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
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.

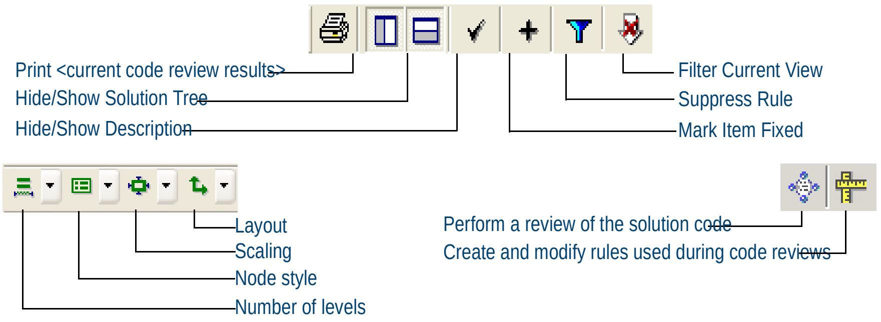
Code Review Default Options (General Node)

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

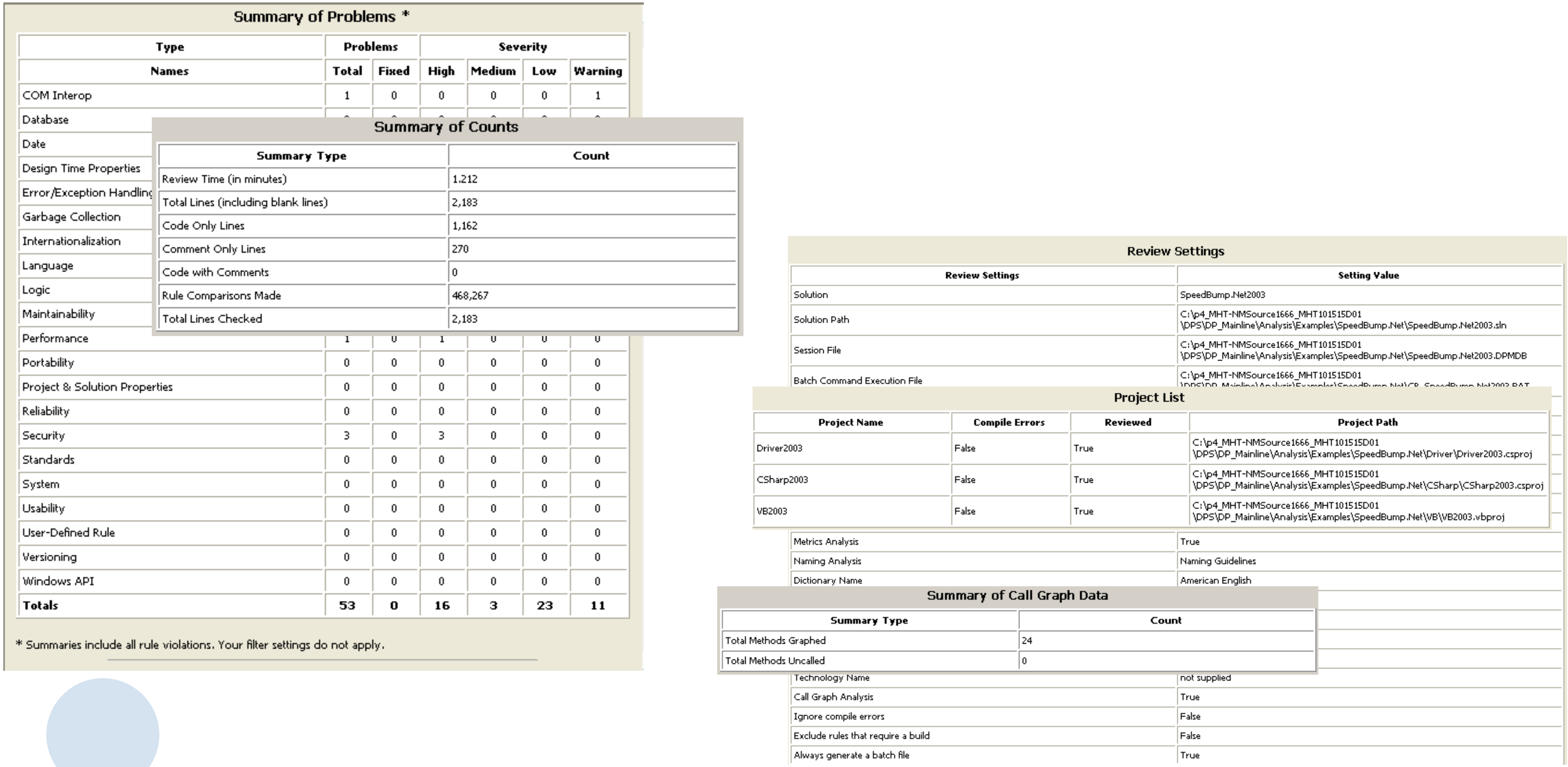
Naming Guidelines

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

Code Review Toolbars



Code Review Summaries



Summary of Problems *

Type	Problems		Severity			
Names	Total	Fixed	High	Medium	Low	Warning
COM Interop	1	0	0	0	0	1
Database						
Date						
Design Time Properties						
Error/Exception Handling						
Garbage Collection						
Internationalization						
Language						
Logic						
Maintainability						
Performance	1	0	1	0	0	0
Portability	0	0	0	0	0	0
Project & Solution Properties	0	0	0	0	0	0
Reliability	0	0	0	0	0	0
Security	3	0	3	0	0	0
Standards	0	0	0	0	0	0
System	0	0	0	0	0	0
Usability	0	0	0	0	0	0
User-Defined Rule	0	0	0	0	0	0
Versioning	0	0	0	0	0	0
Windows API	0	0	0	0	0	0
Totals	**53**	**0**	**16**	**3**	**23**	**11**

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

Summary of Counts

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

Review Settings

Review Settings	Setting Value
Solution	SpeedBump.Net2003
Solution Path	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.sln
Session File	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\SpeedBump.Net2003.DPMDB
Batch Command Execution File	C:\p4_MHT-NMSource1666_MHT101515D01 \DPS\DP_Mainline\Analysis\Examples\SpeedBump.Net\CR_SpeedBump.Net2003.BAT
Metrics Analysis	True
Naming Analysis	Naming Guidelines
Dictionary Name	American English
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

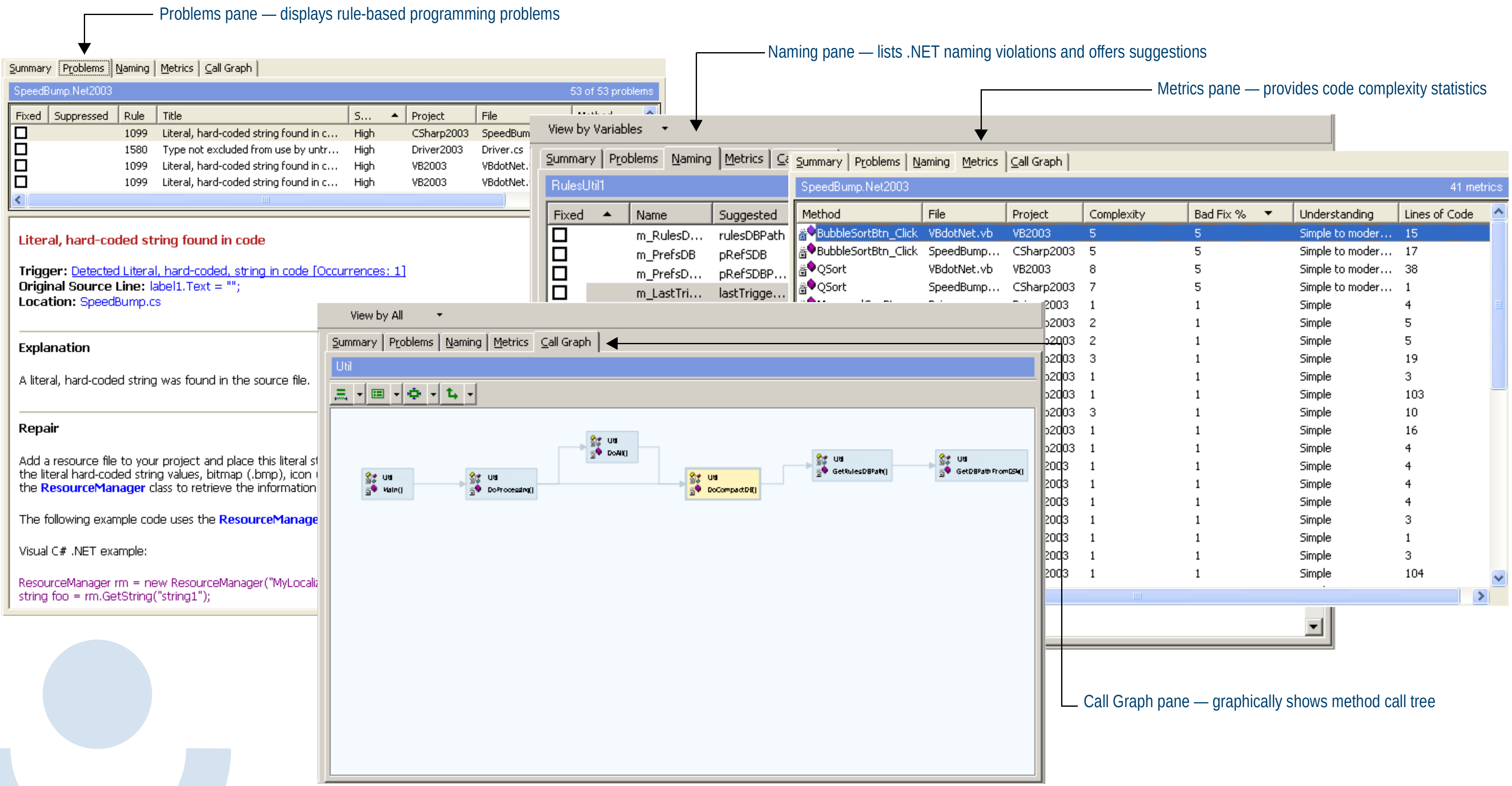
Project List

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

Summary of Call Graph Data

Summary Type	Count
Total Methods Graphed	24
Total Methods Uncalled	0

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

Call Graph pane — graphically shows method call tree

SpeedBump.Net2003 — 53 of 53 problems

Fixed	Suppressed	Rule	Title	S...	Project	File
□		1099	Literal, hard-coded string found in c...	High	CSharp2003	SpeedBum
□		1580	Type not excluded from use by untr...	High	Driver2003	Driver.cs
□		1099	Literal, hard-coded string found in c...	High	VB2003	VBdotNet.
□		1099	Literal, hard-coded string found in c...	High	VB2003	VBdotNet.

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 st
the literal hard-coded string values, bitmap (.bmp), icon (
the **ResourceManager** class to retrieve the information

The following example code uses the **ResourceManage**

Visual C# .NET example:

```
ResourceManager rm = new ResourceManager("MyLocaliz
string foo = rm.GetString("string1");
```

View by Variables

RulesUtil1

Fixed	Name	Suggested
□	m_RulesD...	rulesDBPath
□	m_PrefsDB	pRefSDB
□	m_PrefsD...	pRefSDBP...
□	m_LastTri...	lastTrigge...

SpeedBump.Net2003 — 41 metrics

Method	File	Project	Complexity	Bad Fix %	Understanding	Lines of Code
BubbleSortBtn_Click	VBdotNet.vb	VB2003	5	5	Simple to moder...	15
BubbleSortBtn_Click	SpeedBump...	CSharp2003	5	5	Simple to moder...	17
QSort	VBdotNet.vb	VB2003	8	5	Simple to moder...	38
QSort	SpeedBump...	CSharp2003	7	5	Simple to moder...	1
		2003	1	1	Simple	4
		b2003	2	1	Simple	5
		b2003	2	1	Simple	5
		b2003	3	1	Simple	19
		b2003	1	1	Simple	3
		b2003	1	1	Simple	103
		b2003	3	1	Simple	10
		b2003	1	1	Simple	16
		b2003	1	1	Simple	4
		2003	1	1	Simple	4
		2003	1	1	Simple	4
		2003	1	1	Simple	4
		2003	1	1	Simple	3
		2003	1	1	Simple	1
		2003	1	1	Simple	3
		2003	1	1	Simple	104

View by All

Util

Coverage, Memory, and Performance Analysis

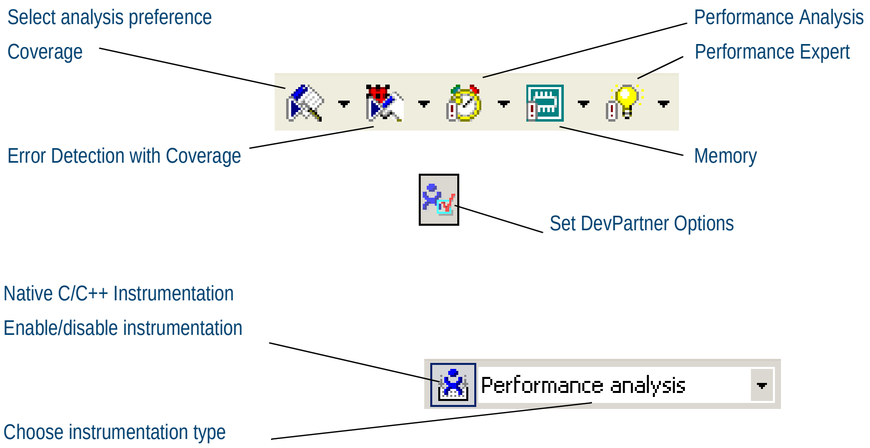
Determine application test coverage, analyze an application’s use of memory, and profile application performance.

General and Data Collection Properties

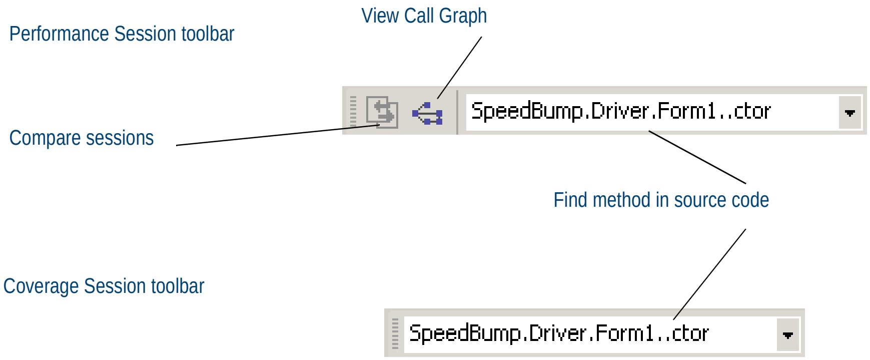
The following data collection properties apply to Performance, Coverage, and Memory analysis.

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
Performance Analysis
Performance Expert
Error Detection with Coverage
Memory
Set DevPartner Options
Native C/C++ Instrumentation
Enable/disable instrumentation
Performance analysis
Choose instrumentation type

Performance and Coverage Analysis Session Toolbars


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

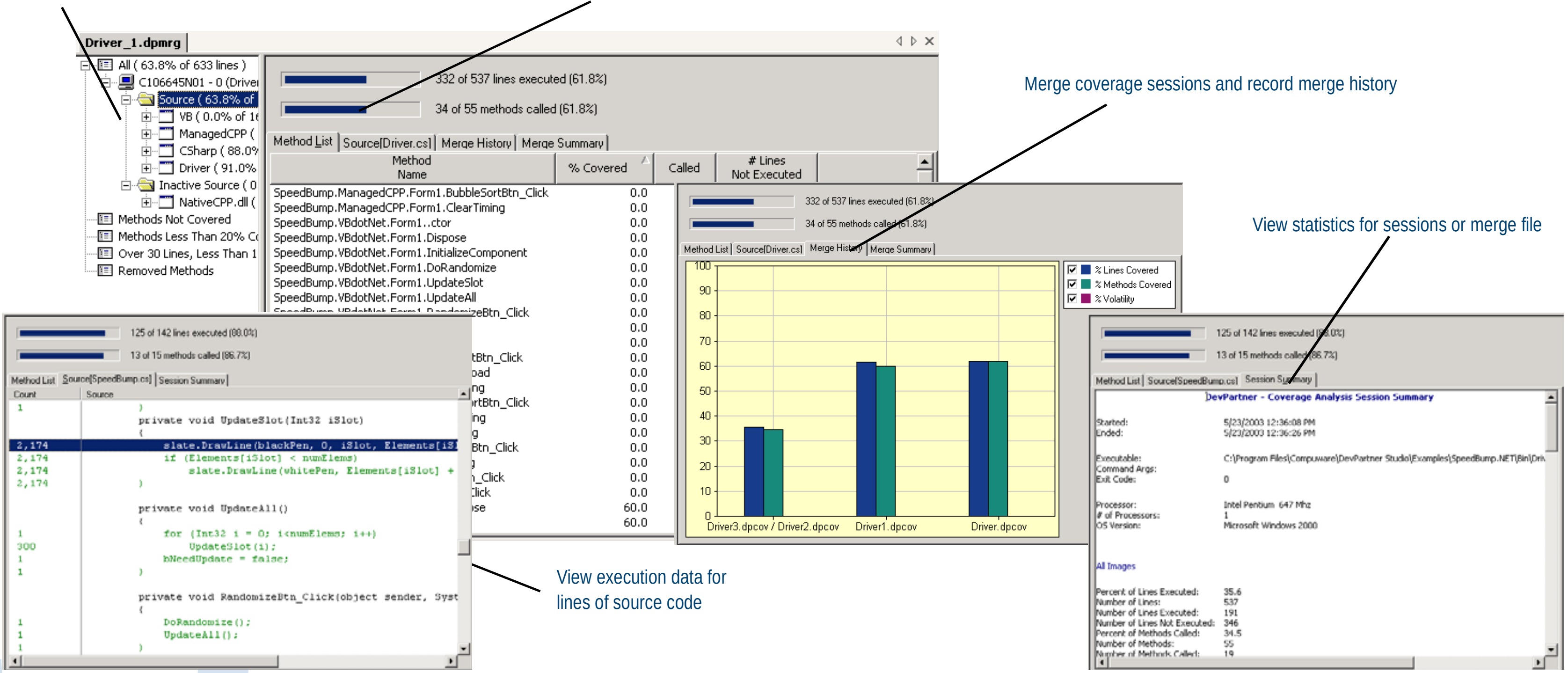
Coverage Analysis

Coverage Analysis Session Data

Results Summaries

DevPartner displays results for Coverage Analysis in session files. 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
```

Filter the data view

View coverage metrics for methods

Merge coverage sessions and record merge history

View statistics for sessions or merge file

View execution data for lines of source code

Memory Analysis

Session Control for memory analysis

Memory Leaks session controls

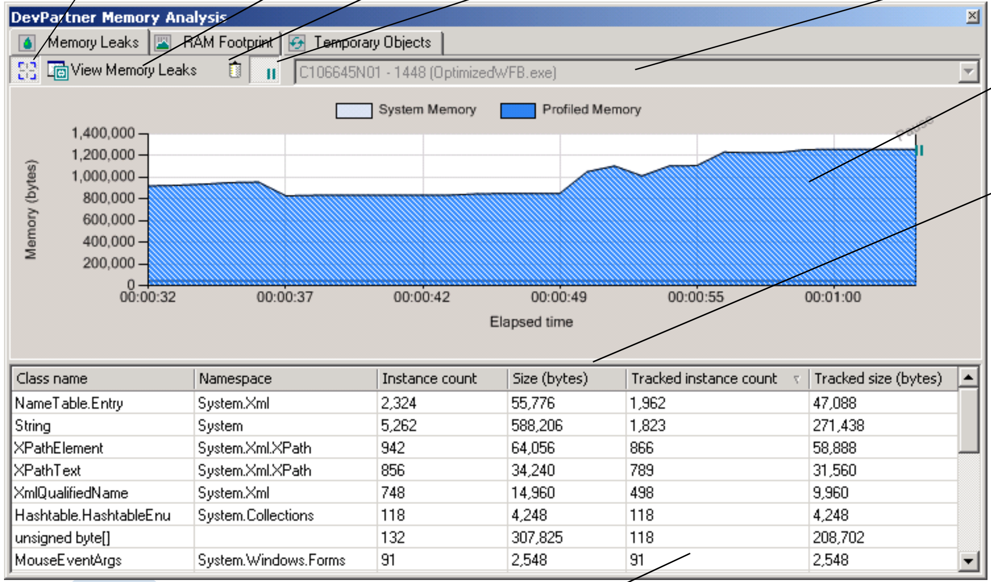
Start and Stop Tracking potential memory leaks

Take a memory leaks snapshot

Force a garbage collection

Pause real-time graph (data collection continues)

Choose process to profile



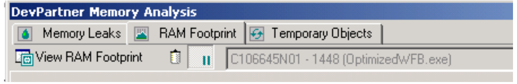
Class name	Namespace	Instance count	Size (bytes)	Tracked instance count	Tracked size (bytes)
NameTable.Entry	System.Xml	2,324	55,776	1,962	47,088
String	System	5,262	588,206	1,823	271,438
XPathElement	System.Xml.XPath	942	64,056	866	58,888
XPathText	System.Xml.XPath	856	34,240	789	31,560
XmlQualifiedName	System.Xml	748	14,960	498	9,960
Hashtable.HashtableEnu	System.Collections	118	4,248	118	4,248
unsigned byte[]		132	307,825	118	208,702
MouseEventArgs	System.Windows.Forms	91	2,548	91	2,548

Graph shows state of managed heap in real time

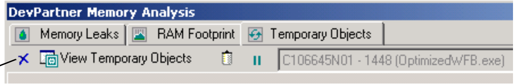
Class list dynamically updates to show what is in memory

Session controls tailored to type of memory data collection

RAM Footprint session controls:


DevPartner Memory Analysis
Memory Leaks
RAM Footprint
Temporary Objects
View RAM Footprint
C106645N01 - 1448 (OptimizedWFB.exe)

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

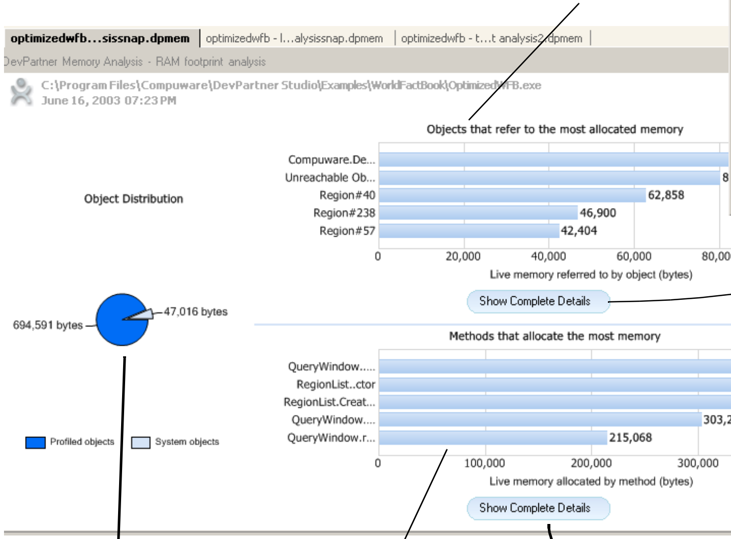
Memory Analysis Session Data

Results tailored to type data collection

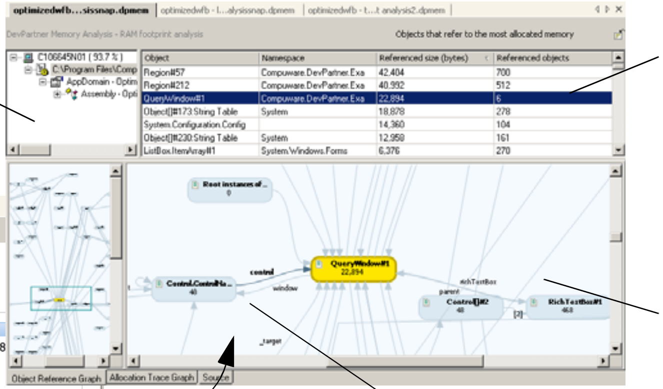
- RAM Footprint
- Memory Leaks
- Temporary Objects
- Click Show Complete Details to view session data

Analyze object allocations in depth

Summary of most memory-intensive object allocations



optimizedwfb…sissnap.dpmem
optimizedwfb - l…alysissnap.dpmem
optimizedwfb - t…t analysis2.dpmem
DevPartner Memory Analysis - RAM footprint analysis
C:\Program Files\Compuware\DevPartner Studio\Examples\WorldFactBook\OptimizedWFB.exe
June 16, 2003 07:23 PM
Object Distribution
694,591 bytes
47,016 bytes
Profiled objects
System objects
Objects that refer to the most allocated memory
Compuware.De...
Unreachable Ob...
Region#40
Region#238
Region#57
62,858
46,900
42,404
0
20,000
40,000
60,000
80,000
100,000
Live memory referred to by object (bytes)
Show Complete Details
Methods that allocate the most memory
QueryWindow.....
RegionList..ctor
RegionList.Creat...
QueryWindow....
QueryWindow.r...
391,295
384,817
303,296
215,068
0
100,000
200,000
300,000
Live memory allocated by method (bytes)
Show Complete Details



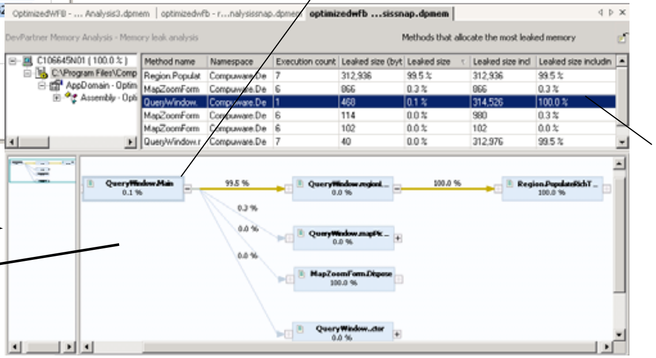
Object	Namespace	Referenced size (bytes)	Referenced objects
Region#57	Compuware.DevPartner.Exa	42,404	700
Region#212	Compuware.DevPartner.Exa	40,992	512
QueryWindow#1	Compuware.DevPartner.Exa	22,894	6
Object[]#173:String Table	System	18,878	278
System.Configuration.Config		14,360	104
Object[]#230:String Table	System	12,958	161
ListBox.ItemArray#1	System.Windows.Forms	6,376	270

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



Method name	Namespace	Execution count	Leaked size (byt	Leaked size	Leaked size incl	Leaked size includin
Region.Populat	Compuware.De	7	312,936	99.5 %	312,936	99.5 %
MapZoomForm	Compuware.De	6	866	0.3 %	866	0.3 %
QueryWindow.	Compuware.De	1	468	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.r	Compuware.De	7	40	0.0 %	312,976	99.5 %

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

Summary of most memory-intensive methods

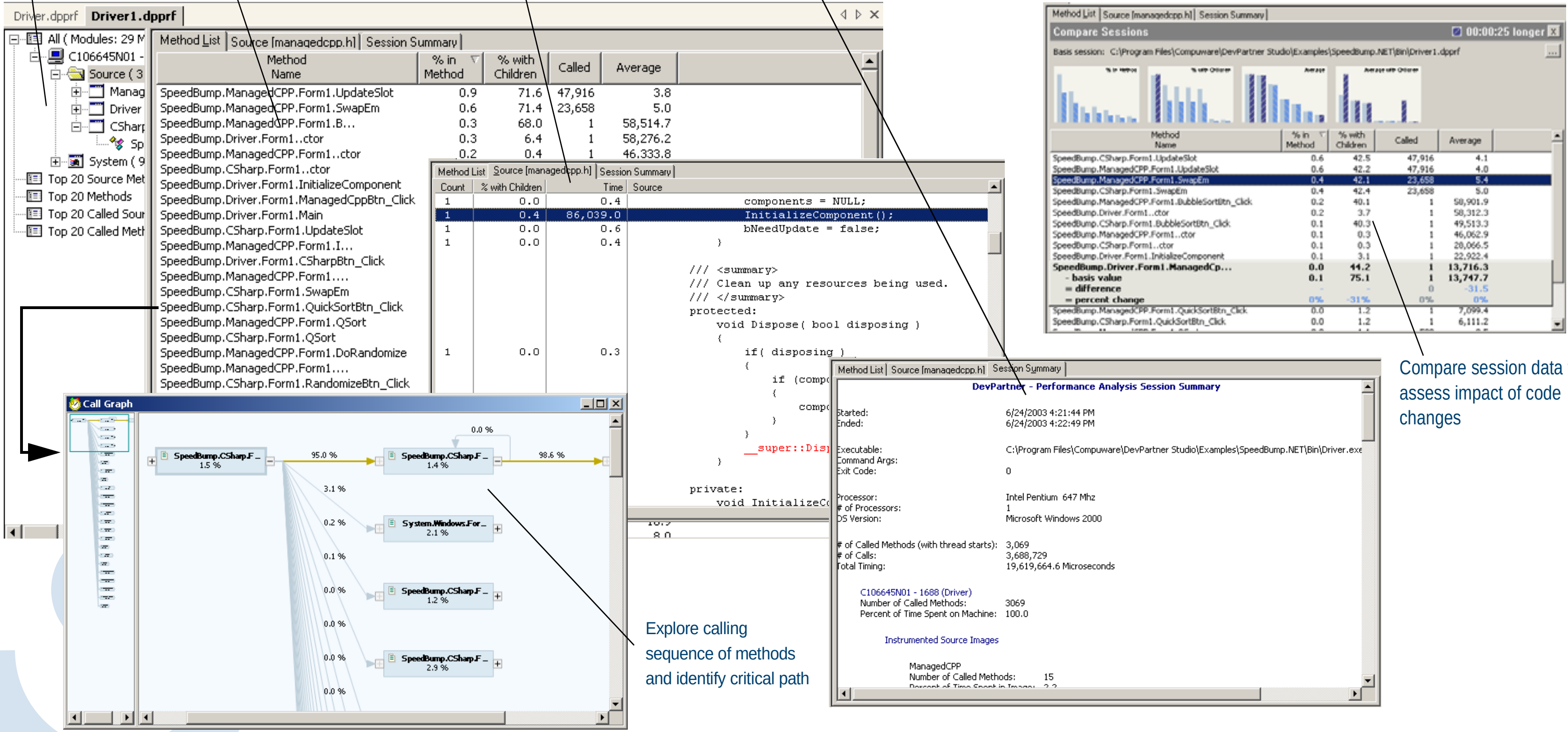
Object distribution: user vs. system objects (RAM footprint only)

Call Graph

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

Performance Analysis

Performance Analysis Session Data



Filter the data view

View performance metrics for methods

Locate methods in source code

View session statistics

Explore calling sequence of methods and identify critical path

Compare session data to assess impact of code changes

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				

```
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
```

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
```

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

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

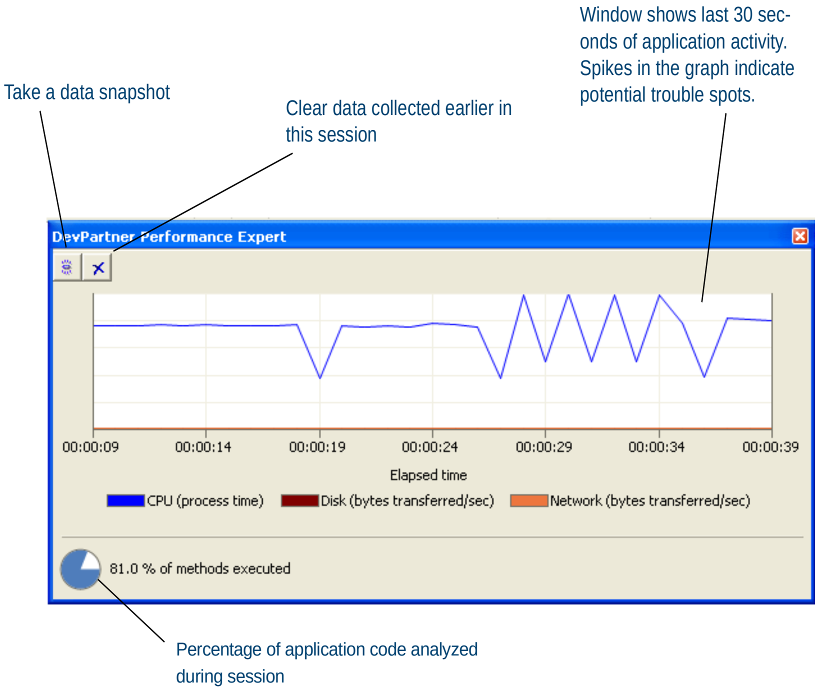
Performance Expert

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

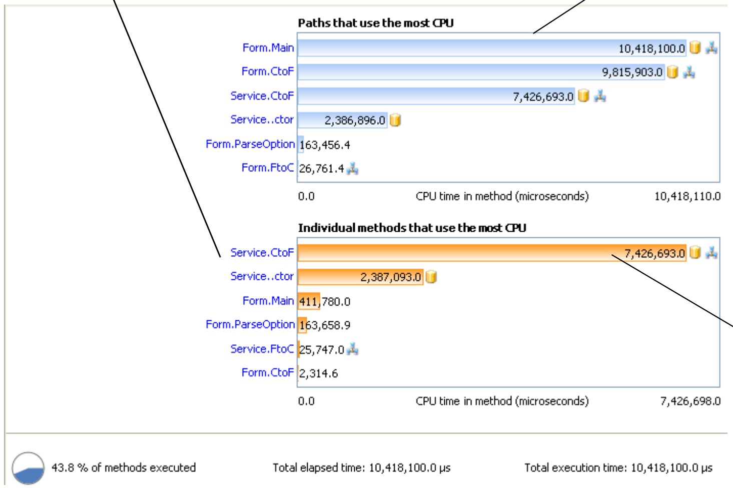
Performance Expert Session Controls


Take a data snapshot
Clear data collected earlier in this session
Window shows last 30 seconds of application activity. Spikes in the graph indicate potential trouble spots.
DevPartner Performance Expert
00:00:09
00:00:14
00:00:19
00:00:24
00:00:29
00:00:34
00:00:39
Elapsed time
CPU (process time)
Disk (bytes transferred/sec)
Network (bytes transferred/sec)
81.0 % of methods executed
Percentage of application code analyzed during session

Performance Expert Session Data

Click individual method for Methods analysis (without children)

Click entry point method for Path analysis (with children)



Paths that use the most CPU

Method	CPU time in method (microseconds)
Form.Main	10,418,100.0
Form.CtoF	9,815,903.0
Service.CtoF	7,426,693.0
Service..ctor	2,386,896.0
Form.ParseOption	163,456.4
Form.FtoC	26,761.4

Individual methods that use the most CPU

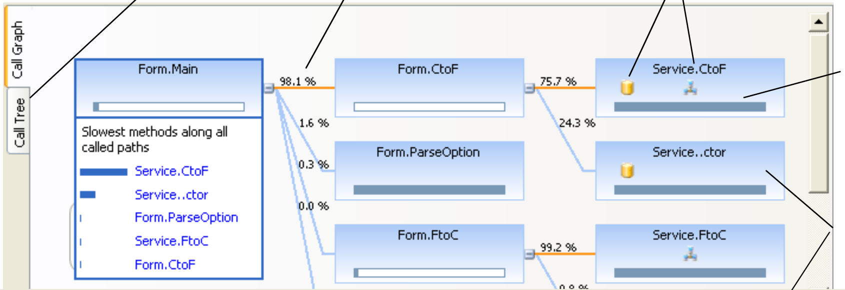
Method	CPU time in method (microseconds)
Service.CtoF	7,426,693.0
Service..ctor	2,387,093.0
Form.Main	411,780.0
Form.ParseOption	163,658.9
Service.FtoC	25,747.0
Form.CtoF	2,314.6

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

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

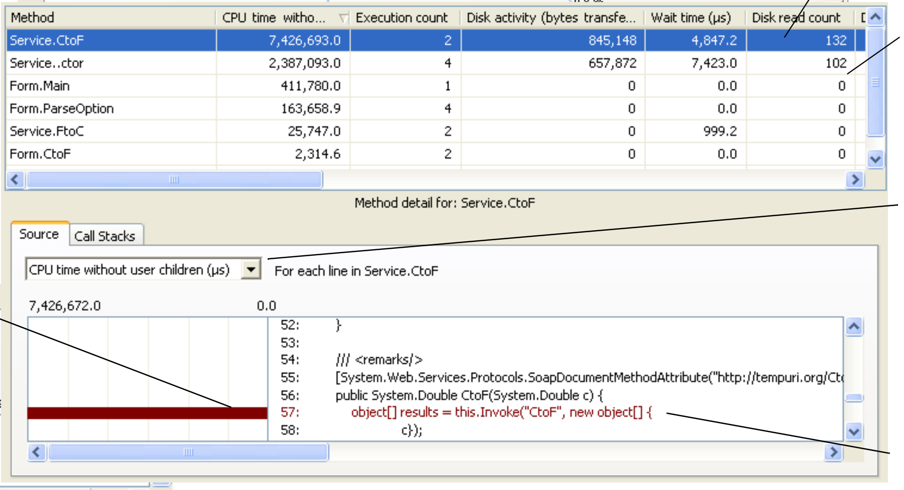


Slowest methods along all called paths

- Service.CtoF
- Service..ctor
- Form.ParseOption
- Service.FtoC
- Form.CtoF

Bars show time in method vs. time in child methods

Select a method to update source and call stack tabs



Method	CPU time witho...	Execution count	Disk activity (bytes transfe...	Wait time (µs)	Disk read count
Service.CtoF	7,426,693.0	2	845,148	4,847.2	132
Service..ctor	2,387,093.0	4	657,872	7,423.0	102
Form.Main	411,780.0	1	0	0.0	0
Form.ParseOption	163,658.9	4	0	0.0	0
Service.FtoC	25,747.0	2	0	999.2	0
Form.CtoF	2,314.6	2	0	0.0	0

Method detail for: Service.CtoF

CPU time without user children (µs) For each line in Service.CtoF

```
52:     }
53:
54:     /// <remarks/>
55:     [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Ct
56:     public System.Double CtoF(System.Double c) {
57:         object[] results = this.Invoke("CtoF", new object[] {
58:                     c});
```

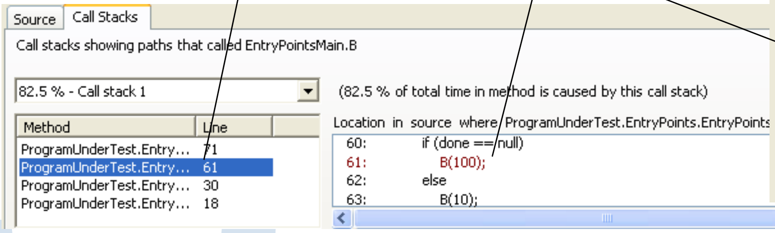
Methods table shows impact of disk, network I/O, and wait time

Choose metric

Source tab shows most expensive line with metrics

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



Call stacks showing paths that called EntryPointsMain.B

82.5 % - Call stack 1 (82.5 % of total time in method is caused by this call stack)

Method	Line
ProgramUnderTest.Entry...	71
ProgramUnderTest.Entry...	61
ProgramUnderTest.Entry...	30
ProgramUnderTest.Entry...	18

Location in source where ProgramUnderTest.EntryPoints.EntryPoints

```
60:         if (done == null)
61:             B(100);
62:         else
63:             B(10);
```

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
	/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 (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[ocess] - Specify a target process (followed by arguments to process)
	/S[ervice] - Specify a target service (followed by arguments to service)
	/C[onfig] - Path to configuration file

Configuration File

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

```
DPAnalysis /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. *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 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 performance and maximum session time.
Service	The configuration file must contain at least one Process or one Service element. Specifies a target service. *Attributes:* CollectData, Start, RestartIfRunning, RestartAtEndOfRun
Targets	Required. One only. Begins a block of one or more Process or Service entries. Target processes and services are started in the order they are listed in the configuration file. *Attributes:* RunInParallel

Error Detection

File Extensions Used by Error Detection

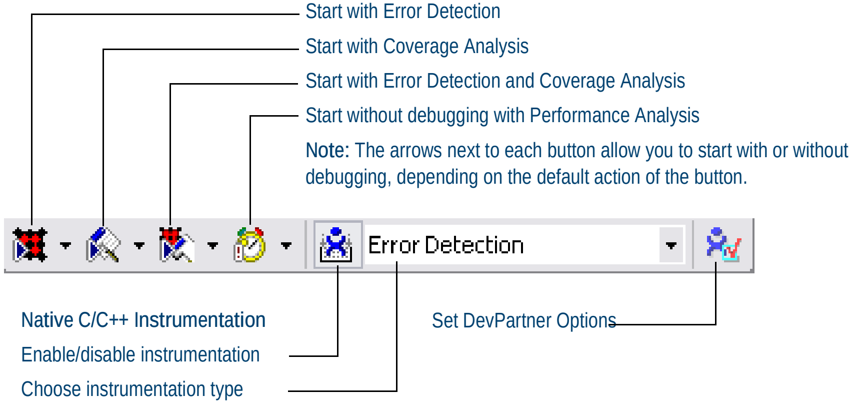
Extension	File Type	Description
.dpbcl	Error Detection Session File	This is the Error Detection log for the user's program execution.
.dpbcc .dpbcd	Error Detection Settings File	This file contains the various settings for Error Detection. The .dpbcd extension refers to the default settings file created,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

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

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

Category		Settings
Deadlock Analysis	Off	Enable deadlock analysis
	-	Assume single process - *Default when active: On*
	-	Enable watcher thread - *Default when active: Off*
	-	Generate errors when: A critical section is re-entered - *Default when active: Off*
	-	Generate errors when: A wait is requested on an owned mutex - *Default when active: Off*
	-	Number of historical events per resource - *Default when active: 10*
	-	Report synchronization API timeouts - *Default when active: Off*
	-	Report wait limits or actual waits exceeding (seconds) - *Default when active: 60*
	-	Synchronization Naming Rules - *Default when active: Don't warn about resource naming*
Memory Tracking	On	Enable memory tracking
	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 - *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

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

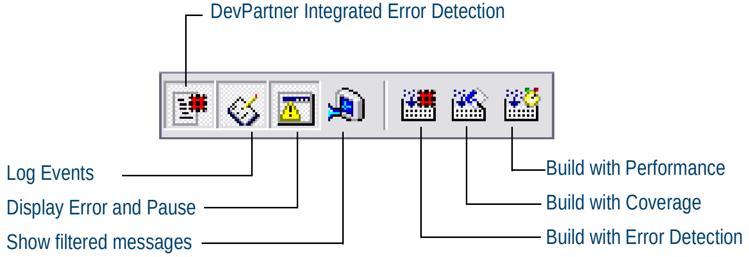
Native C/C++ Instrumentation

Enable/disable instrumentation

Choose instrumentation type

Set DevPartner Options

Error Detection Toolbar in Visual C++ 6.0



DevPartner Integrated Error Detection

Log Events

Display Error and Pause

Show filtered messages

Build with Performance

Build with Coverage

Build with Error Detection

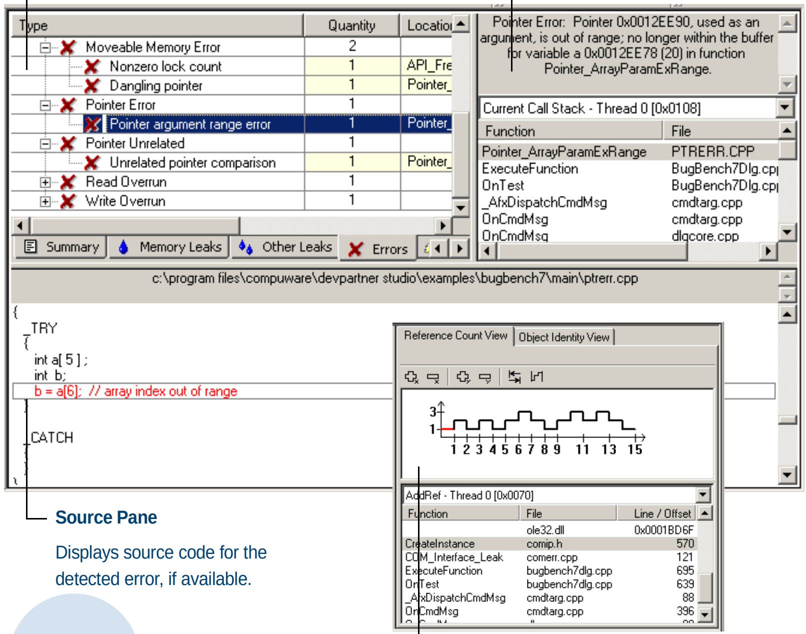
Error Detection Window

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.cpp
OnTest	BugBench7Dlg.cpp
_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.

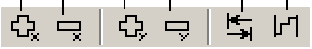
Icons Used in the Results Pane

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



Vertical Zoom Out

Vertical Zoom In

Horizontal Zoom Out

Horizontal Zoom In

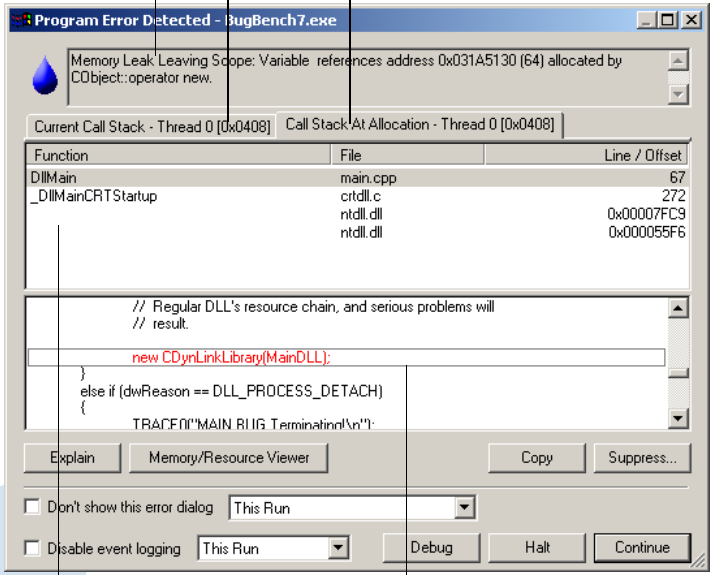
Scale to Size

Select Viewing Area

Program Error Detected Dialog Box

Error description

Tabs for multiple call stacks



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");
```

□ Don't show this error dialog: This Run

□ Disable event logging: This Run

Call stack information

Source code for the detected error

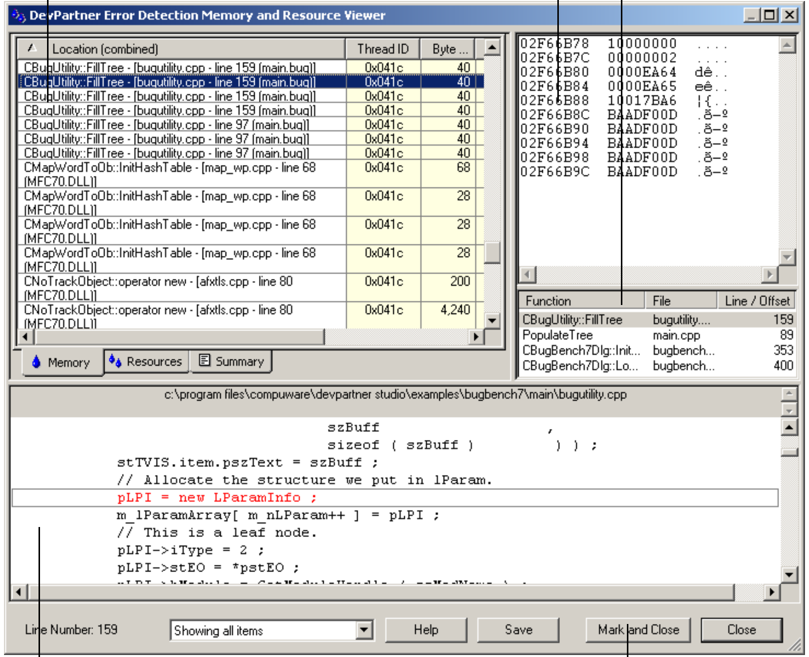
Memory and Resource Viewer Dialog Box

Results Pane

Displays Memory, Resource, and Summary tabs

Memory Contents Pane

Stack Pane



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, 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.

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) files to XML files.

Note: DevPartner.Analysis.DataExport.exe is the command line executable for the Export DevPartner Data command, a separately licensed feature.

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

Options

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

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.
`/r[ecurse]`	Search subdirectories for DevPartner Session Files.
`/showAll`	Show all session file data regardless of analysis type. For example, if you export a coverage session file with this option, the resulting XML file will contain both coverage 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.