified target. Do not set to **true** for **`aspnet_wp.exe`** or **`w3wp.exe`**. DevPartner cannot spawn the ASP.NET worker process directly. Launch the ASP.NET worker process by opening the target Web page.

NoWaitForCompletion: Optional. Specify **true** or **false**. Defaults to **false** if omitted. The default is to wait until the process has completed. If set to **true**, causes **`DPAnalysis.exe`** to wait only until the target has started executing. **`DPAnalysis.exe`** does not wait for processes on remote computers (using the **`Host`** element). The `MaximumSessionDuration` attribute in the `RuntimeAnalysis` element overrides `NoWaitForCompletion`.

NewConsole: Optional. Specify **true** or **false**. Defaults to **false** if omitted. Causes **`DPAnalysis.exe`** to run the target in its own console window. The default is to use the same console that you typed the **`DPAnalysis.exe`** command line in. If you use **`DPAnalysis.exe`** to analyze a console application that requires keyboard input, you must use the `/NewConsole` switch to open a console window to accept the input.

RepeatInjection: Optional. Specify **true** or **false**. Defaults to **false** if omitted. Causes **`DPAnalysis.exe`** to profile the target in every time it runs until you explicitly specify **false**.

Element Information

Number of occurrences	One or more
Parent elements	Target
Contents	AnalysisOptions, Path, Arguments, WorkingDirectory, ExcludeImages

Remarks

Specifies a target executable.

Example

The following example shows a construction using `Process` and includes **`AnalysisOptions`**, **`Path`**, **`Arguments`**, and **`WorkingDirectory`** tags.

```
<Targets RunInParallel=”true”>

<Process CollectData=”true” Spawn=”true”

 NoWaitForCompletion=”true” NewConsole=”true”>

  <AnalysisOptions NO_QUANTUM=”1” NM_METHOD_GRANULARITY=”1”

   SESSION_DIR=”c:\MyDir” />

  <Path>ClientApp.exe</Path>

  <Arguments>/arg1 /agr2 /arg3</Arguments>

  <WorkingDirectory>c:\temp</WorkingDirectory>

</Process>
```

```
</Targets>
```

Analysis Options Element

Attributes that work with **`AnalysisOptions`** vary depending on the type of analysis session you run. Refer to the table at the end of this description. **`DPAnalysis.exe`** ignores attributes mismatched with the type of analysis.

```
<AnalysisOptions

SESSION_DIR = “c:\MyDir”

SESSION_FILENAME = “myfile.dpcov”

NM_METHOD_GRANULARITY = “1”

EXCLUDE_SYSTEM_DLLS = “1”

NM_ALLOW_INLINING = “1”

NO_OLEHOOKS = “1”

NM_TRACK_SYSTEM_OBJECTS = “1”

NO_QUANTUM = “1”

FORCE_PROFILING = “1”

ASSEMBLY_DIR = “C:\temp1;C:\temp2“

/>
```

Attributes

SESSION_DIR: Optional. Use with coverage analysis, memory analysis, performance analysis, and Performance Expert. Specify a folder for saving the session file generated by the profiled target. Without this attribute, the resulting session file is saved to the user's **`My Documents`** or **`Documents`** folder. If both `SESSION_DIR` and `SESSION_FILENAME` are absent, **`DPAnalysis.exe`** prompts you for the save location at the end of the session.

SESSION_FILENAME: Optional. Use with coverage analysis, memory analysis, performance analysis, and Performance Expert. Specify a session name for the session file generated for this target. Without this attribute, **`DPAnalysis.exe`** creates a unique name by combining the target's image name with a number (for example, **`iexplore1.dpprf`**). If you specify a name but no folder, the file is saved in the user's **`My Documents`** folder. If both `SESSION_FILENAME` and `SESSION_DIR` are absent, **`DPAnalysis.exe`** prompts you for the save location at the end of the session.

NM_METHOD_GRANULARITY: Optional. Use with performance analysis to set data collection granularity to method-level (line-level is default). Specify a value of **`1`** to set the attribute. Omitting the attribute disables it.

EXCLUDE_SYSTEM_DLLS: Optional. Use with performance analysis to exclude system images. Specify a value of `1` to set the attribute. Omitting the attribute disables it.

NM_ALLOW_INLINING: Optional. Use with coverage analysis and performance analysis to specify level of analysis detail. Enables run-time instrumentation of inline methods. Equivalent to the **Instrument Inline Functions** property. Specify a value of `1` to instrument inline functions. Omit the attribute to disable it.

NO_OLEHOOKS: Optional. Use with performance analysis to activate tracking of system objects. Specify a value of `1` to set the attribute. Omitting the attribute disables it.

NM_TRACK_SYSTEM_OBJECTS: Optional. Use with memory analysis to ignore system or third-party object allocations when tracking allocated objects. Specify a value of `1` to set the attribute. Omitting the attribute disables it. The default state (disabled) enables you to see memory allocations made when your application uses system or other non-profiled resources.

NO_QUANTUM: Optional. Use with performance analysis and Performance Expert to exclude time spent in threads of other running applications. Specify a value of `1` to set the attribute. Omitting the attribute disables it.

FORCE_PROFILING: Optional. Use with coverage analysis and performance analysis to force profiling of applications written without managed code or Native C/C++ Instrumentation. Specify a value of `1` to set the attribute. Omitting the attribute disables it.

ASSEMBLY_DIR: Optional. Use with coverage analysis to specify the absolute path(s) containing referenced but not loaded assemblies to be shown in the Coverage session file. Separate multiple locations with a semicolon. Omitting the attribute disables it.

Attribute	**Coverage**	**Memory**	**Performance**	**Performance Expert**
`NM_METHOD_GRANULARITY`			X	
`EXCLUDE_SYSTEM_DLLS`			X	
`NM_ALLOW_INLINING`	X		X	
`NO_OLEHOOKS`			X	
`NM_TRACK_SYSTEM_OBJECTS`		X		
`NO_QUANTUM`			X	X
`FORCE_PROFILING`	X		X	
`ASSEMBLY_DIR`	X			

Element Information

Number of occurrences	One or none per Process or Service
Parent elements	Process, Service
Contents	None

Remarks

Optional. Defines runtime attributes for the specified target process or service. Attributes correspond to coverage analysis, memory analysis, and performance analysis properties accessible from the Properties window in Visual Studio.

Example

The following example shows a construction using **`AnalysisOptions`** within a Service.

```
<Service CollectData=”true”>
  <AnalysisOptions NM_METHOD_GRANULARITY=”1”
   EXCLUDE_SYSTEM_DLLS=”1” NM_ALLOW_INLINING=”1”
   NO_OLEHOOKS=”1” ASSEMBLY_DIR=”C:\TEMP”>
</Service>
```

Path Element

```
<Path> c:\MyDir\target.exe </Path>
```

Attributes

None.

Element Information

Number of occurrences	One
Parent elements	Process
Contents	Path to the executable

Remarks

Required. Specify a fully qualified or relative path to the executable. You can specify the executable name without the path if the executable exists in the current folder.

Example

The following example shows a construction using `Path` within a `Process` element.

```
<Process CollectData=”true”>
  <Path>ClientApp.exe</Path>
</Process>
```

Arguments Element

```
<Arguments>/arg1 /arg2 /arg3</Arguments>
```

Attributes

None.

Element Information

Number of occurrences	Zero or one per Process or Service
Parent elements	Process, Service

Contents	None

Remarks

Optional. No default if omitted. Arguments to be passed to the target `process` or `service`.

Example

The following example shows a construction using `Arguments` within a `Process` element.

```
<Process CollectData=”true”>

  <Arguments>/arg1 /agr2 /arg3</Arguments>

</Process>
```

Working Directory Element

```
<WorkingDirectory> c:\MyWorkingDir </WorkingDirectory>
```

Attributes

None.

Element Information

Number of occurrences	One per Process or Service element
Parent elements	Process, Service
Contents	Path to the target folder

Remarks

Optional. No default if omitted. Set the working folder of the target process or service.

Example

The following example shows a construction using `WorkingDirectory` nested within a parent `Process` element.

```
<Process CollectData=”true”>

  <WorkingDirectory>c:\temp</WorkingDirectory>

</Process>
```

Exclude Images Element

```
<ExcludeImages>

<Image>ClassLibrary1.dll</Image>

<Image>ClassLibrary2.dll</Image>

</ExcludeImages>
```

Attributes

None

Element Information

Number of occurrences	Zero or one per process or service
Parent elements	Process, Service
Contents	Image

Remarks

Optional. No default if omitted. Provide a list of at least one image (no maximum) which, if loaded by the target process or service, is not profiled.

Example

The following example shows a construction using `ExcludeImages`. within a `Process` element. Note the `Image` elements contained within `ExcludeImages`.

```
<Process CollectData=”true”>

  <ExcludeImages>

    <Image>ClassLibrary1.dll</Image>

    <Image>ClassLibrary2.dll</Image>

  </ExcludeImages>

</Process>
```

Service Element

```
<Service>

CollectData = “true or false”

Start = “true or false”

RestartIfRunning = “true or false”

RestartAtEndOfRun = “true or false”

RepeatInjection = “true or false”

>

...

</Service>
```

Attributes

CollectData: Optional. Specify **true** or **false**. Defaults to **true** if omitted. Specifies whether profiling is enabled for the target service.

Start: Optional. Specify **true** or **false**. Defaults to **true** if omitted. Specifies if **`DPAnalysis.exe`** starts the specified target. If set to **false**, profiling is enabled for this target but it is not started; profiling begins the next time the service is started (by whatever means).

RestartIfRunning: Optional. Specify **true** or **false**. Defaults to **false** if omitted. When you set `RestartIfRunning` to **true**, **`DPAnalysis.exe`** attempts to restart the specified service if it is running on the host computer.

RestartAtEndOfRun: Optional. Specify **true** or **false**. Defaults to **false** if omitted. When you specify **true**, **`DPAnalysis.exe`** attempts to restart the service (generating a session file) at the end of the run.

RepeatInjection: Optional. Specify **true** or **false**. Defaults to **false** if omitted. Causes **`DPAnalysis.exe`** to profile the target every time it runs until you explicitly specify **false**.

Element Information

Number of occurrences	The configuration file must contain at least one Process or one Service element.
Parent elements	Targets
Contents	AnalysisOptions, Path, Arguments, Working Directory, ExcludeImages, Name, Host

Remarks

Specifies a target service.

Example

The following example shows a construction using `Service` within a `Targets` element.

```
<Targets RunInParallel=”true”>

  <Service CollectData=”true” Start=”true”

   RestartIfRunning=”true” RestartAtEndOfRun=”true”>

   <Name>ServiceApp</Name>

  </Service>

</Targets>
```

Name Element

```
<Name>MyServiceName</Name>
```

Attributes

None

Element Information

Number of occurrences	One

Parent elements	Service
Contents	Service name

Remarks

Required. The name of the service as registered with the service control manager. This is the same name you would use with a `NET START` command.

Example

The following example shows a construction using `Name` within a `Service`.

```
<Service CollectData=”true”>

  <Name>ServiceApp</Name>

</Service>
```

Host Element

```
<Host>hostmachine</Host>
```

Attributes

None.

Element Information

Number of occurrences	For each Process or Service, zero or one
Parent elements	Process, Service
Contents	Name of the host computer

Remarks

Optional. No default if omitted. Set the host computer of the target process or service.

Example

The following example shows a construction using `Host` within a `Service`. Note that the example includes the required `Name` element.

```
<Service CollectData=”true”>

  <Name>ServiceApp</Name>

  <Host>remotemachine</Host>

</Service>
```

Profiling Web Applications with the XML Config File

In general, there are three processes of interest for Web profiling: the browser; the Web server; and the ASP.NET worker process. All three entries can be contained in a single configuration file. Specify the browser and the ASP.NET worker process within <`Process`> elements;

specify the Web server within a `<Service>` element where a `<Name>` element identifies the service name. For IIS, this is `iisadmin`.

For example:

```
<?xml version=”1.0” ?>

<ProductConfiguration xmlns=”http://www.microfocus.com/products”>

  <RuntimeAnalysis Type=”Expert”/>

  <Targets>

     <Process CollectData=”true”>

       <AnalysisOptions

         SESSION_DIR=”z:\SessionFiles”/>

       <Path>aspnet_wp.exe</Path>

       <Host>remotemachine</Host>

     </Process>

     <Service CollectData=”true” Start=”true”

       RestartIfRunning=”true”

       RestartAtEndOfRun=”true”>

       <AnalysisOptions

         SESSION_DIR=”z:\SessionFiles”/>

       <Name>iisadmin</Name>

       <Host>remotemachine</Host>

     </Service>

     <Process CollectData=”true” Spawn=”true”>

       <AnalysisOptions

         SESSION_DIR=”c:\SessionFiles”/>

       <Path>iexplore.exe</Path>

       <Arguments>

         http://remotemachine/WebApplication/

         StartPage.aspx

       </Arguments>

      </Process>

  </Targets>
```

```
</ProductConfiguration>
```

The configuration file above:

- Enables data collection for the ASP.NET worker process on `remotemachine`.
- Enables data collection for **`inetinfo.exe`** (`iisadmin`) on the remote computer and restarts it so profiling can begin.
- Opens a local browser to the local computer directed at a Web page on the remote computer. This causes **`aspnet_wp.exe`** to be spawned on the remote computer and profiling for it begins.

When the browser is closed on the local computer, IIS on the remote computer is restarted on the remote computer (killing `aspnet_wp`) and session files are saved to the respective save folders. If you wish, you can use an existing mapped drive on the remote computer to save the session files to the computer where profiling was initiated, as shown by the **`z:\`** drive in the <`Process`> and <`Service`> elements in the example.

Sample Configuration Files

DevPartner Studio includes sample read-only configuration files. Use them as models to construct custom configuration files.

```
Sample.Process.Config.xml
Sample.Service.Config.xml
Sample.WebApp.Config.xml
Sample.DCOM.Config.xml
Sample.ClassicASP_IIS_High_Isolation.Config.xml
Sample.ClassicASP_IIS_Low_Isolation.Config.xml
Sample.Multi_Process.Config.xml
```

The default installation places the files in this folder:

`<install drive>:\Program Files\Micro Focus\DevPartner Studio\Analysis\SampleConfigs\`

For installs on 64-bit versions of Windows, the default location is **`\Program Files (x86)\Micro Focus\DevPartner Studio\Analysis\SampleConfigs\`**

`DPAnalysis.exe` does not instrument unmanaged code. To collect performance or coverage analysis data for an unmanaged application, you must first instrument the application. See "Collecting Data for Unmanaged Code" on page 119 for coverage analysis and "Collecting Data from Unmanaged Code" on page 190 for performance analysis.

Collecting Analysis Data from a Remote Computer

If you use **`DPAnalysis.exe`** to collect data for an application that executes on a remote computer, be aware of the following considerations:

- When using **`DPAnalysis.exe`** to run an application on a remote system, use the command line or XML configuration file to specify a folder and file name in order to save the session file.

- You can specify any folder to which you have write permission, including an existing mapped folder to the local (client) computer on which profiling was initiated.
- If you do not specify the folder and file name, the File Save dialog box appears on the remote computer. You must have physical access, a Terminal Services connection, or a Remote Desktop connection to the computer in order to use the dialog. The File Save dialog box default save location is the **`My Documents`** or **`Documents`** folder of the active user account.

Appendix C
Analysis Session Controls

This appendix contains information about session control files and the Session Control API, which can be used with DevPartner coverage analysis, memory analysis, performance analysis, and Performance Expert.

Introducing Session Control Files

Use the Session Control File options to create a set of rules and actions to control the data DevPartner collects as your application runs. DevPartner stores these rules and actions in a session control file (**`SessionControl.txt`**) in your application's solution folder.

A session control file includes data collection actions for selected methods so you can:

- Specify data collection actions at the entry to or exit from methods.
- Retain the session control file from session to session.
- Create entries in the session control file that affect coverage analysis, memory analysis, performance analysis, and Performance Expert sessions.

Creating a Session Control File Within Visual Studio

From supported releases of Visual Studio, you can create the file through the **DevPartner** > **Options** menu, as described below. Refer to "Using the Session Control API" on page 305 for information on creating a session control file with a text editor.

To create a session control file:

1. Choose **DevPartner** > **Options**.
2. In **Options**, choose **DevPartner > Analysis > Session Control File**. The first time you set session control file options, you access an empty session control (**`SessionControl.txt`**) file.
3. Click **Add**.
4. In the Module text box, choose or browse tto select the module from which you want to select methods. Use the Open dialog box to navigate to and select a module. Click **OK**.

   The module instrumentation status appears in the Add Action dialog box.

   All managed code modules display a "not instrumented" status. Only unmanaged (native) C++ modules that have been built with native C/C++ instrumentation show an "instrumented" status.

5. From the Methods list, choose a method for which you want to record data.

   If you are choosing methods from a .NET module (**`.netmodule`**), the Method List displays methods in **`namespace.classname.method`** format. DevPartner Studio supports a maximum of 512 characters for the qualified method name in the session control file. Names longer than 512 characters are ignored and no session control action occurs for that method.

6. In the **When to perform action** field, select whether you want the action to start upon entry into the selected method, or upon exit from the selected method.

7. In the **Action** field, Select the action you want to apply. Click **OK**.

8. Click **Add** in the Session Control options page and repeat the steps above to add more actions to the Session Control file.

9. Click **OK** to close and save the session control file.

If you have a solution open in Visual Studio, DevPartner saves the session control file in the solution folder.

Note: DevPartner searches for the **`SessionControl.txt`** file in the solution folder that contains the application executable you are profiling. If DevPartner does not find the file in the solution folder, it looks in the output folder where your application executable is built. If you place your **`SessionControl.txt`** file in another location, DevPartner is not able to recognize the session control commands.

Entries in the session control file affect analysis sessions in coverage analysis, memory analysis, performance analysis, and Performance Expert.

Session Control File Actions

The following are session control actions in the Actions list of the Add Action dialog box.

- **Stop recording (take final snapshot) -** Stops all data collection either at the entry or the exit of the selected method, and takes a snapshot.

- **Take snapshot -** Takes a data collection snapshot at the entry or exit of the specified method and saves it to the session control file. Data collection continues while taking the snapshot.

  Using the Snapshot actions in a session control file only lets you take a snapshot at the entrance to or exit from a method. To take a snapshot from anywhere within a method, use the Snapshot API or toolbar button.

  The snapshot actions produce a temporary objects session file in memory analysis sessions. To see memory size data for objects allocated since the last garbage collection, add a Run GC action immediately before a Snapshot action.

- **Clear all recorded data -** Deletes the data collected up to the point where the Clear action applies. Data collection then continues.

- **Start tracking (memory leak analysis) -** Starts monitoring calls in the application that free memory and reports on memory not freed while the application is running with Memory Analysis.

- **Stop tracking (memory leak analysis) -** Stops the preceeding Start tracking action for an application running with Memory Analysis.
- **Run GC (Memory analysis) -** Forces a garbage collection when running the application with Memory Analysis.

The following session control actions apply to Performance analysis and Coverage analysis.

- **Start transaction processing -** With the **Start Disabled** check box selected in the Session Control File options, this action starts recording data only for code executed specifically between this action and any subsequent **Stop transaction processing** action.
- **Stop transaction processing -** Stops data collection only for code executed specifically between this action and a preceeding **Start transaction processing** action.

  For example, if a **Start transaction processing** action is defined to start at a button click, and a **Stop transaction processing** action is defined to execute at the exit of the button click, data is recorded only for the code executed for that button click.

  You can also use this action with the **Start transaction processing** action to start data collection in one method and end it in another method.
- **Start recording (Global) -** With the **Start Disabled** check box selected in the Session Control File options, this action starts data collection on all executed threads in the application. This action is useful for starting data collection at a specific point in code execution after warming up the application.
- **Stop recording (Global) -** Stops data collection on all executed threads in the application. This action also stops any data collection started using the **Start transaction processing** action that has not been stopped with a subsequent **Stop transaction processing** action. If you start data collection when the application starts, you can use this action to stop data collection at upon entry into a method and resume data collection upon exit from the method, thereby excluding unneeded methods from data collection.
- **Transaction Start on Entry and Stop on Exit -** Starts data collection on a specified method when the method starts, and stops data collection when the method exits. Use this action to monitor specific threads in an application. This action has the same effect as setting a **Start transaction processing action** on a method to execute on entry into the method and a **Stop transaction processing action** on exit from the method.

Using the Session Control API

Call the Session Control API from anywhere in your source code to control data collection for any Visual Studio application. Using the session control text file allows DevPartner session control actions only on entry to and exit from methods.

DevPartner Session Control API functions

Clear	Clears the data collected up to this point. Data collection continues. Returns `NMStatusSuccess` if data was successfully cleared or `NMStatusFailure` if data was not cleared.
Snap	Takes a snapshot of the data being recorded. Returns `NMStatusSuccess` if the snapshot was successfully saved or `NMStatusFailure` if snapshot was not saved.
SaveNow	Takes a snapshot and stops data collection. Takes the filename for the method, if provided. Returns `NMStatusSuccess` or `NMStatusFailure`.
StartTrackingForLeakAnalysis	Starts tracking allocated objects. (Memory Leak analysis only)
StopTrackingForLeakAnalysis	Stops tracking allocated objects. (Memory Leak analysis only)
RunGC	Runs the system garbage collector. (Memory analysis)

Note: Make sure **`SaveNow`** is the last API function call used in your code. It stops data collection for the process, therefore all subsequent API calls are ignored.

The `Snap Session Control` API call produces a temporary objects session file in memory analysis sessions. In order to capture memory size data for objects allocated since the last garbage collection, insert the `RunGC` API call before the `Snap` API call.

Location of API

The files below contain the session control API functions for use with managed and unmanaged code, respectively. All are installed in the **`\Analysis`** folder of your DevPartner Studio installation.

Managed code Visual Studio applications	**`DevPartner.Analysis.SessionControl.dll`**
Unmanaged (native) code C/C++ or C++ applications	**`NmTxApi.h`**

Using the Session Control APIs with Managed Applications

To use the session control API functions in managed code Visual Studio applications, you must reference **`DevPartner.Analysis.SessionControl.dll`** in your project.

This gives you access to the session control APIs in the DevPartner namespace. You can insert calls to the API at appropriate points in your code using the syntax shown below.

```
Clear
```

```
DevPartner.Analysis.SessionControl.Clear()
```

```
Snap
```

```
DevPartner.Analysis.SessionControl.Snap(<your session file
name>.dpxxx)
```

Where dpxxx is the extension for your analysis type: dpcov, dpmem, dpprf, or dppxp.

```
SaveNow
```

```
DevPartner.Analysis.SessionControl.SaveNow(<your session file
name>.dpxxx)
```

Where dpxxx is the extension for your analysis type: dpcov, dpmem, dpprf, or dppxp.

```
StartTrackingForLeakAnalysis
```

```
DevPartner.Analysis.SessionControl.StartTrackingForLeakAnalysis()
```

```
StopTrackingForLeakAnalysis
```

```
DevPartner.Analysis.SessionControl.StopTrackingForLeakAnalysis()
```

```
RunGC
```

```
DevPartner.Analysis.SessionControl.RunGC()
```

Valid input for the **`Snap`** and **`SaveNow`** API functions includes:

- A file name
- A fully qualified path to a folder, terminated with a "\" (backslash)
- A fully qualified path including a file name
- Nothing (Null)

For information on how DevPartner treats the file and path information, see "Saving Files through the Session Control API" on page 308.

If You Get a Security Exception

If you get a security exception when using the session control APIs to profile a managed code application, it means that your security policy is preventing normal DevPartner instrumentation of your code at runtime. To remedy this, you must enable secure profiling.

Set the following global environment variable:

```
NM_NO_FAST_INSTR=1
```

Retry profiling the application.

By default, assemblies need to have the **`SkipVerification`** permission in order to be profiled. If you remove this permission from the permission set of the policy under which the code executes, or add imperative security declarations to the assembly that cause this permission to be revoked, you can not profile it. The solution described above allows you to work around this issue, although it does exact a slight performance penalty. If you choose not to implement the solution described above, you can also enable profiling of such assemblies with DevPartner Studio by either changing the policy for the assembly using the .NET Framework Configuration tool MMC snap-in, or by temporarily removing any imperative security declarations in the assembly.

See the *.NET Framework Developers Guide* in the Visual Studio on-line help for more information on security policy in Visual Studio.

Using the Session Control APIs with Unmanaged Applications

You can use the Session Control API to control coverage analysis and performance analysis sessions for unmanaged C/C++.

Unmanaged (Native) C/C++ Projects

Before you can collect coverage data for your native C/C++ application, you must rebuild your solution (or native C/C++ projects) with Native C/C++ Instrumentation.

To use the Session Control API functions in native C/C++:

1. Include **`NmTxApi.h`** in a file to which you want to add Session Control API calls. Add **`TxInterf.lib`** to the link library list.
2. Insert calls to the Session Control API functions at appropriate points in your code. See "Session Control API Syntax for Unmanaged Projects" on page 308.
3. Rebuild the solution or single native C/C++ projects with Native C/C++ Instrumentation.

Unmanaged (Native) C++ Projects

Before you can collect coverage data for your native C++ application, you must rebuild your project with instrumentation in Visual Studio.

Session Control API Syntax for Unmanaged Projects

Refer to the information below for Session Control API syntax for unmanaged projects.

`Clear`	`Clear()`
`Snap`	`Snap("<your session file name>.dpxxx")` Where `dpxxx` is the extension for your analysis type: **`.dpcov`**, or **`.dpprf.`**
`SaveNow`	`SaveNow("<your session file name>.dpxxx")` Where `dpxxx` is the extension for your analysis type: **`.dpcov`**, or **`.dpprf.`**

Saving Files through the Session Control API

When you use the Session Control API to take data snapshots or create final session files, you can specify the session file name and folder in the API call.

File names and folders specified in Session Control API calls override file names and folders specified by other means, for example, the `/output` switch on the command line or the `SESSION_FILENAME` or `SESSION_DIR` attributes in the XML configuration file.

- If you specify a file name and folder in the session control `Snap` or `SaveNow` API call, DevPartner saves the file. Any file with the same name in the folder is overwritten.

- If specifying only a folder, DevPartner saves the session under a unique file name based on the name of the target process. DevPartner automatically increments the file name to avoid overwriting existing files.
- If specifying only a file name, DevPartner saves the session under the specified name and determines the destination folder by the means you used to start the application. If the application is started from Visual Studio, the file is saved to the current project's solution folder. If you started the application from the command line with **`DPAnalysis.exe`**, the file is saved to the **`Documents`** folder (**`My Documents`** folder in Windows XP) of the active user account. If a file with the same name exists in the folder, it is overwritten.
- If not specifying a file name nor a folder, DevPartner saves using a unique file name and determines the destination folder by the means you used to start the application, as above. DevPartner automatically increments the file name to avoid overwriting files.

Note: If your project does not have an output folder, for example, a Visual Studio 2005 Web site project, DevPartner saves the files to the project folder.

Note the following when specifying paths:

- DevPartner evaluates path information based on the current working folder of the filing process. In some cases, the working folder can change as the application executes.
- To ensure that you can easily locate your session files, it is a good practice to specify the complete path.
- On the local computer, DevPartner creates the complete path if it does not already exist. If you are collecting data on a remote computer, you must specify an existing folder.
- If you intend to specify a path, but no file name, be sure to terminate the path with a “\” (backslash). DevPartner treats characters following the final backslash as a file name.
- If the path contains invalid data, DevPartner saves the file as if no folder was specified.

Interactions and Precedence

File names and folders specified in Session Control API calls override file names and folders specified by any other means.

Recommendation: Set the file name and folder in either the API call or the command line, but not both.

For example: If you specify only a folder (or a file name) in the Session Control API, but specify a file name (or a folder) in the **`DPAnalysis.exe`** command line or in the XML configuration file, DevPartner combines the information to name and save the file. In this example, if you intended to let DevPartner create unique file names, you would have defeated your purpose.

Recommendation: To simplify file management, specify both snapshots and the final session file with API calls.

For example: If you do not specify a final snapshot (**`SaveNow`**) through the Session Control API, DevPartner takes a final snapshot when the process terminates. If you started the application with **`DPAnalysis.exe`**, DevPartner saves the final session file according to the options specified on the command line or in the XML configuration file. If you started the application from Visual Studio, DevPartner displays the unsaved session data.

Appendix D
Exporting Analysis Data to XML

This appendix contains information about exporting analysis data to XML, which can be used with DevPartner coverage analysis, performance analysis, and Performance Expert.

Introducing DevPartner Data Export

DevPartner allows you to export saved session files from coverage analysis, performance analysis, and Performance Expert data to XML. You can export the XML data from Visual Studio or from the command line.

You can analyze the exported XML data using your own or third-party software. For example:

- Use **Export DevPartner Data** on a development build server or QA server where unit tests, functional tests, or regression tests are staged. Analyze the exported XML data to monitor daily progress.
- Use **Export DevPartner Data** to collect data for longer-term analysis. You can accumulate the XML data in a database or data warehouse in order to:
  - Integrate the data with development and QA methodologies, tools and infrastructure
  - Run custom analytics on the data
  - Archive the data for historical or auditing purposes

Exporting Analysis Data to XML

From within Visual Studio, you can export saved Coverage Analysis (**`*.dpcov`**), Coverage Analysis merge (**`*.dpmrg`**), Performance Analysis (**`*.dpprf`**), and Performance Expert (**`*.dppxp`**) data to XML format.

To export to XML in Visual Studio:

1. Open a saved session file (see above).
2. Choose **File** > **Export DevPartner Data.**

By default, DevPartner saves the XML file in the folder where the session file is saved and appends an **`.xml`** extension to the saved session file name. For example, **`Chart1.dpcov.xml`**.

The file **`DevPartnerPerformanceCoverage`***`xx`***`.xsd`** defines the XML Schema that DevPartner uses to export coverage analysis and performance analysis data. The file **`DevPartnerPerformanceExpert`***`xx`***`.xsd`** defines the XML Schema that DevPartner uses to export Performance Expert data. Both schemas are located in **`C:\Program Files\Micro Focus\DevPartner Studio\Analysis`**.

For installs on 64-bit versions of Windows, DevPartner Studio is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\Analysis\`**.

Exporting Analysis Data to XML from the Command Line

As an alternative to using Visual Studio, you can use **`DevPartner.Analysis.DataExport.exe`** from a command line to export coverage analysis, coverage analysis merge, performance analysis, and Performance Expert data to XML.

The utility is located in **`C:\Program Files\Micro Focus\DevPartner Studio\Analysis.`**

For installs on 64-bit versions of Windows, DevPartner Studio is located at: **`\Program Files (x86)\Micro Focus\DevPartner Studio\Analysis`**.

Usage

```
DevPartner.Analysis.DataExport.exe [ sessionfilename | pathtodirectory
] { options }
```

Options

`/out[put]=<String>`	Specify the local or remote output folder for exported XML files. Creates the folder if it does not exist.
`/r[ecurse]`	Search subfolders for DevPartner session files.
`/f[ilename]=<String>`	Specify the name of the XML output file. Appends .xml to the name specified.
`/showAll`	Shows all performance and coverage session file data available in a performance or coverage session file. For example, if you export a performance session file with this option, the resulting XML file contains both performance and coverage data fields. This option is not available for Performance Expert session files.
`/w[ait]`	Wait for input before closing console window.
`/nologo`	Do not display the logo or copyright notice.
`/help or /?`	Display help in the console window.
`/summary`	Export Performance Expert summary data which includes a default maximum of the top ten callpaths and the top ten methods that use the most CPU resources. Use the **`/maxpaths`** and **`/maxmethods`** options to override the maximums. The summary data displays by default.
`/method`	Export Performance Expert method data.

`/calltree`	Export Performance Expert call tree data.
`/maxpaths=<integer>`	Used only with Performance Expert. Exports the specified number of the top call paths that use the most CPU resources.
`/maxmethods=<integer>`	Used only with Performance Expert. Exports the specified number of the top methods that use the most CPU resources.

You can use an equal sign, a colon, or a space to separate an option from the value or values you specify.

DevPartner.Analysis.Export.exe Usage Examples

The following examples show some of the ways you can use **`DevPartner.Analysis.DataExport.exe`**.

Example 1: Export a coverage analysis session file to an XML file in the same folder.

```
DevPartner.Analysis.DataExport.exe
“c:\WindowsApplication1\WindowsApplication1.dpcov”
```

Output is saved to:

`c:\windowsApplication1\WindowsApplication1.dpcov.xml`

Example 2: Export a performance analysis session file saved in one location to another folder.

```
DevPartner.Analysis.DataExport.exe
“c:\WindowsApplication1\WindowsApplication1.dpprf”
/output=”c:\temp”
```

Output is saved to:

`c:\temp\WindowsApplication1.dpprf.xml`

Example 3: Export multiple Performance Expert session files saved in the same folder.

`This example assumes two Performance Expert session files saved in the same folder:` **`WindowsApplication1.dppxp`** `and` **`WindowsApplication2.dppxp.`**

```
DevPartner.Analysis.DataExport.exe “c:\WindowsApplication1\*.dppxp”
```

Output is saved to:

`c:\WindowsApplication1\WindowsApplication1.dppxp.xml`
`c:\WindowsApplication1\WindowsApplication2.dppxp.xml`

Example 4: Export multiple Coverage Analysis, Performance Analysis, and Performance Expert session files saved in the same folder.

This example assumes three session files saved in the same folder:

`WindowsApplication1.dpprf; WindowsApplication2.dpcov; and WindowsApplication3.dppxp`

```
DevPartner.Analysis.DataExport.exe “c:\WindowsApplication1”
```

Output is saved to these three files:

```
c:\WindowsApplication1\WindowsApplication1.dpprf.xml
c:\WindowsApplication1\WindowsApplication2.dpcov.xml
c:\WindowsApplication1\WindowsApplication3.dppxp.xml
```

Example 5: Export a Performance Expert summary and change the default output from the top ten methods to the top twenty methods that use the most CPU resources.

```
DevPartner.Analysis.DataExport.exe
“c:WindowsApplication1WindowsApplication1.dppxp”
/summary /maxmethods=20
```

Output is saved to:

```
c:WindowsApplication1WindowsApplication1.dppxp.xml
```

Index