

DPVC 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
Diagnose run-time errors in the source code	Error Detection
Locate performance bottlenecks in the application	Coverage and Performance Analysis
Ensure code base stability throughout development and testing phases	Coverage Analysis Session Data

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..

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
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
 Options	Access DevPartner options Choices include: Analysis, Code review, Error detection

DevPartner File Extensions

File extensions for session files.

Running this DevPartner feature	Creates this session file (extension)
Code coverage	.dpcov
Code coverage merge files	.dpmrg
Error detection	.dpbcl
Performance analysis	.dpprf

Command Line Instrumentation Options

NMCL Options

The following table lists the NMCL options that you can use to instrument your unmanaged (native) C++ code from the command line. Use NMCL.EXE only to compile unmanaged 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.

Use...	To...
/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.

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\Micro Focus\NMShared

Note: For installs on 64-bit versions of Windows, add the following directory to your path:

C:\Program Files (x86)\Common Files\Micro Focus\NMShared

NMLINK Options

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

Use...	To...
/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.

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\Micro Focus\NMShared

Note: For installs on 64-bit versions of Windows, add the following directory to your path:

C:\Program Files (x86)\Common Files\Micro Focus\NMShared



Coverage and Performance Analysis

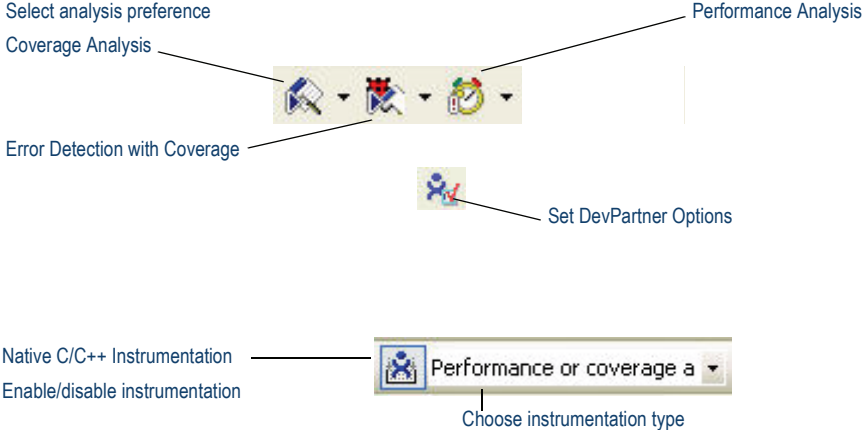
Determine application test coverage and profile application performance.

General and Data Collection Properties

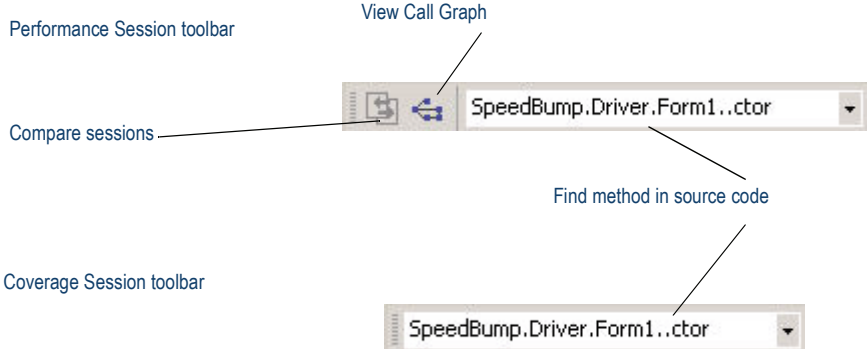
The following data collection properties apply to coverage and performance analysis.

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 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

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:

- Source Tree:** Shows a project structure for 'Driver01.dpmrg' with sub-items like 'All (96.1% of 205 lines)', 'DTWLIB4M-053 - 0 (Driver)', 'Source (96.1% of 205 lines)', 'Driver.exe (95.4% of 67 lines)', 'SpeedBump.Driver.F', 'CSharp.dll (96.4% of 14 lines)', and 'SpeedBump.CSharp'.
- Summary Panel:** Shows '197 of 205 lines executed (96.1%)' and '20 of 22 methods called (90.9%)'.
- Method List Table:**

Method Name	% Covered	Called	# Lines
SpeedBump.CSharp.Form1.ClearTiming(void)	0.0	62 of 65 lines executed (95.4%)	
SpeedBump.Driver.Form1.NativeCppBtn_Click(Object, EventArgs)	0.0		
SpeedBump.CSharp.Form1.Dispose(Boolean)	60.0	6 of 7 methods called (85.7%)	
SpeedBump.Driver.Form1.Dispose(Boolean)	80.0		
SpeedBump.CSharp.Form1.QSort(Int32, Int32)	94.7		
SpeedBump.CSharp.Form1.ctor(void)	100.0		
SpeedBump.CSharp.Form1.InitializeComponent(void)	100.0		
SpeedBump.CSharp.Form1.Form1_Load(Object, EventArgs)	100.0		
SpeedBump.CSharp.Form1.DoRandomize(void)	100.0		
- Bar Chart:** A grouped bar chart comparing coverage metrics for 'driver3.dpcov / driver1.dpcov', 'driver2.dpcov', and 'driversnap'. The legend indicates:
 - Blue bar: % Lines Covered
 - Green bar: % Methods Covered
 - Purple bar: % Volatility
- Source Code View:** Shows the source code for 'Source[SpeedBump.cs]' with a 'Count' column on the left. The code includes:


```
private const Int32 numItems = 300; //Adjust this down
private Int32[] Elements = new Int32[numItems];
private Bool BHeedUpDate = false;

private Graphics slate;
private Pen whitePen = new Pen(Color.White, 1);
private Pen blackPen = new Pen(Color.Black, 1);
private System.Windows.Forms.Label labell;

// <summary>
// Required designer variable.
// </summary>
private System.ComponentModel.IContainer components = null;

public Form1()
{
    // Required for Windows Form Designer support
    //
    InitializeComponent();
}
```
- Summary Panel (Bottom Right):** Shows '62 of 65 lines executed (95.4%)' and '6 of 7 methods called (85.7%)'. It includes a 'DevPartner - Coverage Analysis Session Summary' with details:
 - Started: 2/28/2009 5:32:17 PM
 - Ended: 2/28/2009 5:33:17 PM
 - Executable: C:\SpeedBump.Net\Driver\bin\Release\Driver.exe
 - Command Args: 0
 - Exit Code: 0
 - Processor Speed: 2992 Mhz
 - # of Processors: 1
 - OS Version: Microsoft Windows XP
- Merge History:** A list of sessions:
 - Driver3.dpcov
 - Driver1.dpcov (2/28/2008 5:40:50 PM Administrator)
 - Driver2.dpcov (2/28/2008 5:40:50 PM Administrator)
 - DriverSnap1.dpcov (2/28/2008 5:42:00 PM Administrator)

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

- Method List:** A table with columns: Method Name, % in Method, % with Children, Called, and Average (us).

Method Name	% in Method	% with Children	Called	Average (us)
SpeedBump.CSharp.Form1.UpdateSlot(Int32)	0.1	3.5	97,632	3.7
SpeedBump.CSharp.Form1.SwapEm(Int32, Int32)	0.1	3.5	94,632	3.2
SpeedBump.CSharp.Form1.BubbleSortBtn_Click(Object, EventArgs)	0.0	3.4	4	23,165.8
SpeedBump.Driver.Form1.ctor(void)	0.0	1.1	1	45,462.6
- Source [SpeedBump.cs]:** Shows the source code for the selected method:


```
private void QuickSortBtn_Click(object sender, System.EventArgs e)
{
    StartTiming("Timing: ");
    QuickSortBtn.Enabled = false;
    BubbleSortBtn.Enabled = false;
    QSort(0, numElems-1);
    if (!bNeedUpdate) UpdateAll();
    RandomizeBtn.Enabled = true;
    EndTiming("Quick Sort:");
}

private void BubbleSortBtn_Click(object sender, System.EventArgs e)
{
    StartTiming("Timing: ");
    QuickSortBtn.Enabled = false;
    BubbleSortBtn.Enabled = true;
    for(i = numElems - 1; i > 0; i--)
    {
        if (!bNeedUpdate) UpdateAll();
        Swap();
    }
    if (!bNeedUpdate) UpdateAll();
}

```
- Call Graph:** A flow diagram showing the sequence of methods. The top node is 'SpeedBump.Driver.Form1.Main(void)' (0.0%), which calls 'System.Windows.Form1.ShowDialog()' (0.1%) and 'SpeedBump.CSharp.Form1.Main(void)' (0.0%). 'System.Windows.Form1.ShowDialog()' calls 'SpeedBump.CSharp.Form1.Main(void)' (0.0%) and 'RtlGetLastWin32Error()' (100.0%). 'SpeedBump.CSharp.Form1.Main(void)' calls 'RtlAllocateHeap()' (100.0%).

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

The 'Compare Sessions' window shows a comparison between two sessions. It includes a bar chart and a table with columns: Method Name, % in Method, % with Children, Called, and Average (us). The table highlights the difference and percent change between the two sessions.

Method Name	% in Method	% with Children	Called	Average (us)
SpeedBump.CSharp.Form1.UpdateSlot(Int32)	0.1	2.7	98,632	3.8
SpeedBump.CSharp.Form1.SwapEm(Int32, Int32)	0.1	2.7	94,632	3.1
SpeedBump.CSharp.Form1.BubbleSortBtn_Click(Object, EventArgs)	0.0	2.6	4	25,168.3
SpeedBump.Driver.Form1.ctor(void)	0.0	0.8	1	46,499.4
SpeedBump.CSharp.Form1.ctor(void)	0.0	0.1	3	8,562.1
SpeedBump.CSharp.Form1.QSort(Int32, Int32)	0.0	0.1	2,396	6.3
SpeedBump.Driver.Form1.InitializeComponent(void)	0.0	0.3	1	14,867.8
SpeedBump.CSharp.Form1.DoRandomize(void)	0.0	0.1	8	555.7
SpeedBump.CSharp.Form1.InitializeComponent(void)	0.0	0.1	3	893.8
SpeedBump.Driver.Form1.NativeCppBtn_Click(Object, EventArgs)	0.0	0.0	2	1,241.5
SpeedBump.CSharp.Form1.EndTiming(String)	0.0	0.0	8	293.4
SpeedBump.CSharp.Form1.QuickSortBtn_Click(Object, EventArgs)	0.0	0.1	4	586.2
SpeedBump.CSharp.Form1.UpdateAll(void)	0.0	0.1	12	192.8
SpeedBump.Driver.Form1.CSharpBtn_Click(Object, EventArgs)	0.0	12.2	3	688.1
- basis value	0.0	22.6	1	1,984.6
= difference	0%	-10%	200%	-6.5%
= percent change	0%	0%	0%	4%

Compare session data to assess impact of code changes

Explore calling sequence of methods and identify critical path

Using DPAnalysis.exe

Use DPAnalysis.exe to run coverage or performance analysis 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 or performance sessions from the command line:

```
DPAnalysis.exe [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 /Perf[ormance] - Sets analysis type to DevPartner performance analysis
[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[eppeat] - Profiling will occur any time you run the specified process until you use the /D switch to disable profiling.

Category	Switches
{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 /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 /F[orce] - Forces profiling for coverage or performance of applications written without managed code or CTI.
{d} Analysis Options	/NO_QUANTUM - 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[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 or performance analysis 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_QUANTUM
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_QUANTUM
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. 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 (DevPartner Studio Professional and Enterprise Editions) or Settings (Visual C++ BoundsChecker Suite)

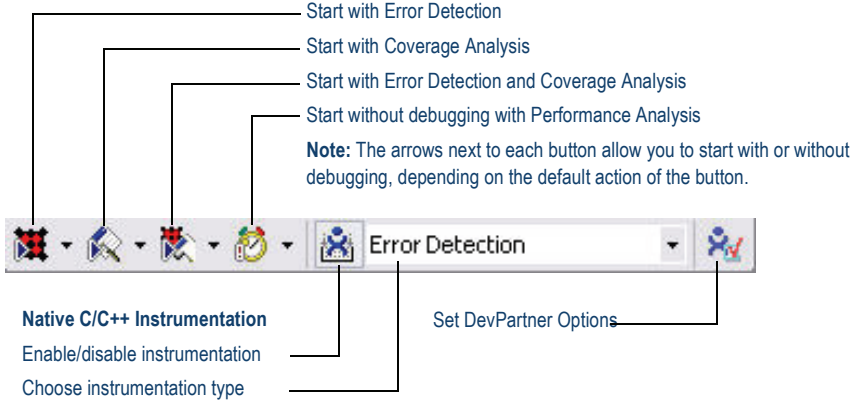
Category	Settings
General	On Log events
	On Display error and pause
	Off Prompt to save program results
	Off Show memory and resource viewer when application exits
	On Source file search path - based on the location of the .EXE (standalone), .DSW (C++), or .SLN (Visual Studio).
	- Override symbol path (standalone only) - <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
	Off Use raw stack capture method
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 (C++), or .SLN (Visual Studio).	

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>
Deadlock Analysis	Off Enable deadlock analysis <ul style="list-style-type: none"> - 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>

Category	Settings
Deadlock Analysis (continued)	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> On Enable memory tracking Off Enable Leak Analysis Only Off Show leaked allocation blocks On Enforce strict reallocation semantics On Enable FinalCheck On Enable guard bytes; Pattern: (0x) = FC; Count = 4 bytes <ul style="list-style-type: none"> - Check heap blocks at runtime: On free On Enable fill on allocation; Pattern: (0x) = FB On Check uninitialized memory; Size = 2 bytes On Enable poison on free; Pattern: (0x) = FD
.NET Analysis	<ul style="list-style-type: none"> Off Enable .NET analysis <ul style="list-style-type: none"> - 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> - PInvoke interop monitoring - <i>Default when active: On</i> - Interop reporting threshold - <i>Default when active: 1</i>
.NET Call Reporting	<ul style="list-style-type: none"> Off Enable .NET method call reporting <ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> On Enable resource tracking On Resources tree view. All listed resources are selected by default

Error Detection Toolbar in Visual Studio



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.

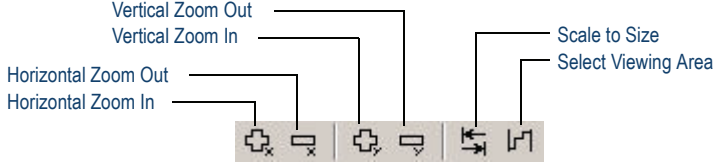
Icons Used in the Results Pane

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

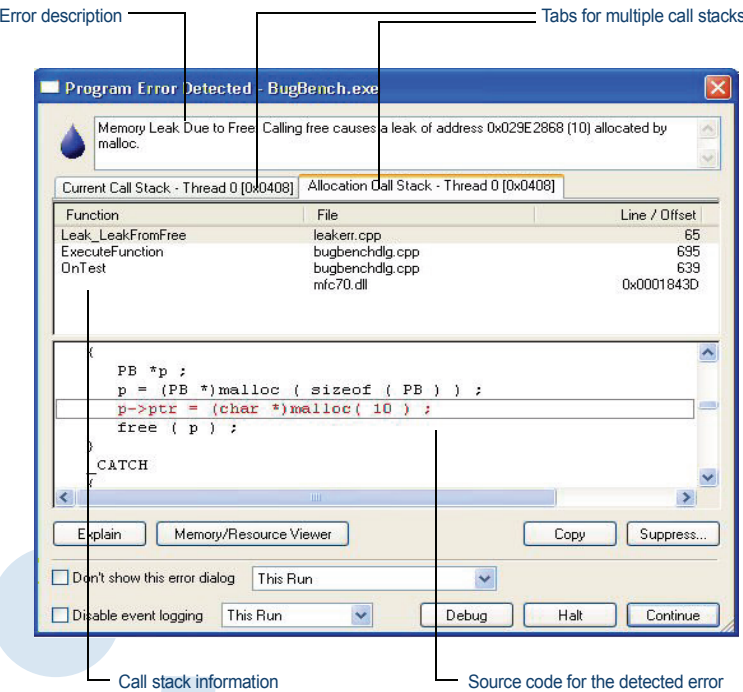
Icons Used in the Details Pane

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

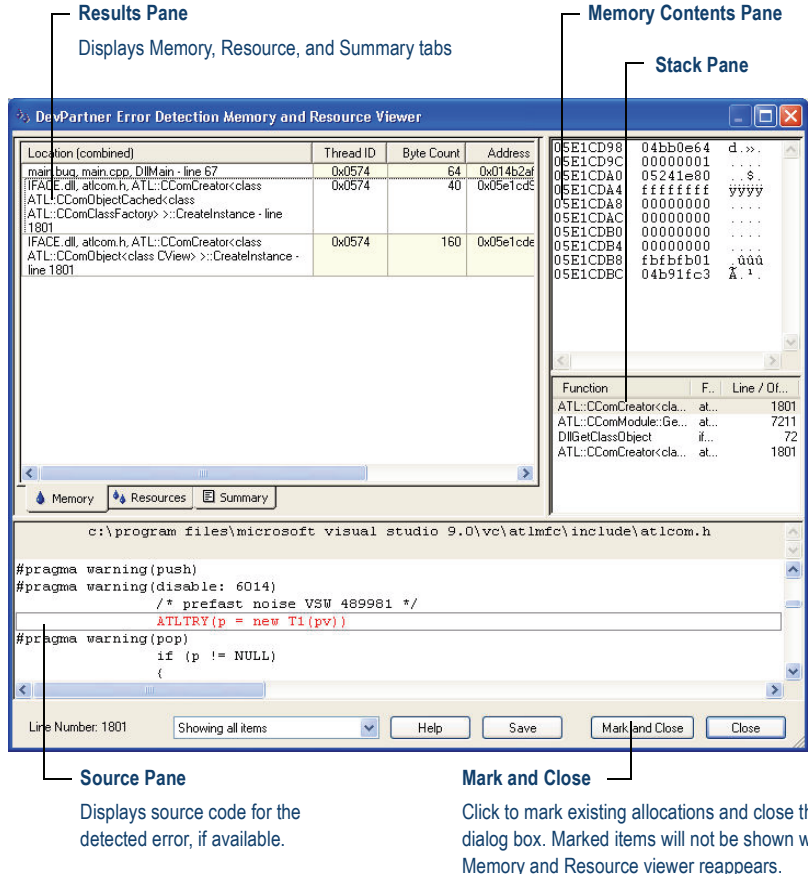
Reference Count Graph Toolbar



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, 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.

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*

* Visual Studio 2005, 2008

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), and performance analysis (.dpprf) 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
<code>/out [put] =<String></code>	Specify the output directory for exported XML files. Creates the directory if the directory does not exist.
<code>/r [recurse]</code>	Search subdirectories for DevPartner session files.
<code>/f [ilename] =<String></code>	Specify the name of the XML output file. Appends .xml to the name specified.
<code>/showAll</code>	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.
<code>/w [ait]</code>	Wait for input before closing console window.
<code>/nologo</code>	Do not display the logo or copyright notice.
<code>/help or /?</code>	Display help in the console window.