

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

Use the DevPartner online help to obtain “how to” information. See the *Understanding DevPartner Studio* manual for an overview of the DevPartner software.

Common Elements

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

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

DevPartner Toolbar

Accessed from the Visual Studio toolbar.

Toolbar button	Shortcut function for
	Run-time error detection using BoundsChecker technology
	Run-time code coverage analysis
	Run-time error detection with code coverage analysis
	Run-time performance analysis
	Run-time memory analysis
	Run-time analysis with Performance Expert
	Perform a review of the solution code
	Create and modify rules used during code reviews
	Compile-time instrumentation for error detection, coverage analysis, both error detection and coverage analysis, performance analysis
	DevPartner options for Analysis, Code review, Error detection

DevPartner Menu

Accessed from the Visual Studio Tools menu.

Choose this menu item	To
 Error detection	Perform run-time error detection using BoundsChecker technology
 Coverage Analysis	Perform run-time code coverage analysis

Choose this menu item	To
 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, Coverage analysis, Error detection and coverage analysis , Performance 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
Note: The Submit TrackRecord defect toolbar button is only available when TrackRecord is installed.	
 Options	Access DevPartner options Choices include: Analysis, Code review, Error detection

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

Run this DevPartner feature	To create this session file (extension)
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 error detection instrumentation. NMCL is not used with managed code, which DevPartner instruments as it is passed to the common language runtime as it executes.

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.

Use...	To...
/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

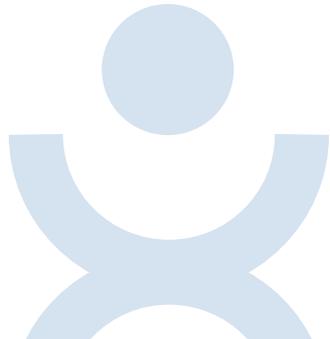
The following table lists the NMLINK options that you can use to link your unmanaged (native code) Visual C++ application to DevPartner.

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 performance and coverage 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

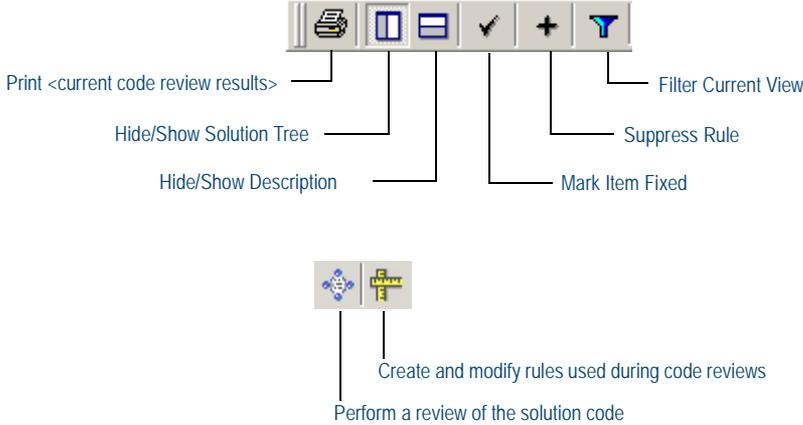
CRBatch /<switch>

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.

Code Review Default Options (General Node)

Category	Settings
Projects to be reviewed	All projects selected (Visual C++ .NET projects do not apply)
Rule set	All Rules
Naming analysis	On
Metrics analysis	Off
Ignore compile errors	Off
Exclude rules that require a build	Off
Always generate a batch file	On
Always save review results	On
Prompt for session file name	Off

Code Review Toolbar



Code Review Summaries

Summary of Problems *

Type	Problems			Severity		
	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	0	0	0	0	0	0
Garbage Collection	0	0	0	0	0	0
Internationalization	0	0	0	0	0	0
Language	0	0	0	0	0	0
Logic	0	0	0	0	0	0
Maintainability	0	0	0	0	0	0
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-NMSource\1666_MHT101515D01\DP5\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.sln
Session File	C:\p4_MHT-NMSource\1666_MHT101515D01\DP5\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.DPMD8
Batch Command Execution File	C:\p4_MHT-NMSource\1666_MHT101515D01\DP5\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.BAT

Project List

Project Name	Compile Errors	Reviewed	Project Path
Driver2003	False	True	C:\p4_MHT-NMSource\1666_MHT101515D01\DP5\DP_Mainline\Analysis\Examples\SpeedBump.Net\Driver\Driver2003.csproj
CSharp2003	False	True	C:\p4_MHT-NMSource\1666_MHT101515D01\DP5\DP_Mainline\Analysis\Examples\SpeedBump.Net\CSharp\CSharp2003.csproj
VB2003	False	True	C:\p4_MHT-NMSource\1666_MHT101515D01\DP5\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

Problems pane — displays rule-based programming problems

Naming pane — lists .NET naming violations and offers suggestions

Metrics pane — provides code complexity statistics

The screenshot displays the Visual Studio Code Review Results interface with several panes open:

- Problems pane:** Shows a table of rule-based programming problems. The table has columns for Fixed, Suppressed, Rule, Title, Severity, Project, and File. Three problems are listed, all with a severity of 'High'.
- Naming pane:** Shows a table of .NET naming violations. The table has columns for Fixed, Name, and Suggested. Four violations are listed for 'RulesUtil1'.
- Metrics pane:** Shows a table of code complexity statistics. The table has columns for Method, File, Project, Complexity, Bad Fix %, Understanding, and Lines of Code. It lists 41 metrics for 'SpeedBump.Net2003'.
- Call Graph pane:** Shows a graphical method call tree. The root node is 'Main()', which calls 'DoProcessing()', which calls 'DoAll()', which calls 'DoCompactDir()', which calls 'GetRulesDBPath()', which calls 'GetDBPathFromDB()'.

The **Problems pane** details for the first problem are as follows:

Fixed	Suppressed	Rule	Title	Severity	Project	File
<input type="checkbox"/>	<input type="checkbox"/>	1099	Literal, hard-coded string found in c...	High	CSharp2003	SpeedBum...
<input type="checkbox"/>	<input type="checkbox"/>	1580	Type not excluded from use by untr...	High	Driver2003	Driver.cs
<input type="checkbox"/>	<input type="checkbox"/>	1099	Literal, hard-coded string found in c...	High	VB2003	VBdotNet...
<input type="checkbox"/>	<input type="checkbox"/>	1099	Literal, hard-coded string found in c...	High	VB2003	VBdotNet...

The **Problems pane** details for the first problem are as follows:

Literal, hard-coded string found in code

Trigger: [Detected Literal, hard-coded, string in code \[Occurrences: 1\]](#)

Original Source Line: `label1.Text = "";`

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 the resource file. Use the [ResourceManager](#) class to retrieve the information from the resource file.

The following example code uses the [ResourceManager](#) class:

```
Visual C# .NET example:  
  
ResourceManager rm = new ResourceManager("MyLocalizedResources", Assembly.GetExecutingAssembly());  
string foo = rm.GetString("string1");
```

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

Select analysis preference

Coverage

Error Detection with Coverage

Native C/C++ Instrumentation

Enable/disable instrumentation

Choose instrumentation type

Performance Analysis

Performance Expert

Memory

Set DevPartner Options

Performance and Coverage Analysis Session Toolbars

Performance Session toolbar

Compare sessions

View Call Graph

Find method in source code

Coverage Session toolbar



Coverage Analysis

Coverage Analysis Session Data

Results Summaries

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

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

Filter the data view

View coverage metrics for methods

Merge coverage sessions and record merge history

View statistics for sessions or merge file

The screenshot displays the DevPartner Coverage Analysis interface with several components:

- Left Panel:** A tree view showing the project structure. The 'Source' folder is selected, showing 63.8% coverage of 633 lines. Other folders include 'ManagedCPP', 'CSharp', 'Driver', 'Inactive Source', and 'NativeCPP'.
- Top Panel:** Summary statistics for the selected source: 332 of 537 lines executed (61.8%) and 34 of 55 methods called (61.8%).
- Method List Table:** A table with columns: Method Name, % Covered, Called, # Lines Not Executed. It lists methods like 'SpeedBump.ManagedCPP.Form1.BubbleSortBtn_Click' with 0.0% coverage.
- Bar Chart:** A chart comparing coverage metrics for three sessions: 'Driver3.dpcov / Driver2.dpcov', 'Driver1.dpcov', and 'Driver.dpcov'. The y-axis represents percentage coverage from 0 to 100.
- Source Code View:** A window showing the source code for 'UpdateSlot' and 'UpdateAll' methods. Line 2,174 is highlighted.
- Session Summary Panel:** A panel titled 'DevPartner - Coverage Analysis Session Summary' showing details like 'Started: 5/23/2003 12:36:08 PM', 'Command Args: 0', and hardware information.

View execution data for lines of source code

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

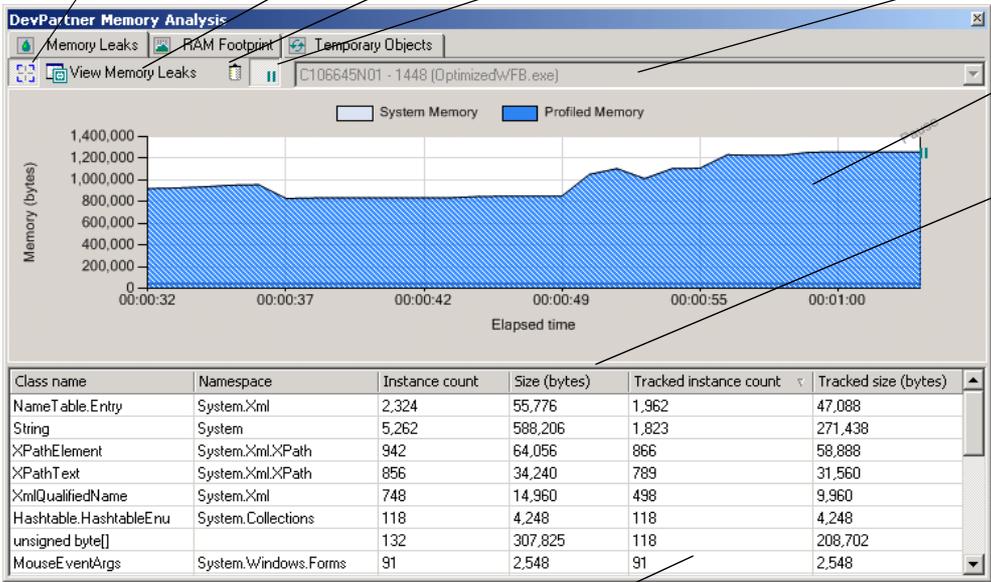
Class list dynamically updates to show what is in memory

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

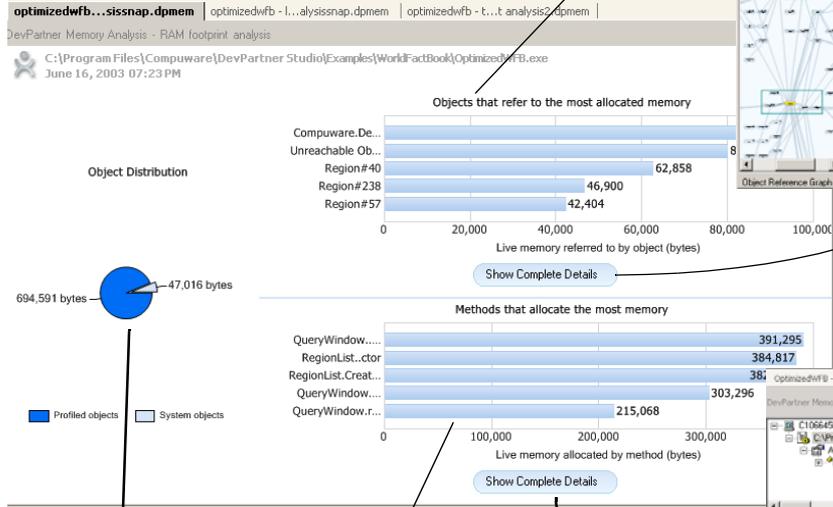
Memory Analysis Session Data

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



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

Summary of most memory-intensive methods

Call Graph

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

Objects that refer to the most allocated memory

Object	Namespace	Referenced size (bytes)	Referenced objects
Region#57	Compuware.DevPartner.Exa	42,404	700
Region#212	Compuware.DevPartner.Exa	40,932	512
QueryWindow#1	Compuware.DevPartner.Exa	22,884	6
Object#173 String Table	System	18,878	278
System Configuration Config	System	14,360	104
Object#1820 String Table	System	12,860	161
ListBox.ItemArray#1	System.Windows.Forms	6,376	270

Methods that allocate the most leaked memory

Method name	Namespace	Execution count	Leaked size (byt)	Leaked size %	Leaked size ind	Leaked size includ
Region.Populat	Compuware.De	7	312,936	99.5 %	312,936	99.5 %
MapZoomForm	Compuware.De	6	102	0.3 %	102	0.3 %
QueryWindow...	Compuware.De	1	400	0.1 %	314,526	100.0 %
MapZoomForm	Compuware.De	6	114	0.0 %	980	0.3 %
MapZoomForm	Compuware.De	6	102	0.0 %	102	0.0 %
QueryWindow	Compuware.De	7	40	0.0 %	312,976	99.5 %

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

Performance Analysis

Performance Analysis Session Data

Filter the data view

View performance metrics for methods

Locate methods in source code

View session statistics

The screenshot shows the DevPartner Performance Analysis tool interface. On the left, a tree view shows the project structure. The main window is divided into three panes:

- Method List:** A table showing performance metrics for various methods.

Method Name	% in Method	% with Children	Called	Average
SpeedBump.ManagedCPP.Form1.UpdateSlot	0.9	71.6	47,916	3.8
SpeedBump.ManagedCPP.Form1.SwapEm	0.6	71.4	23,658	5.0
SpeedBump.ManagedCPP.Form1.B...	0.3	68.0	1	58,514.7
SpeedBump.Driver.Form1...ctor	0.3	6.4	1	58,276.2
SpeedBump.ManagedCPP.Form1...ctor	0.2	0.4	1	46,333.8
- Source Code:** A window showing the source code for the selected method, with a line of code highlighted: `components = NULL; InitializeComponent();`
- Call Graph:** A diagram showing the sequence of method calls. The primary path is from `SpeedBump.CSharp.F...` (1.5%) to `SpeedBump.CSharp.F...` (1.4%) with a 95.0% call rate, and then to `SpeedBump.CSharp.F...` (2.3%) with a 98.6% call rate.

Results Summaries

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

- Method List
- Source Code
- Session Summary

The screenshot shows the 'Compare Sessions' dialog box. It features a bar chart comparing two sessions. Below the chart is a table with the following data:

Method Name	% in Method	% with Children	Called	Average
SpeedBump.CSharp.Form1.UpdateSlot	0.6	42.5	47,916	4.1
SpeedBump.ManagedCPP.Form1.UpdateSlot	0.6	42.2	47,916	4.0
SpeedBump.ManagedCPP.Form1.SwapEm	0.4	42.1	23,658	5.4
SpeedBump.CSharp.Form1.SwapEm	0.4	42.4	23,658	5.0
SpeedBump.ManagedCPP.Form1.BubbleSortBtn_Click	0.2	40.1	1	58,901.9
SpeedBump.Driver.Form1...ctor	0.2	3.7	1	58,312.3
SpeedBump.CSharp.Form1.BubbleSortBtn_Click	0.1	40.3	1	49,513.3
SpeedBump.ManagedCPP.Form1...ctor	0.1	0.3	1	46,062.9
SpeedBump.CSharp.Form1...ctor	0.1	0.3	1	28,066.5
SpeedBump.Driver.Form1.InitializeComponent	0.1	3.1	1	22,932.4
SpeedBump.Driver.Form1.ManagedCp...	0.0	44.2	1	13,747.3
- basis value	0.1	75.1	1	7,099.4
= difference	0	-31.5	0	-31.5
= percent change	0%	-31%	0%	0%
SpeedBump.ManagedCPP.Form1.QuickSortBtn_Click	0.0	1.2	1	7,099.4
SpeedBump.CSharp.Form1.QuickSortBtn_Click	0.0	1.2	1	6,111.2

Compare session data to assess impact of code changes

The screenshot shows the 'DevPartner - Performance Analysis Session Summary' dialog box. It contains the following information:

- Started:** 6/24/2003 4:21:44 PM
- Ended:** 6/24/2003 4:22:49 PM
- Executable:** C:\Program Files\Compuware\DevPartner Studio\Examples\SpeedBump.NET\Bin\Driver.exe
- Command Args:** 0
- Exit Code:** 0
- Processor:** Intel Pentium 647 Mhz
- # of Processors:** 1
- OS Version:** Microsoft Windows 2000
- # of Called Methods (with thread starts):** 3,069
- # of Calls:** 3,688,729
- Total Timing:** 19,619,664.6 Microseconds
- C106645N01 - 1688 (Driver)**
 - Number of Called Methods: 3069
 - Percent of Time Spent on Machine: 100.0
- Instrumented Source Images:**
 - ManagedCPP: Number of Called Methods: 15, Percent of Time Spent on Machine: 2.9

Explore calling sequence of methods and identify critical path

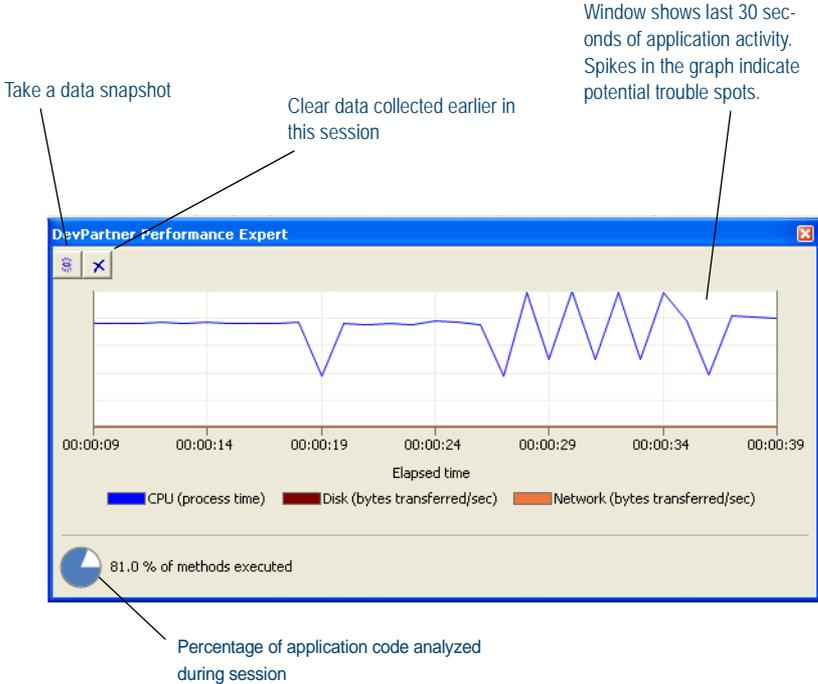
Performance Expert

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

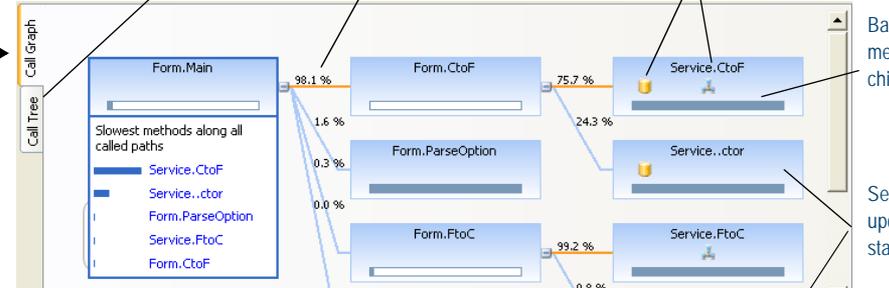
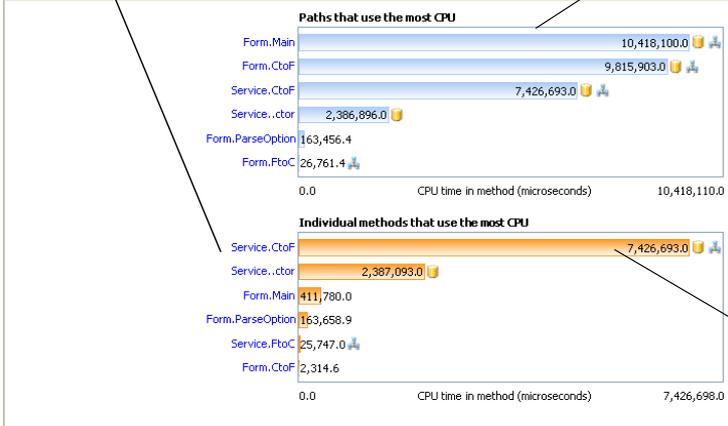
Click individual method for Methods analysis (without children)

Click entry point method for Path analysis (with children)

Call Tree tab shows impact of disk, network I/O, and wait time

Call Graph tab highlights critical path and expensive child methods

Icons indicate type of activity in method: disk, network, or lock wait time



Bars show time in method vs. time in child methods

Select a method to update source and call stack tabs

Method	CPU time without...	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

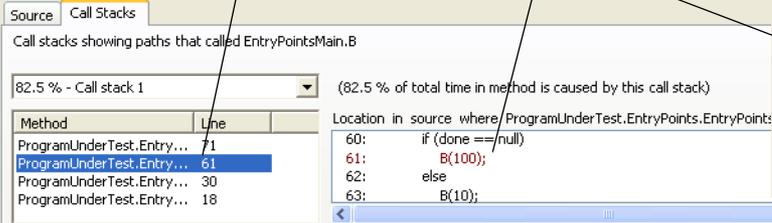
Methods table shows impact of disk, network I/O, and wait time

Choose metric

43.8 % of methods executed Total elapsed time: 10,418,100.0 µs Total execution time: 10,418,100.0 µs

Select method in stack to locate source line that called child method

Double-click a line in a source display to edit in Visual Studio



Source tab shows most expensive line with metrics

Using DPAnalysis.exe

Use DPAnalysis.exe to run Coverage, Memory, Performance, and Performance Expert sessions launched directly from the command line, or through a configuration file called through the command line.

Command Line Operations

Use this syntax to run Coverage, Memory, Performance, or Performance Expert sessions from the command line:

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

DPAnalysis.exe requires Analysis Type and Target Type switches. Use of other switches is 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

Category	Switches
{c} Other Options	/WAIT - In batch files with multiple targets, launches the next process only after the current process exits. /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
{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 (Perf 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 only)
[e] Target Type	Identifies target to follow as either a process or service. Pick only one. All arguments that follow the target name/path will be 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] - Path to configuration file



Configuration File

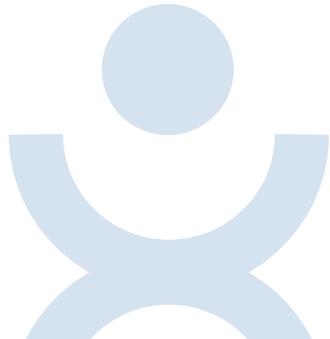
Use this syntax to run Coverage, Memory, Performance, or Performance Expert sessions through a configuration file:

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

The following table briefly describes the XML elements. See the online help for more details.

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. <i>Attributes:</i> SESSION_DIR, SESSION_FILENAME, NM_METHOD_GRANULARITY, EXCLUDE_SYSTEM_DLLS, NM_ALLOW_INLINING, NO_OLEHOOKS, NM_TRACK_SYSTEM_OBJECTS, NO_MACHS
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 properties accessible from the Properties Window in Visual Studio. <i>Attributes:</i> SESSION_DIR, SESSION_FILENAME, NM_METHOD_GRANULARITY, EXCLUDE_SYSTEM_DLLS, NM_ALLOW_INLINING, NO_OLEHOOKS, NM_TRACK_SYSTEM_OBJECTS, NO_MACHS
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.

Element	Description
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. <i>Attributes:</i> CollectData, Spawn, NoWaitForCompletion, NewConsole
RuntimeAnalysis	Required; one only. Defines the type of performance and maximum session time.
Service	The configuration file must contain at least one Process or one Service element. Specifies a target service. <i>Attributes:</i> 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. <i>Attributes:</i> 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.
.dpfft	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++)

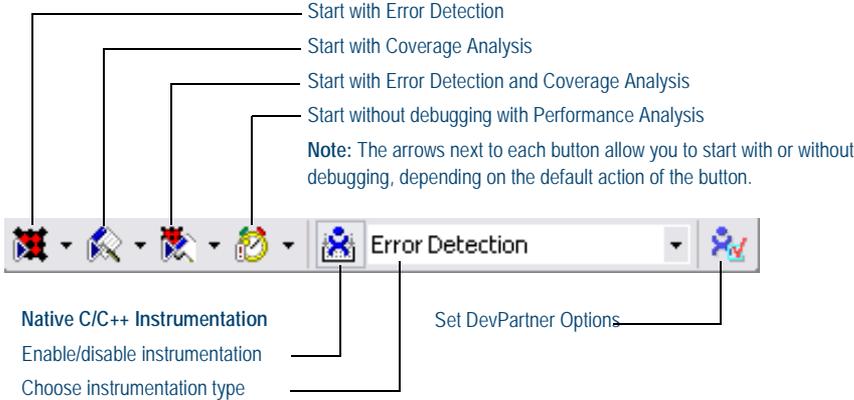
Category	Settings
General	On Log events
	On Display error and pause
	Off Prompt to save program results
	Off Show memory and resource viewer when application exits
	On Source file search path - based on the location of the .EXE (standalone), .DSW (Visual C++), or .SLN (Visual Studio).
	- Override symbol path - <i>Default: empty</i>
	- Working directory (standalone only) based on the location of the .EXE
	- Command line arguments (standalone only) - <i>Default: empty</i>
	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).	
Off Generate NLB files dynamically	

Data Collection

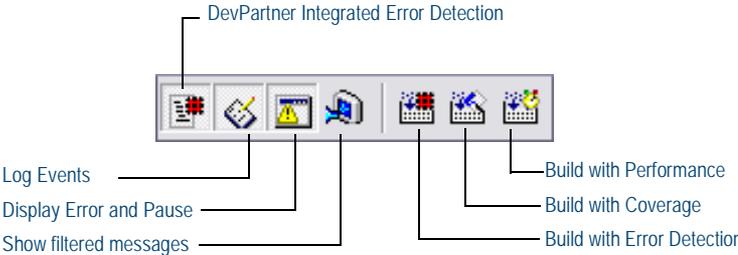
Category	Settings
API Call Reporting	Off Enable API call reporting. <i>All selections are unavailable until you select this item.</i> <ul style="list-style-type: none"> - Collect window messages - <i>Default when active: Off</i> - Collect API method calls and returns. - <i>Default when active: On</i> - View only modules needed by this application - <i>Default when active: On</i> - All modules (tree view). - <i>Default when active: All selected</i>
Call Validation	Off Enable call validation. <i>All selections unavailable until you select this item</i> <ul style="list-style-type: none"> - Enable memory block checking - <i>Default when active: Off</i> - Fill output argument before call - <i>Default when active: Off</i> - COM failure codes - <i>Default when active: On</i> - Check for COM "Not Implemented" return code - <i>Default when active: On</i> - API failure codes - <i>Default when active: On</i> - Check invalid parameter errors: API, COM - <i>Default when active: both On</i> - Category: Handle and pointer arguments - <i>Default when active: On</i> - Category: Flag, range and enumeration arguments - <i>Default when active: On</i> - Check statically linked C run-time library APIs - <i>Default when active: On</i> - DLLs to check for API errors (failures or invalid arguments) - <i>Default when active: All items selected</i>
COM Call Reporting	Off Enable COM method call reporting on objects that are implemented in the selected modules <ul style="list-style-type: none"> - Report COM method calls on objects implemented outside of the listed modules - <i>Default when active: On</i> - All components tree view - <i>Default when active: All selected</i>
COM Object Tracking	Off Enable COM object tracking <ul style="list-style-type: none"> - All COM classes tree view - <i>Default when active: All selected</i>

Category	Settings
Deadlock Analysis	Off Enable deadlock analysis
	- Assume single process - <i>Default when active: On</i>
	- Enable watcher thread - <i>Default when active: Off</i>
	- Generate errors when: A critical section is re-entered - <i>Default when active: Off</i>
	- Generate errors when: A wait is requested on an owned mutex - <i>Default when active: Off</i>
	- Number of historical events per resource - <i>Default when active: 10</i>
	- Report synchronization API timeouts - <i>Default when active: Off</i>
	- Report wait limits or actual waits exceeding (seconds) - <i>Default when active: 60</i>
	- Synchronization Naming Rules - <i>Default when active: Don't warn about resource naming</i>
Memory Tracking	On Enable memory tracking
	On Report leaks immediately
	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 - <i>Default when active: On</i>
	- Finalizer monitoring - <i>Default when active: On</i>
	- COM interop monitoring - <i>Default when active: On</i>
	- PlInvoke interop monitoring - <i>Default when active: On</i>
	- Interop reporting threshold - <i>Default when active: 1</i>
.NET Call Reporting	Off Enable .NET method call reporting
	- All types (tree view node) - <i>Default when active: Selected.</i>
	- .NET User Assemblies (tree view node) - <i>Default when active: Selected</i>
	- .NET System Assemblies (tree view node) - <i>Default when active: Not selected</i>
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.

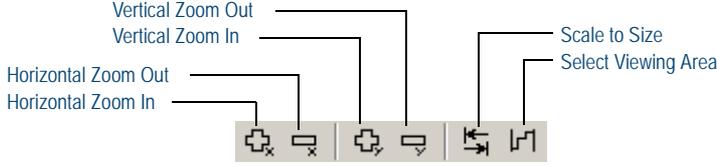
Icons Used in the Results Pane

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

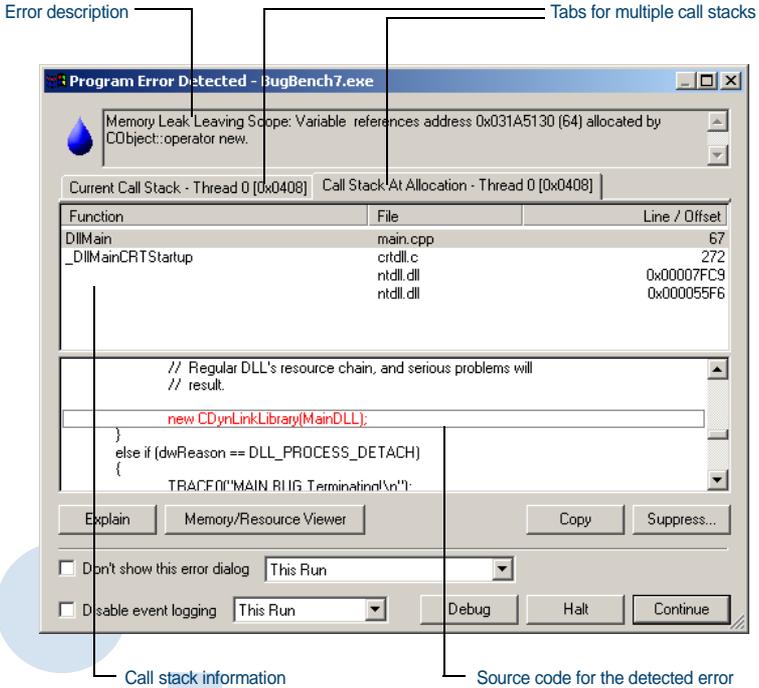
Icons Used in the Details Pane

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

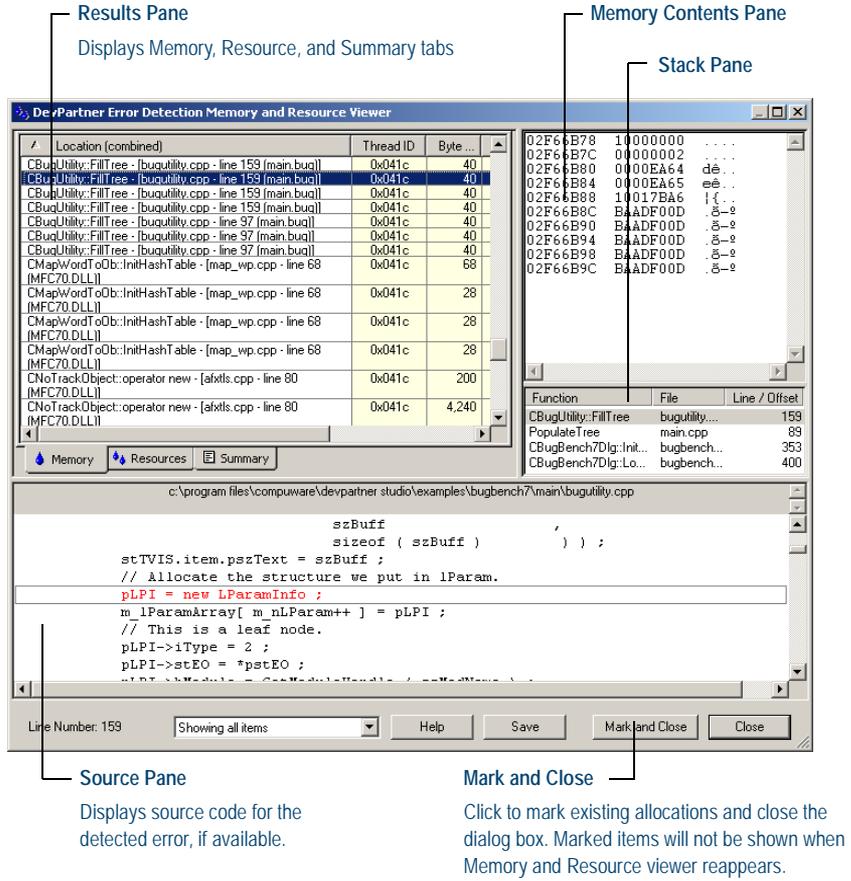
Reference Count Graph Toolbar



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.

Table with 2 columns: Deadlock-related Errors and API and COM Errors. Lists various error types such as Deadlock, Potential deadlock, Thread deadlocked, etc.

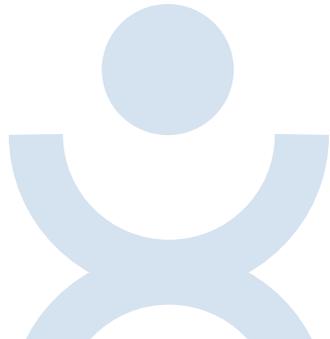
Table with 2 columns: .NET Errors and Pointer and Leak Errors. Lists errors like Finalizer errors, GC.Suppress finalize not called, etc.

Table with 1 column: Memory Errors. Lists errors such as Dynamic memory overrun, Freed handle is still locked, etc.

FinalCheck Compile Time Instrumentation - Deepest Error Detection

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

Table with 2 columns: Memory Errors and Pointer and Leak Errors. Lists errors like Reading overflows buffer, Array index out of range, etc.



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 <i>filename</i>
F7	Build > Build <i>application_name</i>
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