DevPartnerStudio Quick Reference

Print out all or portions of this document and keep it handy for quick reference (use a color printer when available).

DevPartner Features

Use the links in the left column in the following table to locate reference information about DevPartner features.

To solve this problem	Use this DevPartner feature
Detect programming problems and naming inconsistencies	Code Review
Diagnose run-time errors in the source code	Error Detection
Locate performance bottlenecks in the application	Coverage, Memory, and Performance Analysis
Ensure code base stability throughout development and testing phases	Coverage Analysis Session Data
Determine memory allocation in an application and get feedback to reduce memory consumption	Memory Analysis

More Information

Refer to the DevPartner online help or to the *Understanding DevPartner* manual for more information.

Common Elements

The DevPartner software provides these common elements, regardless of feature.

- DevPartner Toolbar
- DevPartner Menu
- DevPartner File Extensions
- Command Line Instrumentation Options

DevPartner Menu and Toolbar

Accessed from the DevPartner menu or toolbar in Visual Studio.

Note: Menu items and icons vary slightly in Visual Studio 6.0.

Choose this menu or toolbar item	To
Error detection	Perform run-time error detection using BoundsChecker technology
Coverage Analysis	Perform run-time code coverage analysis
Error detection and Coverage Analysis	Perform run-time error detection with code coverage analysis
Performance Analysis	Execute run-time performance analysis
Memory Analysis	Execute run-time memory analysis
Performance Expert	Execute run-time analysis with Performance Expert
Perform Code Review	Perform static code analysis
Manage Code Review Rules	Access code review rules management
Error Detection Rules	Access error detection rules management, used to filter or suppress detected errors
Native C/C++ Instrumentation	Perform compile-time instrumentation for: Error detection, Error detection with coverage, Performance or coverage analysis
Native C/C++ Instrumentation Manager	Access the Instrumentation Manager
Correlate	Correlate performance or coverage files
Merge Coverage Files	Merge coverage analysis sessions
Submit TrackRecord defect	Submit TrackRecord defect See **Note**
Options	Access DevPartner options Choices include: Analysis, Code review, Error detection

Note: The Submit TrackRecord defect toolbar button is only available when TrackRecord is installed.

DevPartner File Extensions

File extensions for session files.

Run this DevPartner feature	To create this session file (extension)
Code review	.dpmdb
Code coverage	.dpcov
Code coverage merge files	.dpmrg
Error detection	.dpbcl
Memory analysis	.dpmem
Performance analysis	.dpprf
Performance Expert	.dppxp

Command Line Instrumentation Options

NMCL Options

The following table lists the NMCL options that you can use to instrument your unmanaged (native) Visual C++ code from the command line. Use NMCL.EXE only to compile unmanaged Visual C++ code with DevPartner performance and coverage or error detection instrumentation. NMCL is not used with managed code, which DevPartner instruments as it is passed to the common language runtime during execution.

Note All NMCL options must begin with a forward slash (shown in the following list) or hyphen, followed by the letters NM. For example: /NMoption or –NMoption.

Use...	To...
/NMbcpath:bc-path	Specify the directory location of bcinterf.lib if you do not have the directory that contains NMCL on your path.
/NMclpath:cl-path	Specify the directory location of cl.exe. You can use this option to bypass the installed location of DEVENV, or if DEVENV is not installed.
/NMhelp or /?	Display help text
/NMignore:source-file or /NMignore:source-file:method source-file	Specify a source file or a method in a source file that should not be instrumented
/NMlog:log-file	Specify a log file for NMCL messages (default: stdout)
/NMnogm	Ignore the CL /Gm (minimal rebuild) option if it appears on the command line. You can use this option to avoid a known conflict between the NMAKE /A and CL /Gm options.
/NMonly:source-file	Specify a single source file that should be instrumented
/NMopt:option-file or /NM@option-file	Specify an option file (an ASCII file containing individual command-line options, each on a separate line.
/NMpass	Specify pass-through mode, which instructs NMCL to call CL without intervention. In this case, no instrumentation takes place.
/NMstoponerror	Stop NMCL if an error occurs during instrumentation. If this option is not specified, the default behavior is to fall back to a standard CL compile.
/NMbcOn	Use DevPartner Error Detection instrumentation. This is the default setting.
/NMtxOn	Specifies instrumentation for performance and coverage analysis.
/NMtxInlines	Instruments methods that are marked as inlineable if inline optimizations are enabled (using the /O1, /O2, /Ob1, or /Ob2 option)
/NMtxNoLines	Instruct DevPartner not to collect line information. When you use this option, DevPartner does not display any line data in the Source tab. You can also use this to improve the time required to instrument and run your application.
/NMtxpath:tx-path	Specify the directory location of the performance and coverage analysis library files if you do not have the directory that contains NMCL on your path.

Note: When using NMCL, add the directory containing these utilities to your path. For example, if you installed the product into the default directory, add the following directory to your path:

C:\Program Files\Common Files\Compuware\NMShared

NMLINK Options

The following table lists the NMLINK options that you can use to link your unmanaged (native code) Visual C++ application to DevPartner.

Note: All NMLINK options must begin with a forward slash (shown in the following list) or hyphen, followed by the letters NM. For example: /NMoption or –NMoption.

Use...	To...
/NMbcOn	Use DevPartner Error Detection instrumentation. This is the default setting.
/NMbcpath:bc-path	Specify the directory location of bcinterf.lib if you do not have the directory that contains NMCL on your path.
/NMhelp or /?	Display help text
/NMlinkpath:link-path	Specify the directory location of LINK.EXE. You can use this option to bypass the installed location of DEVENV, or if DEVENV is not installed.
/NMpass	Specify pass-through mode, which instructs NMLINK to call LINK without intervention.
/NMtxOn	Specifies instrumentation for coverage and performance analysis.
/NMtxpath:tx-path	Specify the directory location of the performance and coverage analysis library files if you do not have the directory that contains NMCL on your path.

Note: When using NMCL and NMLINK, add the directory containing these utilities to your path. For example, if you installed the product into the default directory, add the following directory to your path:

C:\Program Files\Common Files\Compuware\NMShared

Code Review

Command Shortcuts for Rule Manager

Use the following keyboard shortcuts to enter Rule Manager commands:

Command	Action
Ctrl+A	Rule > Select All Rules
Ctrl+C	Rule > Copy Selected Rules
Ctrl+N	Rule > New Rule
Ctrl+O	File > Open Rule Set
Ctrl+P	File > Print
Ctrl+V	Rule > Paste Rules
F5	View > Refresh

Command-line Switches Used in CRBatch

CRBatch.exe /<switch>

Switch	Function
/f configuration file/file name	Informs CRBatch what configuration file to use when reviewing a solution or project This switch is mandatory.
/v or /verbose	Instructs CRBatch to report errors in a message box, and to set the exit code used by batch procedures Although this switch is optional, it is useful if you want to physically debug configuration files.
/vs "7.1" or /vs “8.0”	Indicates the Visual Studio environment where the batch review will be executed; choices include 7.1 or 8.0. It is recommended that you use this switch, most importantly if you have more than one version of Visual Studio on your system. If you do not include this switch, DevPartner will default to the latest version.

Command-line Switches Used in CRExport

CRExport.exe /<switch>

Switch	Function
/?	**Help** — Displays this list of the available command line interface arguments.
/f *sessionfile*	**Fully-qualified session file path and name** — Identifies the session database to use for this export. (Mandatory)
/e *xml exportfile*	**Fully-qualified export file path and name** — Identifies the XML file to receive the exported data. (Mandatory)
/a	**Export all session data** — Exports all data for the specified session, including the outbound methods for call graph data. Inbound methods are not exported.
/a i	**Export all session data with inbound methods** — Exports all data for the specified session, including inbound and outbound methods for call graph data.
/p	**Export problems data** — Exports the problems data for the specified session.
/m	**Export metrics data** — Exports the metrics data for the specified session.
/n	**Export naming analysis data** — Exports the naming analysis data for the specified session.
/s	**Export code size data** — Exports the code size data for the specified session.
/c	**Export call graph data** — Exports the outbound, or called, methods in the call graph data for the specified session.
/c i	**Export call graph data with inbound methods** — Exports the call graph data, including inbound and outbound methods, for the specified session.

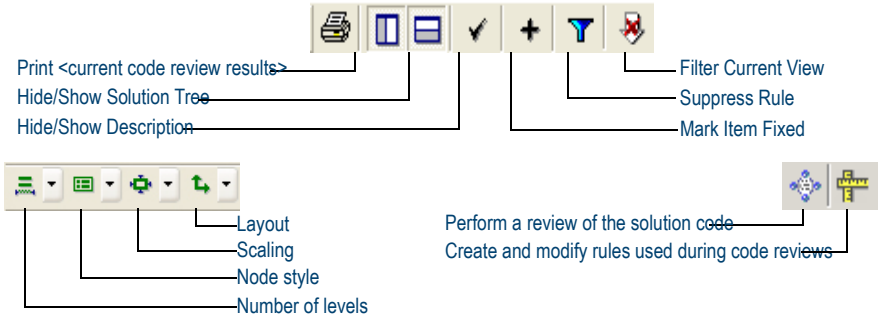
Code Review Default Options (General Node)

Category	Settings
Projects to be reviewed	All projects selected (C# and VB.NET projects only)
Rule set	All Rules
Naming analysis to use	Naming Guidelines (see below)
Collect metrics	On
Collect call graph data	On
Always generate a batch file	On
Always save review results	On
Prompt for session file name	Off

Naming Guidelines

Description	Default
What to analyze	All public or protected identifiers
Choose dictionary	American English
Include naming analysis for	All identifiers selected
Company name	
Technology name	

Code Review Toolbars



Code Review Summaries

Summary of Problems *

Type	Problems		Severity			
Names	Total	Fixed	High	Medium	Low	Warning
COM Interop	1	0	0	0	0	1
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	21	0	0	1	20	0
Garbage Collection	0	0	0	0	0	0
Internationalization	12	0	12	0	0	0
Language	0	0	0	0	0	0
Logic	2	0	0	0	0	2
Maintainability	13	0	0	2	3	8
Performance	1	0	1	0	0	0
Portability	0	0	0	0	0	0
Project & Solution Properties	0	0	0	0	0	0
Reliability	0	0	0	0	0	0
Security	3	0	3	0	0	0
Standards	0	0	0	0	0	0
System	0	0	0	0	0	0
Usability	0	0	0	0	0	0
User-Defined Rule	0	0	0	0	0	0
Versioning	0	0	0	0	0	0
Windows API	0	0	0	0	0	0
Totals	**53**	**0**	**16**	**3**	**23**	**11**

* Summaries include all rule violations. Your filter settings do not apply.

Summary of Counts

Summary Type	Count
Review Time (in minutes)	1,212
Total Lines (including blank lines)	2,183
Code Only Lines	1,162
Comment Only Lines	270
Code with Comments	0
Rule Comparisons Made	468,267
Total Lines Checked	2,183

Review Settings

Review Settings	Setting Value
Solution	SpeedBump.Net2003
Solution Path	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.sln
Session File	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.DPMDB
Batch Command Execution File	C:\p4_MHT-NMSource1666_MHT101515D01

Project List

Project Name	Compile Errors	Reviewed	Project Path
Driver2003	False	True	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\Driver\Driver2003.csproj
CSharp2003	False	True	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\CSharp\CSharp2003.csproj
VB2003	False	True	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\VB\VB2003.vbproj

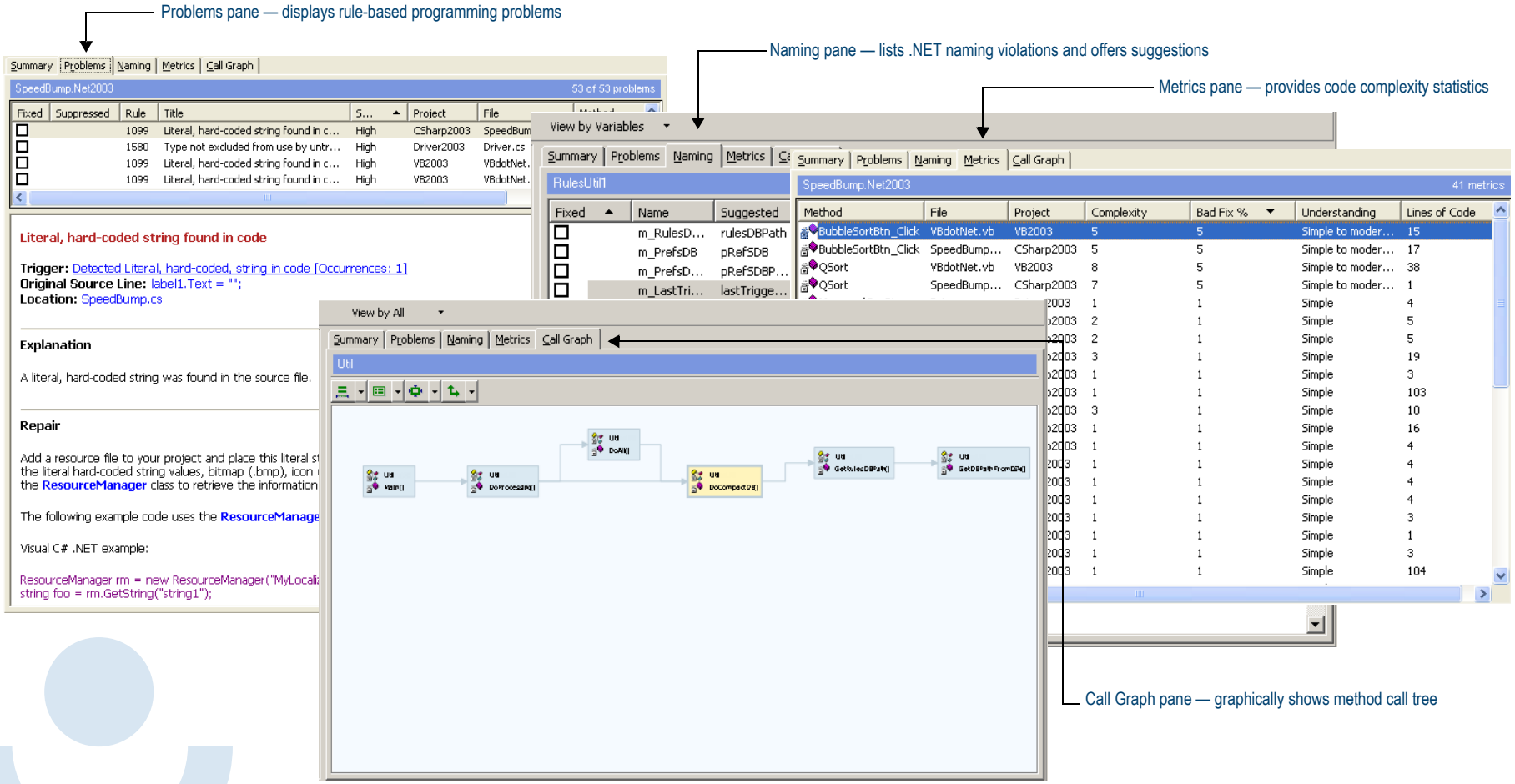
Metrics Analysis	True
Naming Analysis	Naming Guidelines
Dictionary Name	American English

Summary of Call Graph Data

Summary Type	Count
Total Methods Graphed	24
Total Methods Uncalled	0

Technology Name	not supplied
Call Graph Analysis	True
Ignore compile errors	False
Exclude rules that require a build	False
Always generate a batch file	True

Code Review Results Panes



Coverage, Memory, and Performance Analysis

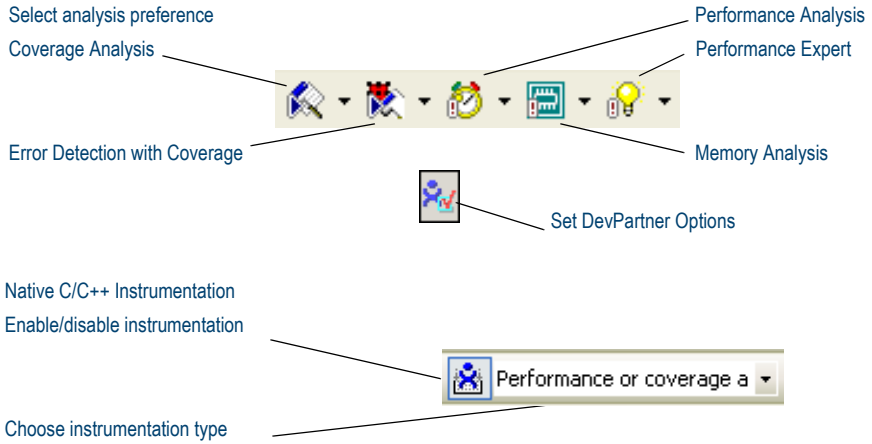
Determine application test coverage, analyze an application's use of memory, and profile application performance.

General and Data Collection Properties

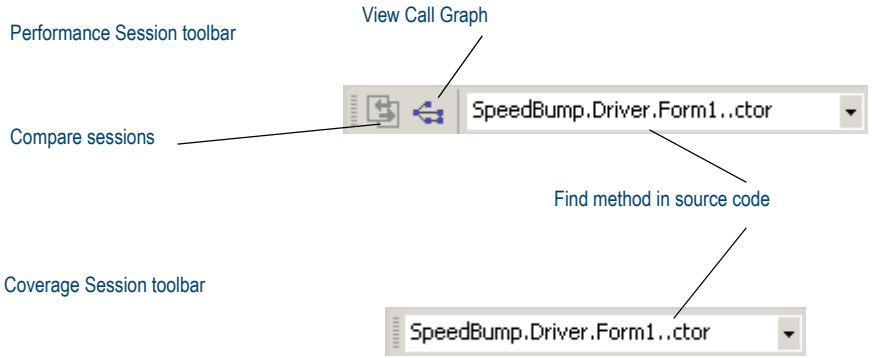
The following data collection properties apply to performance, coverage, and memory analysis.

Property	Default setting
Automatically Merge Session Files	Ask me if I would like to merge it
Collect information about .NET assemblies	True
Collect COM Information	True
Exclude Others	True
Instrument inline functions	True
Instrumentation Level	Line
Track System Objects	True

DevPartner toolbar buttons for Coverage, Memory, and Performance



Performance and Coverage Analysis Session Toolbars



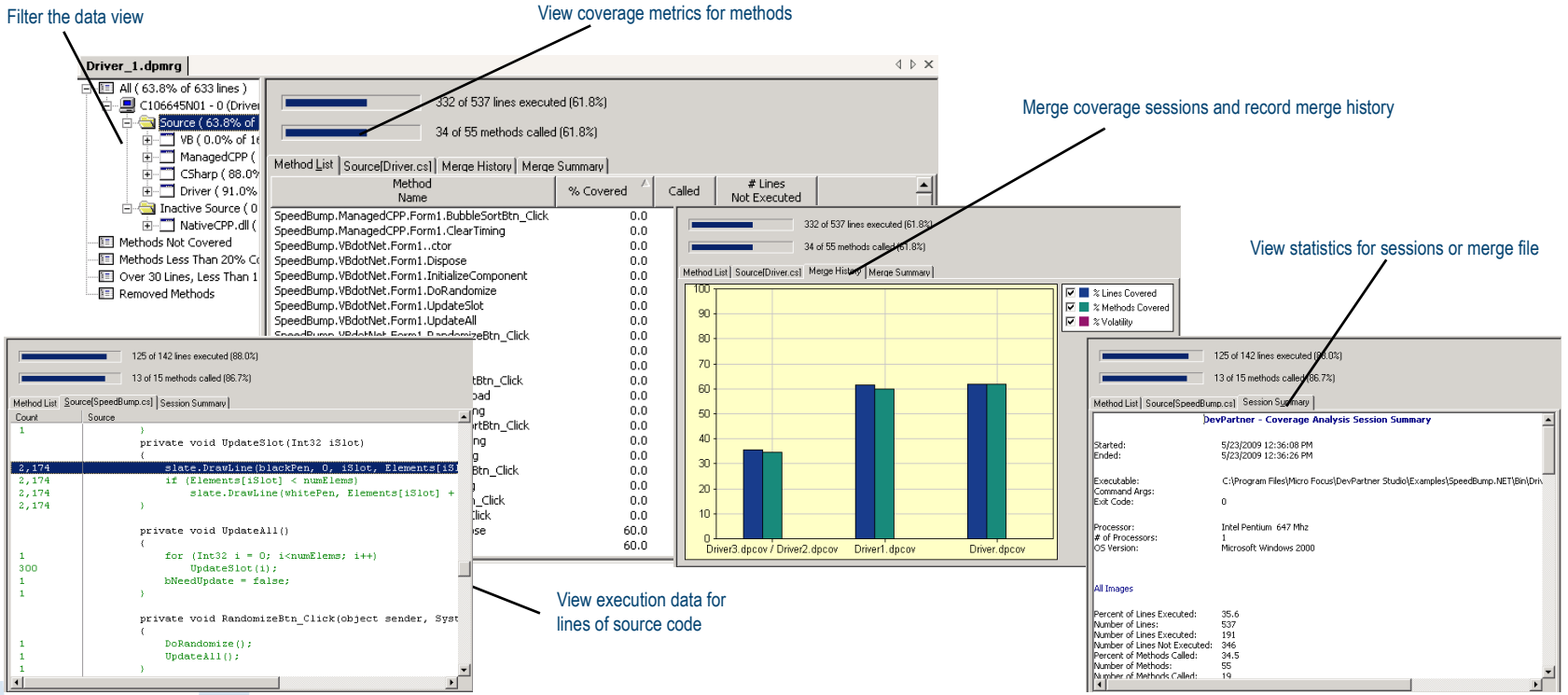
Coverage Analysis

Coverage Analysis Session Data

Results Summaries

DevPartner displays results for coverage analysis in Visual Studio or in the Coverage Analysis Viewer. Session files present data in tabbed format, including the following tabs:

- Method List
- Source Code
- Merge History
- Session or Merge Summary



Memory Analysis

Session Control for memory analysis

Memory Leaks session controls

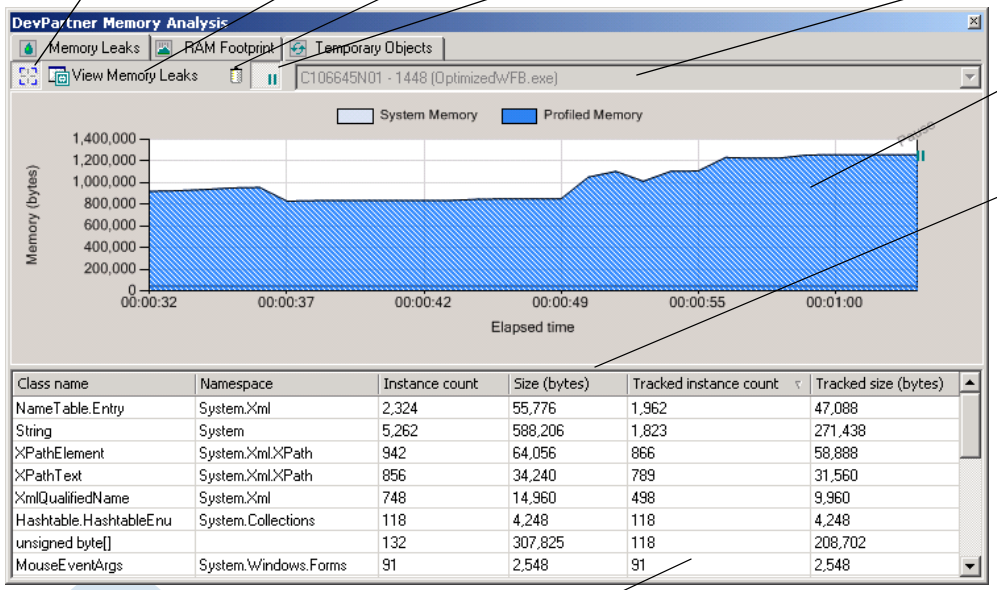
Start and Stop Tracking potential memory leaks

Take a memory leaks snapshot

Force a garbage collection

Pause real-time graph (data collection continues)

Choose process to profile

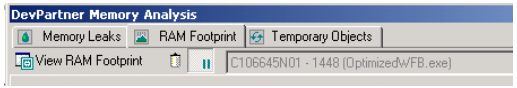


Graph shows state of managed heap in real time

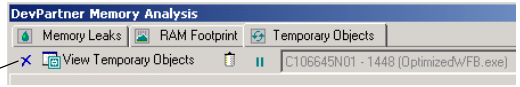
Class list dynamically updates to show what is in memory

Session controls tailored to type of memory data collection

RAM Footprint session controls:



Temporary Objects session controls:



In leak analysis, monitor **Tracked instance count** for objects that were not collected as expected

Clear temporary object allocations tracked to this point

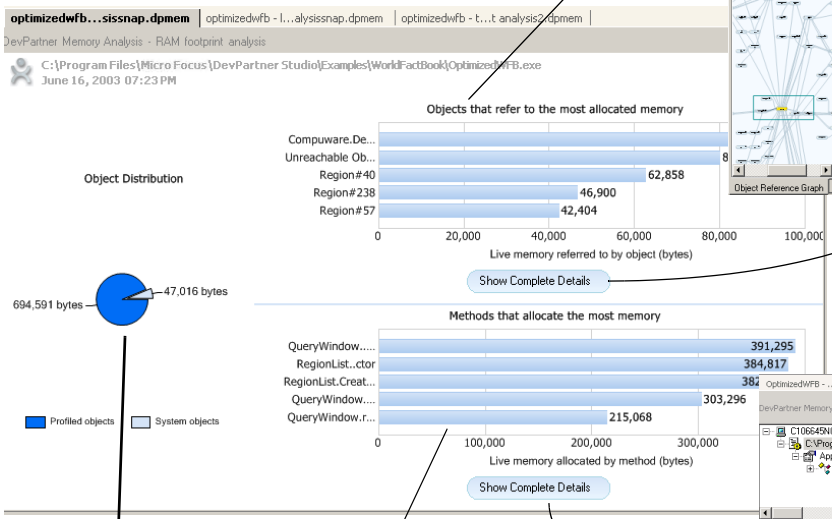
Memory Analysis Session Data

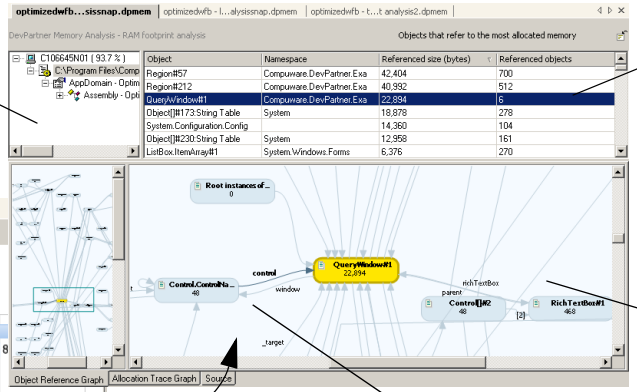
Results tailored to type data collection

- RAM Footprint
- Memory Leaks
- Temporary Objects
- Click Show Complete Details to view session data

Analyze object allocations in depth

Summary of most memory-intensive object allocations



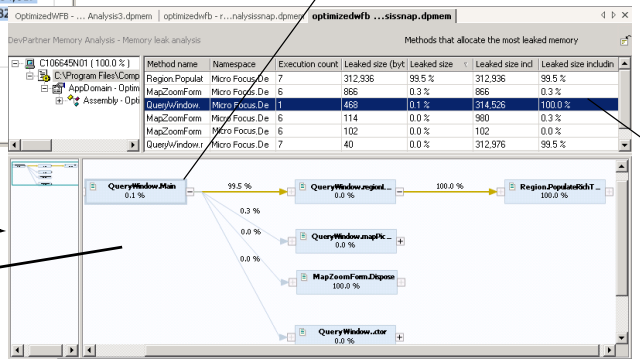


Drill down sequentially from any object in the list to examine referenced objects

Object Reference Graph

Trace object references back to the garbage collection roots that prevent objects from being collected. Answers the question: Why is this object still in memory?

Jump to the allocating source line from any method or object to edit source code



Drill down from any method in the list to examine allocated objects, and the objects they reference

Summary of most memory-intensive methods

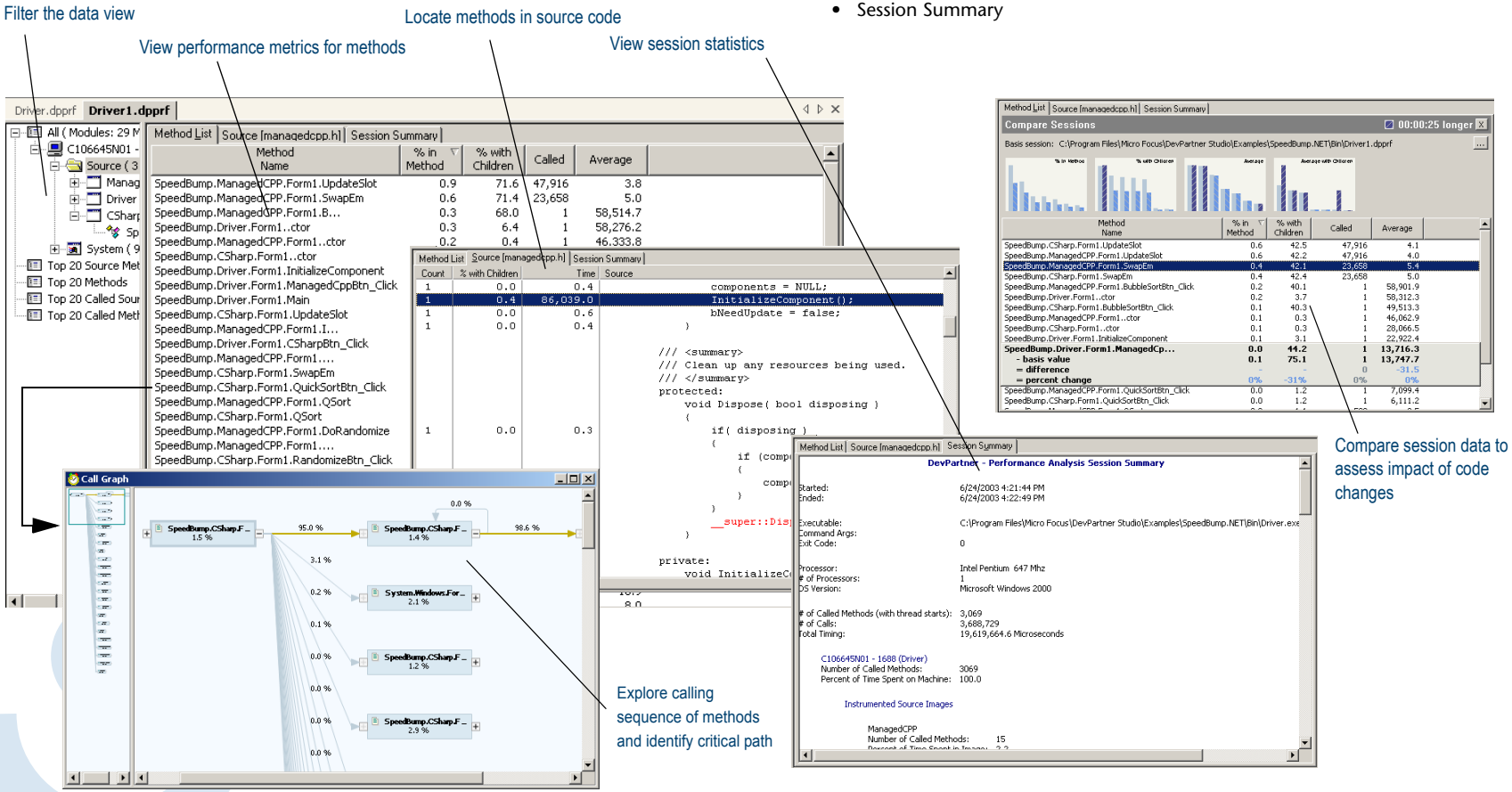
Object distribution: user vs. system objects (RAM footprint only)

Call Graph

Analyze the calling sequence of methods that allocated memory. Answers the question: Who allocated all that memory?

Performance Analysis

Performance Analysis Session Data



Results Summaries

DevPartner displays results for performance analysis in Visual Studio or in the Performance Analysis Viewer. Session files present data in tabbed format, including the following tabs:

- Method List
- Source Code
- Session Summary

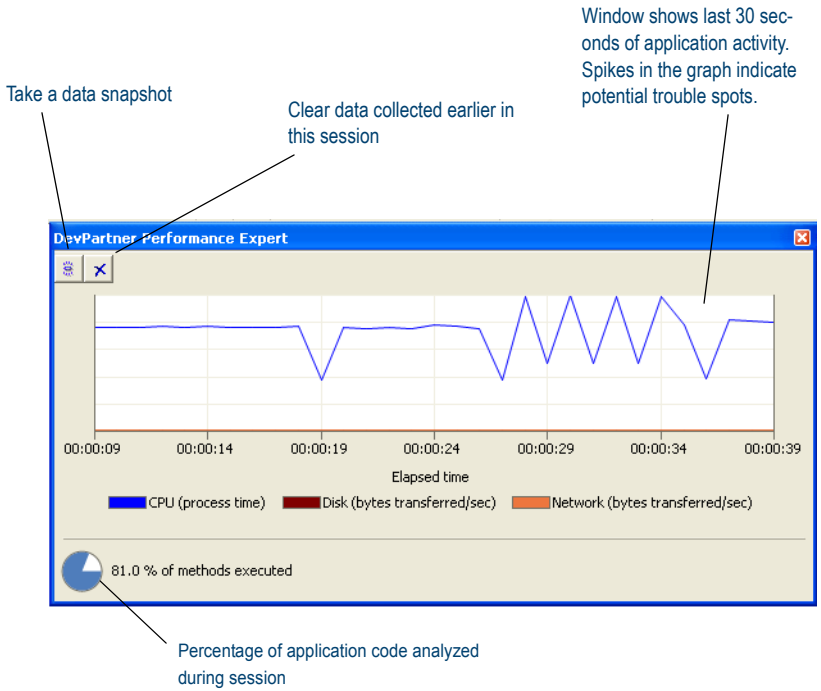
Performance Expert

Results Summaries

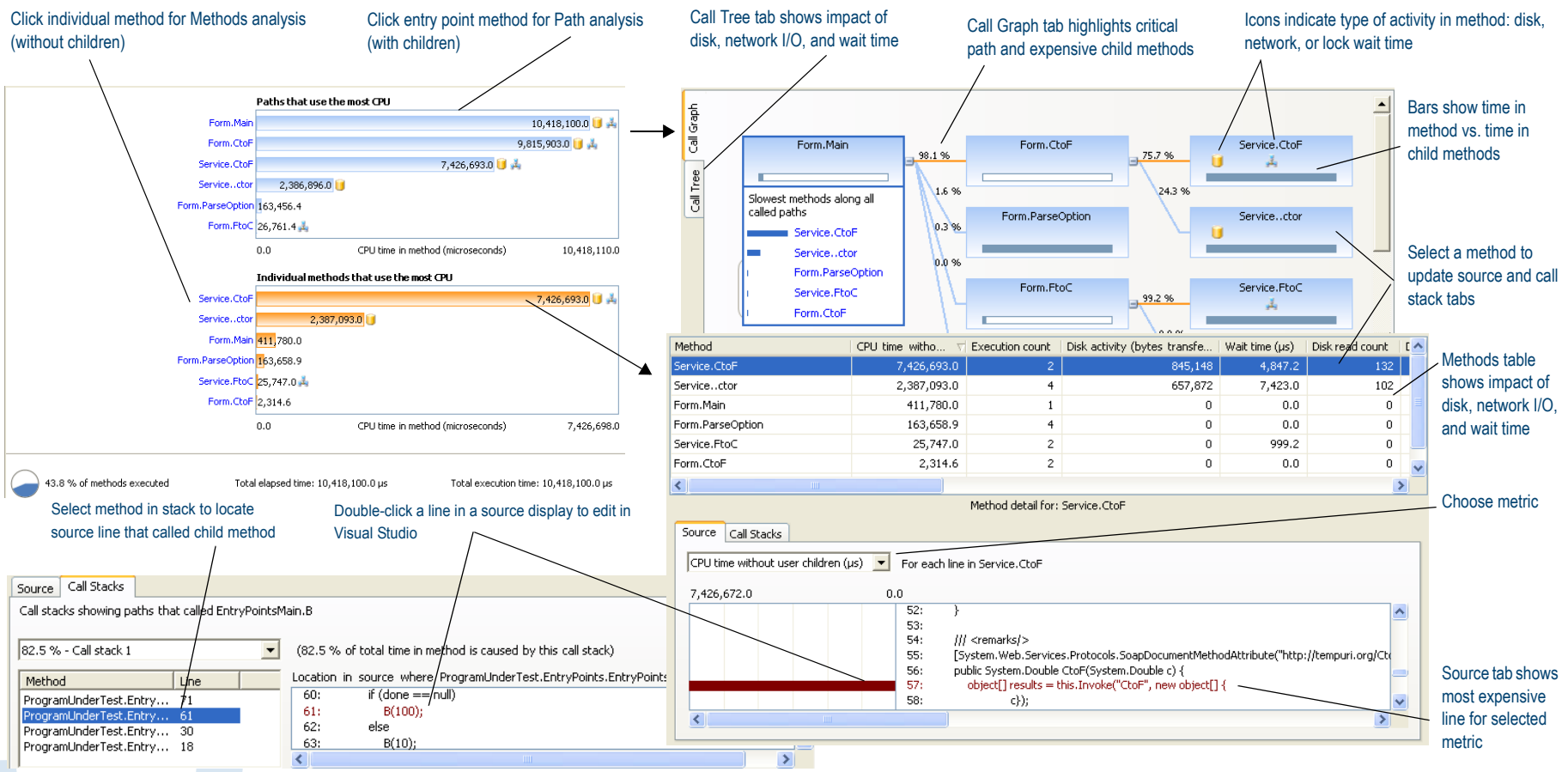
DevPartner displays results for Performance Expert in session files. Session files present data in tabbed format, including the following tabs:

- Call Graph
- Call Tree
- Methods table
- Source code
- Call stacks

Performance Expert Session Controls



Performance Expert Session Data



Using DPAnalysis.exe

Use DPAnalysis.exe to run coverage analysis, memory analysis, performance analysis, or Performance Expert sessions from the command line. DPAnalysis.exe accepts command line switches or an XML configuration file.

Command Line Operations

Use this syntax to run coverage, memory, performance, or Performance Expert sessions from the command line:

```
DPAnalysis.exe [a] {b} {c} {d} [e] target {target args}
```

DPAnalysis.exe requires Analysis Type and Target Type switches. Other switches are optional.

The following table lists the switches used with DPAnalysis.exe:

Category	Switches
[a] **Analysis Type**	/Cov[erage] - Sets analysis type to DevPartner coverage analysis
	/Mem[ory] - Sets analysis type to DevPartner memory analysis
	/Perf[ormance] - Sets analysis type to DevPartner performance analysis
	/Exp[ert] - Sets analysis type to DevPartner Performance Expert
{b} **Data Collection**	/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 will occur any time you run the specified process until you use the /D switch to disable profiling.
{c} **Other Options**	/O[utput] - Specify the session file output directory and/or filename
	/W[orkingDir] - Specify working directory for the process or service
	/H[ost] - Specify the target's host machine
	/NOWAIT - Do not wait for the process to exit, just wait for it to start
	/N[ewconsole] - Run the process in its own command window
	/F[orce] - Forces profiling for coverage or performance of applications written without managed code or CTI.
{d} **Analysis Options**	/NO_MACH5 - Disables excluding time spent on other threads
	/NM_METHOD_GRANULARITY - Sets data collection granularity to method-level (line-level is default)
	/EXCLUDE_SYSTEM_DLLS - Excludes data collection for system dlls (performance analysis only)
	/NM_ALLOW_INLINING - Enable run-time instrumentation of inline methods (coverage and performance analysis only)
	/NO_OLEHOOKS- Disable collection of COM
	/NM_TRACK_SYSTEM_OBJECTS - Track system object allocation (memory analysis only)
[e] **Target Type**	Identifies target that follows as either a process or service. Pick only one. All statements that follow the target name/path are considered arguments to the target
	/P[ocess] - Specify a target process (followed by arguments to process)
	/S[ervice] - Specify a target service (followed by arguments to service)
	/C[onfig] - Path to configuration file

Configuration File

Use this syntax to run coverage, memory, performance, or Performance Expert sessions through a configuration file:

DPAnalysis.exe /config c:\temp\config.xml

The following table briefly describes the XML elements. See the DevPartner online help or the *Understanding DevPartner* manual for more information.

Element	Description
AnalysisOptions	(Optional) For each Process or Service, zero or one. Defines runtime attributes for the specified target process or service. Attributes correspond to DevPartner properties accessible from the Properties Window in Visual Studio. *Attributes:* SESSION_DIR, SESSION_FILENAME, NM_METHOD_GRANULARITY, EXCLUDE_SYSTEM_DLLS, NM_ALLOW_INLINING, NO_OLEHOOKS, NM_TRACK_SYSTEM_OBJECTS, NO_MACH5
Arguments	(Optional) For each Process or Service, zero or one. Defines runtime attributes for the specified target process or service. Attributes correspond to DevPartner Coverage, Memory and Performance Analysis properties accessible from the Properties Window in Visual Studio. *Attributes:* SESSION_DIR, SESSION_FILENAME, NM_METHOD_GRANULARITY, EXCLUDE_SYSTEM_DLLS, NM_ALLOW_INLINING, NO_OLEHOOKS, NM_TRACK_SYSTEM_OBJECTS, NO_MACH5
ExcludeImages	(Optional) For each Process or Service, zero or one. No default if omitted. Defines images (at least one, no maximum) which, if loaded by the target process or service, will not be profiled. No attributes.
Host	(Optional) For each Process or Service, zero or one. No default if omitted. Sets the host machine of the target process or service. No attributes.
Name	One required for each service. Provides the name of the service as registered with the service control manager. This is the same name you would use for the system's NET START command. No attributes.
Path	One required for each process. 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 directory. No attributes.
Process	The configuration file must contain at least one Process or one Service element. Specifies a target executable. *Attributes:* CollectData, Spawn, NoWaitForCompletion, NewConsole
RuntimeAnalysis	Required; one only. Defines the type of analysis and maximum session time.
Service	The configuration file must contain at least one Process or one Service element. Specifies a target service. *Attributes:* CollectData, Start, RestartIfRunning, RestartAtEndOfRun
Targets	Required. One only. 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. *Attributes:* RunInParallel

Error Detection

File Extensions Used by Error Detection

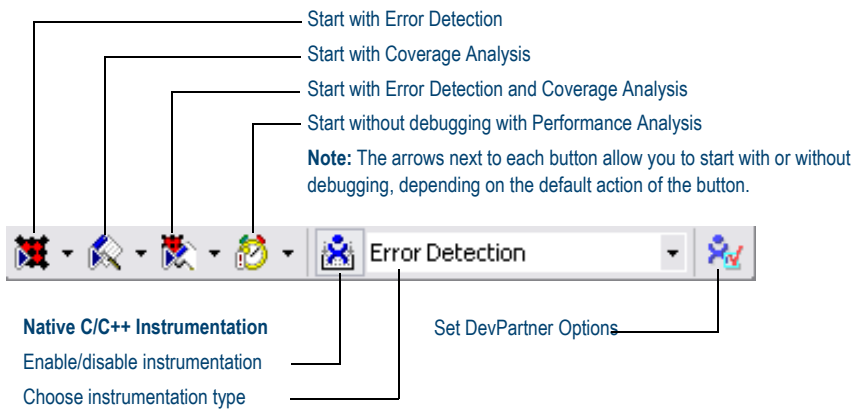
Extension	File Type	Description
.dpbcl	Error Detection Session File	This is the Error Detection log for the user's program execution.
.dpbcc .dpbcd	Error Detection Settings File	This file contains the various settings for Error Detection. The .dpbcd extension refers to the default settings file created, while .dpbcc refers to a custom settings file that has been saved separately.
.dpsup	Error Detection Suppressions File	This file contains the various suppressions for the user's program.
.dpflt	Error Detection Filters File	This file contains the various filters for the user's program.
.dprul	Error Detection Rules File	This is a database of the user's suppressions and filters.

Default Options (Visual Studio) or Settings (Visual C++)

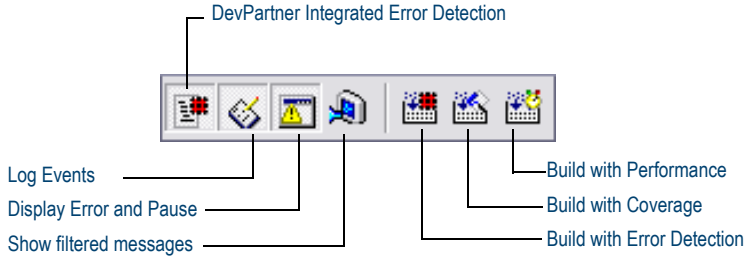
Category		Settings
General	On	Log events
	On	Display error and pause
	Off	Prompt to save program results
	Off	Show memory and resource viewer when application exits
	On	Source file search path - based on the location of the .EXE (standalone), .DSW (Visual C++), or .SLN (Visual Studio).
	-	Override symbol path - *Default: empty*
	-	Working directory (standalone only) based on the location of the .EXE
	-	Command line arguments (standalone only) - *Default: empty*
Data Collection	On	Call parameter coding depth = 1
	On	Maximum call stack depth on allocation = 5
	On	Maximum call stack depth on error = 20
	On	NLB file directory is based on the location of the .EXE (standalone), .DSW (Visual C++), or .SLN (Visual Studio).
	On	Generate NLB files dynamically
API Call Reporting	Off	Enable API call reporting. *All selections are unavailable until you select this item.*
	-	Collect window messages - *Default when active: Off*
	-	Collect API method calls and returns. - *Default when active: On*
	-	View only modules needed by this application - *Default when active: On*
	-	All modules (tree view). - *Default when active: All selected*
Call Validation	Off	Enable call validation. *All selections unavailable until you select this item*
	-	Enable memory block checking - *Default when active: Off*
	-	Fill output argument before call - *Default when active: Off*
	-	COM failure codes - *Default when active: On*
	-	Check for COM "Not Implemented" return code - *Default when active: On*
	-	API failure codes - *Default when active: On*
	-	Check invalid parameter errors: API, COM - *Default when active: both On*
	-	Category: Handle and pointer arguments - *Default when active: On*
	-	Category: Flag, range and enumeration arguments - *Default when active: On*
	-	Check statically linked C run-time library APIs - *Default when active: On*
		DLLs to check for API errors (failures or invalid arguments) - *Default when active: All items selected*
COM Call Reporting	Off	Enable COM method call reporting on objects that are implemented in the selected modules
	-	Report COM method calls on objects implemented outside of the listed modules - *Default when active: On*
	-	All components tree view - *Default when active: All selected*
COM Object Tracking	Off	Enable COM object tracking
	-	All COM classes tree view - *Default when active: All selected*

Category		Settings
Deadlock Analysis	Off	Enable deadlock analysis
	-	Assume single process - *Default when active: On*
	-	Enable watcher thread - *Default when active: Off*
	-	Generate errors when: A critical section is re-entered - *Default when active: Off*
	-	Generate errors when: A wait is requested on an owned mutex - *Default when active: Off*
	-	Number of historical events per resource - *Default when active: 10*
	-	Report synchronization API timeouts - *Default when active: Off*
	-	Report wait limits or actual waits exceeding (seconds) - *Default when active: 60*
	-	Synchronization Naming Rules - *Default when active: Don't warn about resource naming*
Memory Tracking	On	Enable memory tracking
	Off	Enable Leak Analysis Only
	Off	Show leaked allocation blocks
	Off	Enforce strict reallocation semantics
	On	Enable FinalCheck
	On	Enable guard bytes; Pattern = FC; Count = 4 bytes
	-	Check heap blocks at runtime: On free
	On	Enable fill on allocation; Pattern = FB
	On	Check uninitialized memory; Size = 2 bytes
	On	Enable poison on free; Pattern = FD
.NET Analysis	Off	Enable .NET analysis
	-	Exception monitoring - *Default when active: On*
	-	Finalizer monitoring - *Default when active: On*
	-	COM interop monitoring - *Default when active: On*
	-	PInvoke interop monitoring - *Default when active: On*
	-	Interop reporting threshold - *Default when active: 1*
.NET Call Reporting	Off	Enable .NET method call reporting
	-	All types (tree view node) - *Default when active: Selected.*
	-	.NET User Assemblies (tree view node) - *Default when active: Selected*
	-	.NET System Assemblies (tree view node) - *Default when active: Not selected*
Resource Tracking	On	Enable resource tracking
	On	Resources tree view. All listed resources are selected by default

Error Detection Toolbar in Visual Studio



Error Detection Toolbar in Visual C++ 6.0



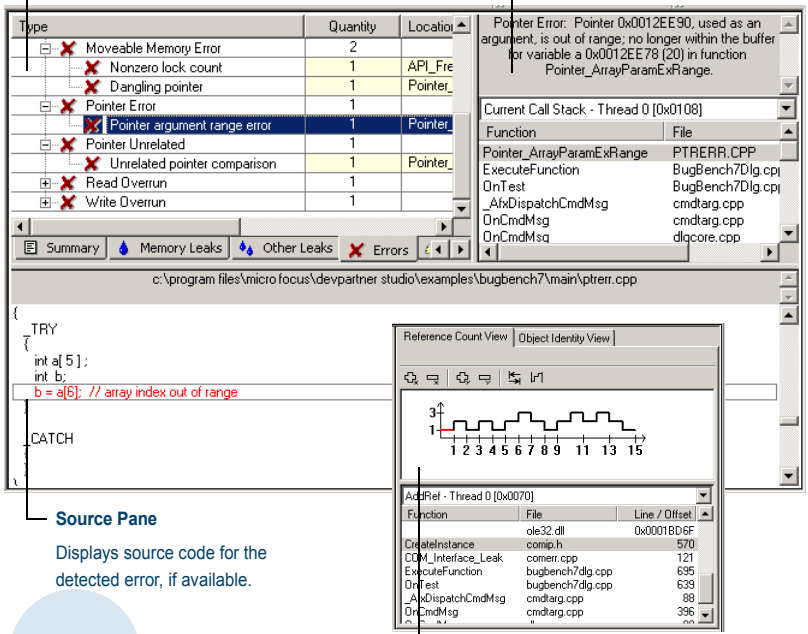
Error Detection Window

Results Pane

Summary, Memory Leaks, Other Leaks, Errors, .NET Performance, Modules, Transcript tabs provide overview and detail about detected errors.

Details Pane

Displays long description of detected error; call stack information; reference count graph (see inset below).



Source Pane

Displays source code for the detected error, if available.

Details Pane - Reference Count Graph

Displays Reference Count View and Object Identity View tabs when you select an Interface Leak in the Results pane.

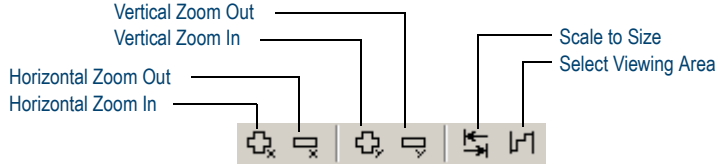
Icons Used in the Results Pane

Icon	Description	Appears in...
	Memory Leaks	Summary, Memory Leaks, and Transcript tabs
	Other Leaks	Summary, Other Leaks, and Transcript tabs
	Errors	Summary, Errors, and Transcript tabs
	.NET Performance	Summary, .NET Performance tabs
	Module Load Event	Summary, Modules, and Transcript tabs
	Subroutine call	Transcript tab
	Garbage Collection Event	Transcript tab
	Event Begins	Transcript tab
	Event Resumes	Transcript tab
	Event Ends	Transcript tab

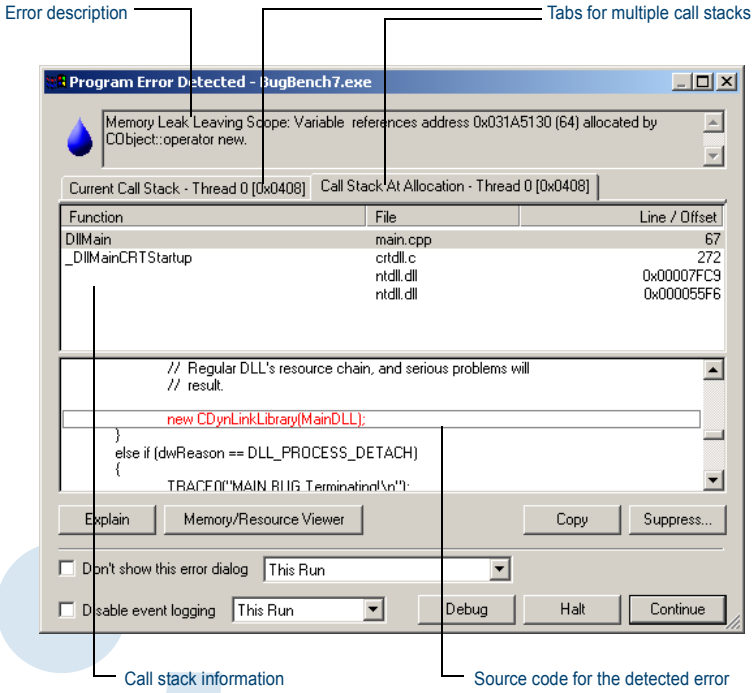
Icons Used in the Details Pane

Icon	Description
	Subroutine call
	Entry Parameters
	Exit Parameters
	Return Value
	Property (default) for data types
	Property for data types

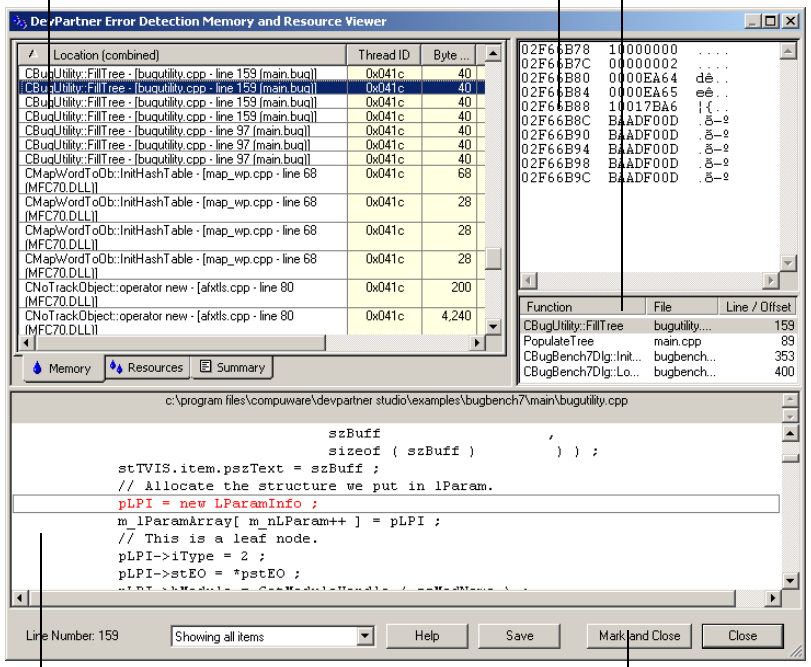
Reference Count Graph Toolbar



Program Error Detected Dialog Box



Memory and Resource Viewer Dialog Box



ActiveCheck and FinalCheck Error Detection

ActiveCheck

ActiveCheck™ analyzes your program and searches for errors in your program executable as well as the dynamic-link libraries (DLLs), third-party modules, and COM components used by your program. The following tables list the types of errors found with ActiveCheck error detection.

Deadlock-related Errors	API and COM Errors
Deadlock	COM interface method failure
Potential deadlock	Invalid argument
Thread deadlocked	Parameter range error
Critical section errors	Questionable use of thread
Semaphore errors	Windows function failed
Resource usage and naming errors	Windows function not implemented
Suspicious or questionable resource usage	Invalid COM interface method argument
Handle errors	
Event errors	
Mutex errors	
Windows event errors	

.NET Errors	Pointer and Leak Errors
Finalizer errors	Interface leak
GC.Suppress finalize not called	Memory leak
Dispose attributes errors	Resource leak
Unhandled native exception passed to managed code	

Memory Errors
Dynamic memory overrun
Freed handle is still locked
Handle is already unlocked
Memory allocation conflict
Pointer references unlocked memory block
Stack memory overrun
Static memory overrun

FinalCheck Compile Time Instrumentation - Deepest Error Detection

FinalCheck™ compile time instrumentation (CTI) enables Error Detection to find more errors (memory leaks, pointer errors, data corruption errors, and so on) as they occur in real time. FinalCheck finds these types of errors, plus all errors found with ActiveCheck.

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

List of Available Keyboard Commands - Visual Studio

Command	Action
Ctrl+Shift+O	File > Open > Project
Ctrl+Shift+N	File > New > Project
Ctrl+S	File > Save Project
Ctrl+Shift+S	File > Save All
Ctrl+Shift+F	Edit > Find in Files
Ctrl+Shift+H	Edit > Replace in Files
Alt+F12	Edit > Find Symbol
Ctrl+Alt+L	View > Solution Explorer
Ctrl+Shift+C	View > Class View
Ctrl+Alt+S	View > Server Explorer
Ctrl+Shift+E	View > Resource View
F4	View > Properties Window
Ctrl+Alt+X	View > Toolbox
Shift+Alt+Enter	View > Full Screen
Shift+F4	View > Property Pages
Ctrl+Shift+B	Build > Build Solution
F5	Debug > Start
Ctrl+F5	Debug > Start Without Debugging
Ctrl+Alt+E	Debug > Exceptions
F11	Debug > Step Into
F10	Debug > Step Over
Ctrl+B	Debug > New Breakpoint
Ctrl+F1	Help > Dynamic Help
Ctrl+Alt+F1	Help > Contents
Ctrl+Alt+F2	Help > Index
Ctrl+Alt+F3	Help > Search
Shift+Alt+F2	Help > Index results
Shift+Alt+F3	Help > Search results

List of Available Keyboard Commands - Visual C++ 6.0

Command	Action
Ctrl+F	Edit > Find
Ctrl+H	Edit > Replace
Ctrl+G	Edit > Go To
Alt+F2	Edit > Bookmarks
Alt+F9	Edit > Breakpoints
Ctrl+Alt+T	Edit > List Members
Ctrl+Shift+space	Edit > Parameter Info
Ctrl+Space	Edit > Complete Word
Ctrl+W	View > ClassWizard
Alt+0	View > Workspace
Alt+2	View > Output
Alt+Enter	View > Properties
Ctrl+F7	Build > Compile *filename*
F7	Build > Build *application_name*
F5	Build > Start Debug > Go
F11	Build > Start Debug > Step Into
Ctrl+F10	Build > Start Debug > Run to Cursor
Alt+F12	Tools > Source Browser
Ctrl+Shift+R	Tools > Record Quick Macro
Ctrl+Shift+S	Tools > Play Quick Macro

Export DevPartner Data: Command Line Use

You can use DevPartner.Analysis.DataExport.exe from the command line to convert DevPartner coverage analysis (.dpcov), coverage analysis merge (.dpmrg), performance analysis (.dpprf), and Performance Expert (.dppxp) session file data to XML.

Use this syntax to export session data to XML:

```
DevPartner.Analysis.DataExport.exe [sessionfilename|pathtodirectory] {options}
```

Options

The following table lists the command line options for DevPartner.Analysis.DataExport.exe.

You can use an equal sign, a colon, or a space to separate an option from the value or values you specify.

Switch	Description
`/out[put]=<String>`	Specify the local or remote output directory for exported XML files. Creates the directory if the directory does not exist
`/r[ecurse]`	Search subdirectories 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.
/method	Exports Performance Expert method data.
/calltree	Export Performance Expert call tree data.
/maxpaths=<integer>	Used only with the Performance Expert /summary option. Exports the specified number of the top call paths that use the most CPU resources.
/maxmethods=<integer>	Used only with the Performance Expert /summary option. Exports the specified number of the top methods that use the most CPU resources.