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.

Note: Options 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
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) 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 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

Coverage and Performance Analysis

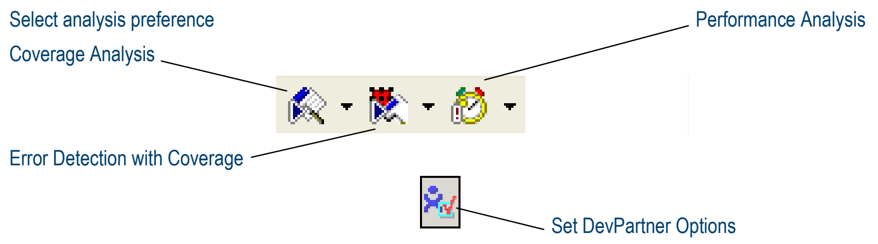
Determine application test coverage and profile application performance.

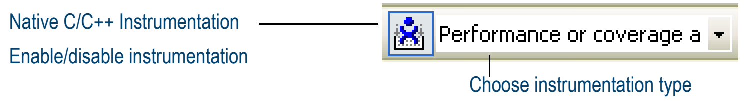
General and Data Collection Properties

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

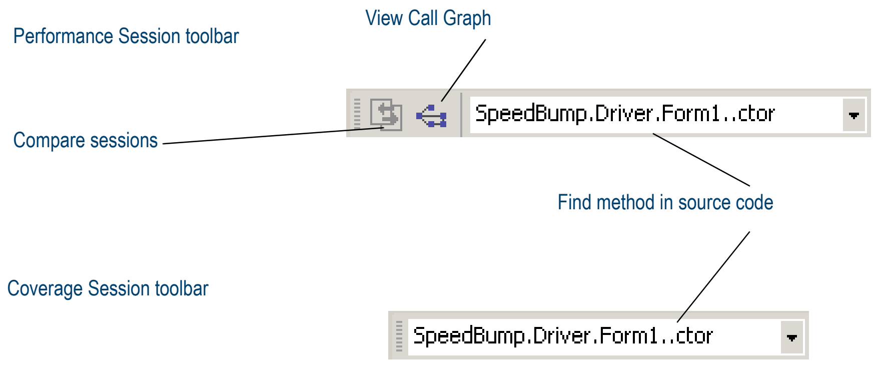
DevPartner toolbar buttons for Coverage and Performance




Native C/C++ Instrumentation
Enable/disable instrumentation
Performance or coverage a
Choose instrumentation type

Performance and Coverage Analysis Session Toolbars

In Visual Studio 6.0, use the context menu to compare sessions and view a call graph.


Performance Session toolbar
View Call Graph
SpeedBump.Driver.Form1..ctor
Compare sessions
Find method in source code
Coverage Session toolbar
SpeedBump.Driver.Form1..ctor

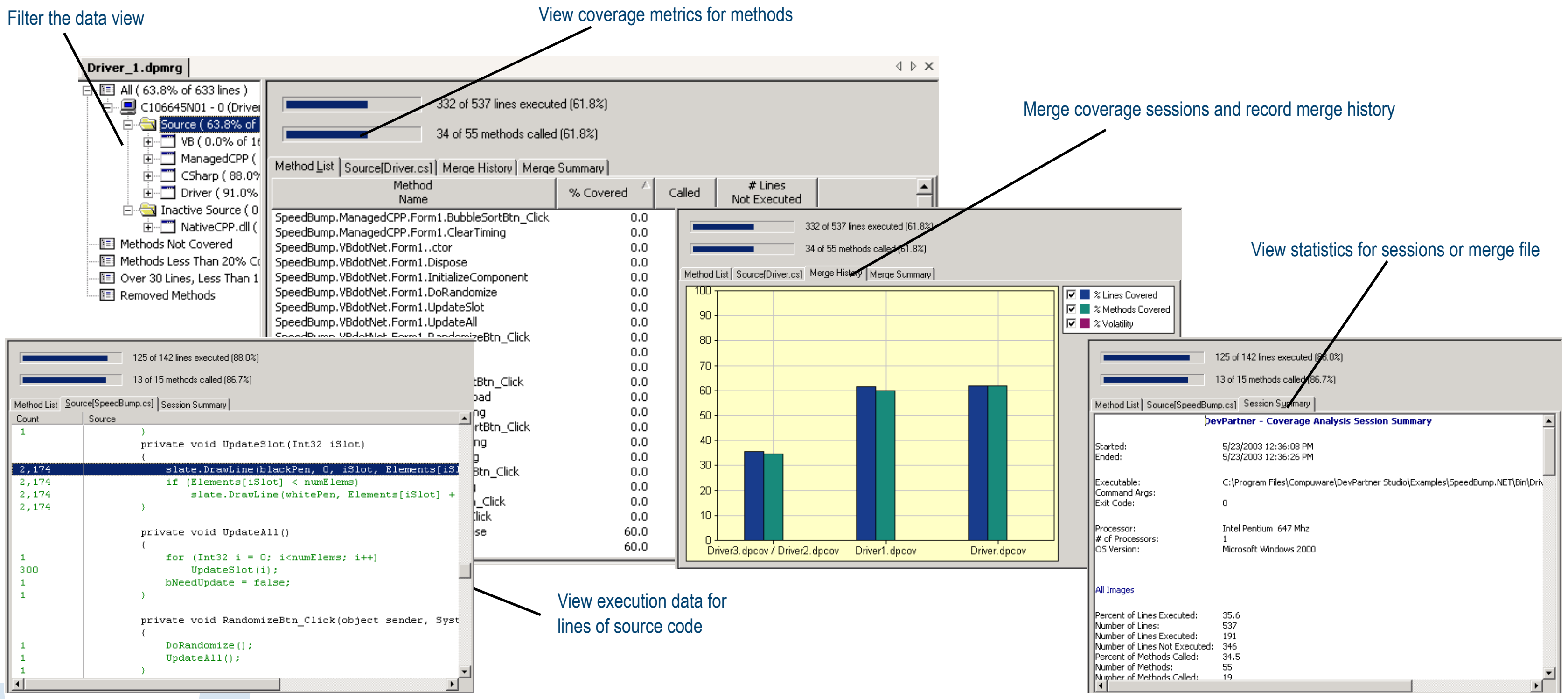
Coverage Analysis

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



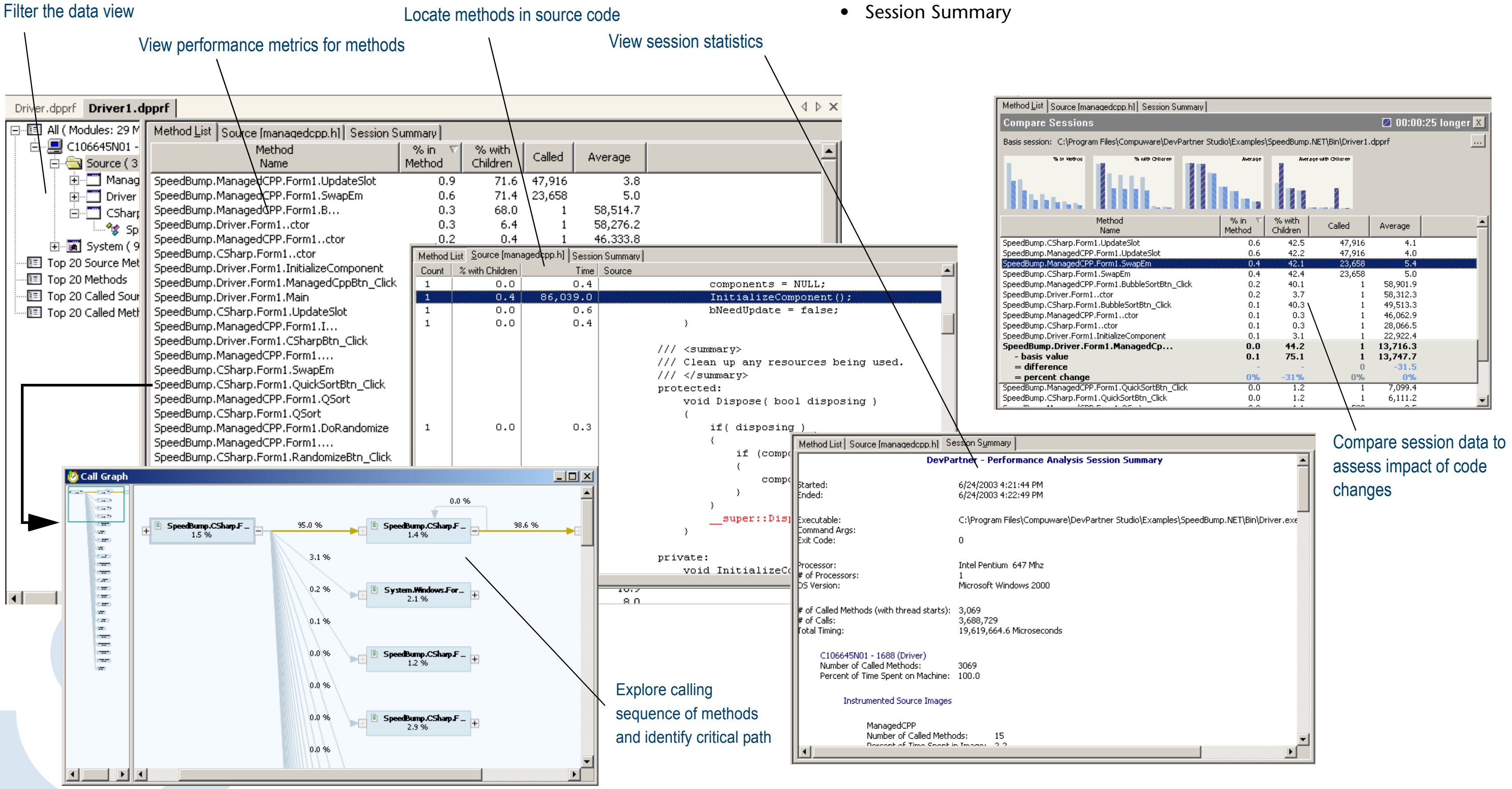
```
332 of 537 lines executed (61.8%)
34 of 55 methods called (61.8%)

Method Name                                        % Covered   Called   # Lines Not Executed
SpeedBump.ManagedCPP.Form1.BubbleSortBtn_Click     0.0
SpeedBump.ManagedCPP.Form1.ClearTiming             0.0
SpeedBump.VBdotNet.Form1..ctor                     0.0
SpeedBump.VBdotNet.Form1.Dispose                   0.0
SpeedBump.VBdotNet.Form1.InitializeComponent       0.0
SpeedBump.VBdotNet.Form1.DoRandomize               0.0
SpeedBump.VBdotNet.Form1.UpdateSlot                0.0
SpeedBump.VBdotNet.Form1.UpdateAll                 0.0
SpeedBump.VBdotNet.Form1.RandomizeBtn_Click        0.0
                                                   0.0
                                                   0.0
tBtn_Click                                         0.0
oad                                                0.0
ng                                                 0.0
rtBtn_Click                                        0.0
ng                                                 0.0
g                                                  0.0
Btn_Click                                          0.0
g                                                  0.0
_Click                                             0.0
Click                                              0.0
se                                                 60.0
                                                   60.0
```

```
125 of 142 lines executed (88.0%)
13 of 15 methods called (86.7%)

Count   Source
1               }
                private void UpdateSlot(Int32 iSlot)
                {
2,174               slate.DrawLine(blackPen, 0, iSlot, Elements[iSl
2,174               if (Elements[iSlot] < numElems)
2,174                   slate.DrawLine(whitePen, Elements[iSlot] +
2,174           }

                private void UpdateAll()
                {
1                   for (Int32 i = 0; i<numElems; i++)
300                     UpdateSlot(i);
1                   bNeedUpdate = false;
1               }

                private void RandomizeBtn_Click(object sender, Syst
                {
1                   DoRandomize();
1                   UpdateAll();
1               }
```

```
125 of 142 lines executed (88.0%)
13 of 15 methods called (86.7%)

DevPartner - Coverage Analysis Session Summary

Started:                        5/23/2003 12:36:08 PM
Ended:                          5/23/2003 12:36:26 PM

Executable:                     C:\Program Files\Compuware\DevPartner Studio\Examples\SpeedBump.NET\Bin\Driv
Command Args:
Exit Code:                      0

Processor:                      Intel Pentium  647 Mhz
# of Processors:                1
OS Version:                     Microsoft Windows 2000


All Images

Percent of Lines Executed:      35.6
Number of Lines:                537
Number of Lines Executed:       191
Number of Lines Not Executed:   346
Percent of Methods Called:      34.5
Number of Methods:              55
Number of Methods Called:       19
```

Performance Analysis

Performance Analysis Session Data



Method List | Source [managedcpp.h] | Session Summary

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
SpeedBump.CSharp.Form1..ctor				
SpeedBump.Driver.Form1.InitializeComponent				
SpeedBump.Driver.Form1.ManagedCppBtn_Click				
SpeedBump.Driver.Form1.Main				
SpeedBump.CSharp.Form1.UpdateSlot				
SpeedBump.ManagedCPP.Form1.I...				
SpeedBump.Driver.Form1.CSharpBtn_Click				
SpeedBump.ManagedCPP.Form1....				
SpeedBump.CSharp.Form1.SwapEm				
SpeedBump.CSharp.Form1.QuickSortBtn_Click				
SpeedBump.ManagedCPP.Form1.QSort				
SpeedBump.CSharp.Form1.QSort				
SpeedBump.ManagedCPP.Form1.DoRandomize				
SpeedBump.ManagedCPP.Form1....				
SpeedBump.CSharp.Form1.RandomizeBtn_Click				

Method List | Source [managedcpp.h] | Session Summary

```
Count  % with Children      Time  Source
1             0.0            0.4              components = NULL;
1             0.4       86,039.0              InitializeComponent();
1             0.0            0.6              bNeedUpdate = false;
1             0.0            0.4          }

                                      /// <summary>
                                      /// Clean up any resources being used.
                                      /// </summary>
                                      protected:
                                          void Dispose( bool disposing )
                                          {
1             0.0            0.3              if( disposing )
                                              {
                                                  if (compo
                                                  {
                                                      compo
                                                  }
                                              }
                                              __super::Disp
                                          }

                                      private:
                                          void InitializeCo
```

Call Graph: SpeedBump.CSharp.F... 1.5 % → 95.0 % → SpeedBump.CSharp.F... 1.4 % (0.0 %) → 98.6 %; 3.1 % → System.Windows.For... 2.1 %; 0.2 %; 0.1 %; 0.0 % → SpeedBump.CSharp.F... 1.2 %; 0.0 %; 0.0 % → SpeedBump.CSharp.F... 2.9 %; 0.0 %

Method List | Source [managedcpp.h] | Session Summary

Compare Sessions 00:00:25 longer

Basis session: C:\Program Files\Compuware\DevPartner Studio\Examples\SpeedBump.NET\Bin\Driver1.dpprf

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	22,922.4
SpeedBump.Driver.Form1.ManagedCp...	**0.0**	**44.2**	**1**	**13,716.3**
- basis value	**0.1**	**75.1**	**1**	**13,747.7**
= difference	-	-	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

Method List | Source [managedcpp.h] | Session Summary

DevPartner - Performance Analysis Session Summary

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:	
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

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

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[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 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_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++)

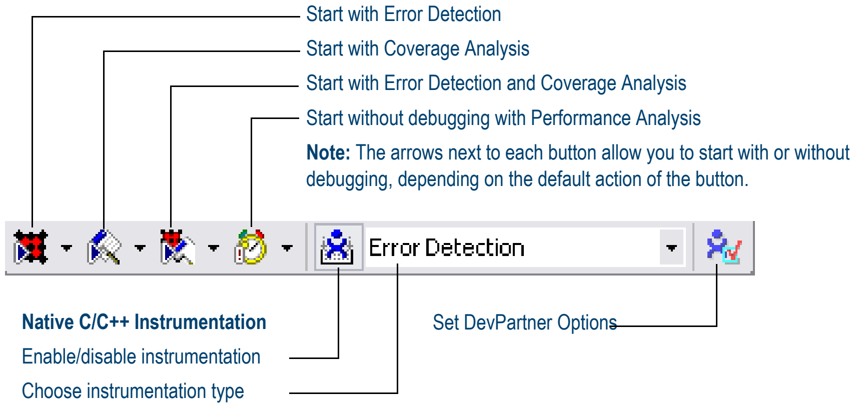
Default values vary slightly between Visual Studio 6.0 and Visual Studio .NET 2003 and 2005.

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

Category		Settings
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*

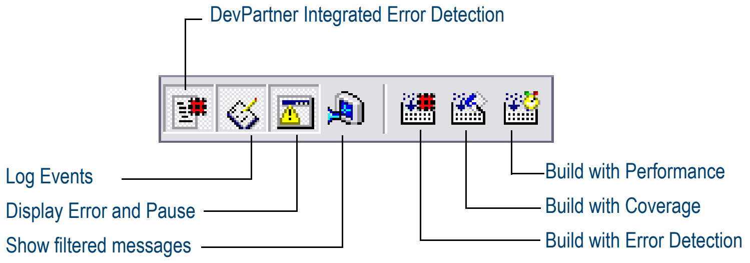
Category		Settings
Deadlock Analysis	Off	Enable deadlock analysis
	-	Assume single process - *Default when active: On*
	-	Enable watcher thread - *Default when active: Off*
	-	Generate errors when: A critical section is re-entered - *Default when active: Off*
	-	Generate errors when: A wait is requested 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


Start with Error Detection
Start with Coverage Analysis
Start with Error Detection and Coverage Analysis
Start without debugging with Performance Analysis
Error Detection
Native C/C++ Instrumentation
Enable/disable instrumentation
Choose instrumentation type
Set DevPartner Options

Note: The arrows next to each button allow you to start with or without debugging, depending on the default action of the button.

Error Detection Toolbar in Visual C++ 6.0



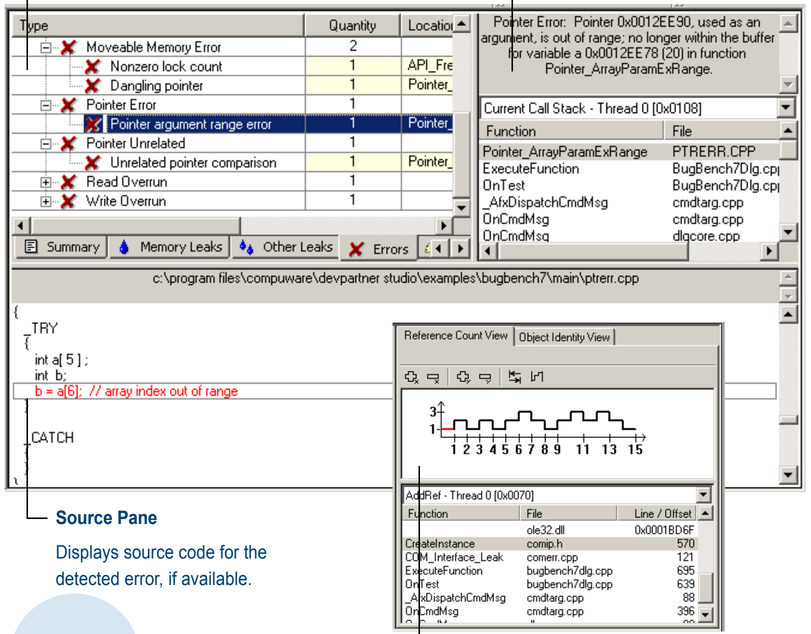
Error Detection Window

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).



Type	Quantity	Location
⊟ ✗ Moveable Memory Error	2	
✗ Nonzero lock count	1	API_Fre
✗ Dangling pointer	1	Pointer_
⊟ ✗ Pointer Error	1	
✗ Pointer argument range error	1	Pointer_
⊟ ✗ Pointer Unrelated	1	
✗ Unrelated pointer comparison	1	Pointer_
⊞ ✗ Read Overrun	1	
⊞ ✗ Write Overrun	1	

Pointer Error: Pointer 0x0012EE90, used as an argument, is out of range; no longer within the buffer for variable a 0x0012EE78 (20) in function Pointer_ArrayParamExRange.

Current Call Stack - Thread 0 [0x0108]

Function	File
Pointer_ArrayParamExRange	PTRERR.CPP
ExecuteFunction	BugBench7Dlg.cp
OnTest	BugBench7Dlg.cp
_AfxDispatchCmdMsg	cmdtarg.cpp
OnCmdMsg	cmdtarg.cpp
OnCmdMsg	dlgcore.cpp

c:\program files\compuware\devpartner studio\examples\bugbench7\main\ptrerr.cpp

```
{
  _TRY
  {
    int a[ 5 ] ;
    int b;
    b = a[6];  // array index out of range
  }
  _CATCH
```

AddRef - Thread 0 [0x0070]

Function	File	Line / Offset
	ole32.dll	0x0001BD6F
CreateInstance	comip.h	570
COM_Interface_Leak	comerr.cpp	121
ExecuteFunction	bugbench7dlg.cpp	695
OnTest	bugbench7dlg.cpp	639
_AfxDispatchCmdMsg	cmdtarg.cpp	88
OnCmdMsg	cmdtarg.cpp	396

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.

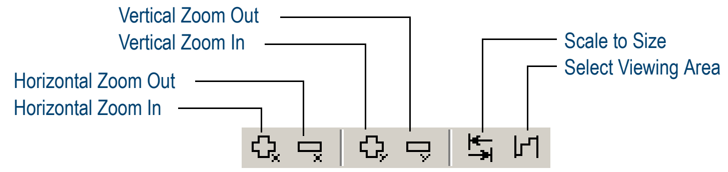
Icons Used in the Results Pane

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

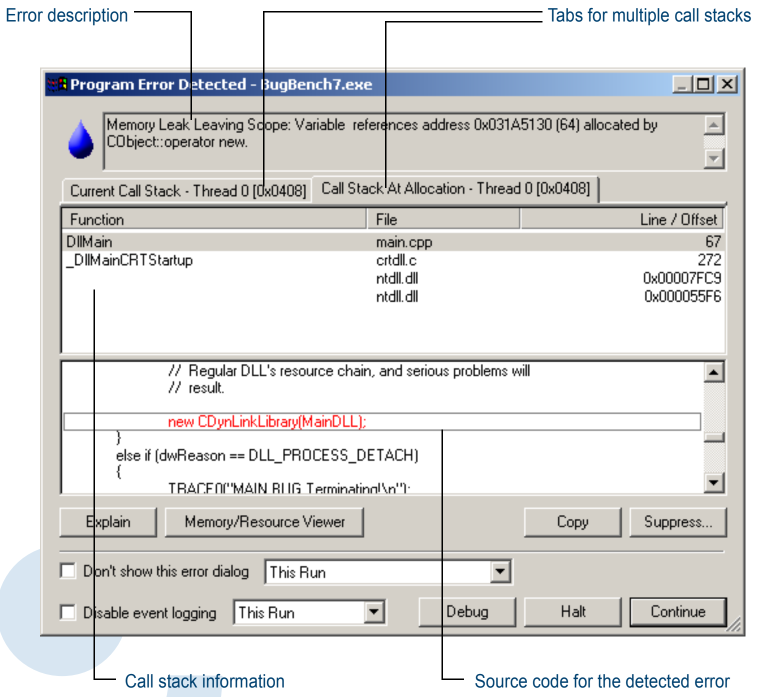
Icons Used in the Details Pane

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


Error description
Tabs for multiple call stacks
Call stack information
Source code for the detected error

Program Error Detected - BugBench7.exe

Memory Leak Leaving Scope: Variable references address 0x031A5130 (64) allocated by CObject::operator new.

Current Call Stack - Thread 0 [0x0408] | Call Stack At Allocation - Thread 0 [0x0408]

Function	File	Line / Offset
DllMain	main.cpp	67
_DllMainCRTStartup	crtdll.c	272
	ntdll.dll	0x00007FC9
	ntdll.dll	0x000055F6

```
                // Regular DLL's resource chain, and serious problems will
                // result.

                new CDynLinkLibrary(MainDLL);
        }
        else if (dwReason == DLL_PROCESS_DETACH)
        {
                TRACE0("MAIN BUG Terminating!\n");
```

Explain | Memory/Resource Viewer | Copy | Suppress...

Don't show this error dialog: This Run

Disable event logging: This Run

Debug | Halt | Continue

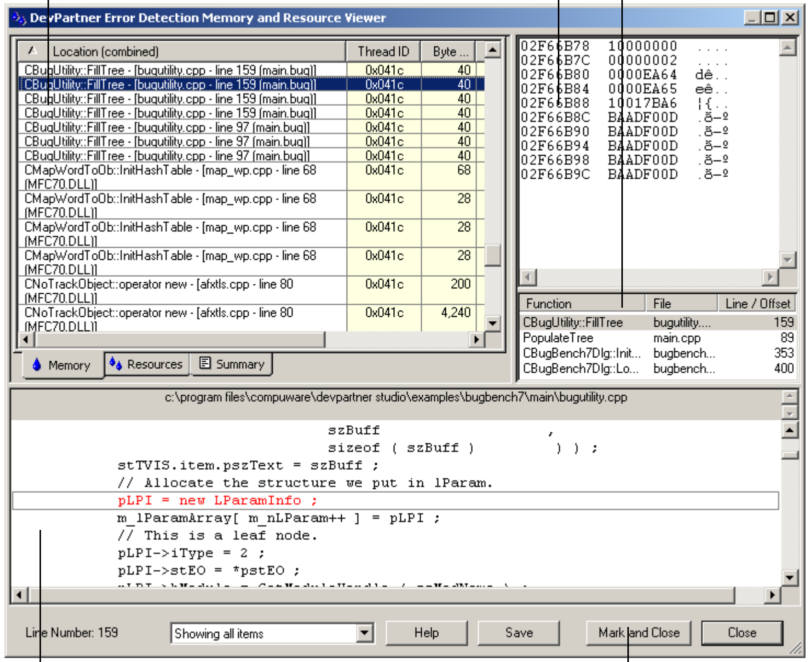
Memory and Resource Viewer Dialog Box

Results Pane

Displays Memory, Resource, and Summary tabs

Memory Contents Pane

Stack Pane



DevPartner Error Detection Memory and Resource Viewer

Location (combined)	Thread ID	Byte ...
CBugUtility::FillTree - [bugutility.cpp - line 159 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 159 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 159 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 159 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)]	0x041c	40
CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)]	0x041c	40
CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 (MFC70.DLL)]	0x041c	68
CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 (MFC70.DLL)]	0x041c	28
CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 (MFC70.DLL)]	0x041c	28
CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 (MFC70.DLL)]	0x041c	28
CNoTrackObject::operator new - [afxtls.cpp - line 80 (MFC70.DLL)]	0x041c	200
CNoTrackObject::operator new - [afxtls.cpp - line 80 (MFC70.DLL)]	0x041c	4,240

```
02F66B78  10000000  ....
02F66B7C  00000002  ....
02F66B80  0000EA64  dê..
02F66B84  0000EA65  eê..
02F66B88  10017BA6  ¦{..
02F66B8C  BAADF00D  .ð–º
02F66B90  BAADF00D  .ð–º
02F66B94  BAADF00D  .ð–º
02F66B98  BAADF00D  .ð–º
02F66B9C  BAADF00D  .ð–º
```

Function	File	Line / Offset
CBugUtility::FillTree	bugutility....	159
PopulateTree	main.cpp	89
CBugBench7Dlg::Init...	bugbench...	353
CBugBench7Dlg::Lo...	bugbench...	400

c:\program files\compuware\devpartner studio\examples\bugbench7\main\bugutility.cpp

```
                                szBuff                   ,
                                sizeof ( szBuff )         ) ) ;
            stTVIS.item.pszText = szBuff ;
            // Allocate the structure we put in lParam.
            pLPI = new LParamInfo ;
            m_lParamArray[ m_nLParam++ ] = pLPI ;
            // This is a leaf node.
            pLPI->iType = 2 ;
            pLPI->stEO = *pstEO ;
```

Line Number: 159 | Showing all items

Source Pane

Displays source code for the detected error, if available.

Mark and Close

Click to mark existing allocations and close the dialog box. Marked items will not be shown when Memory and Resource viewer reappears.

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), 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
`/out[put]=<String>`	Specify the 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.
`/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.