DPVC Quick Reference

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

				-	Check for COM "Not Implemented" return code - Default when active: On
Category		Settings		-	API failure codes - Default when active: On
General	On	Log events		-	Check invalid parameter errors: API, COM - Default when active: both On
	On	Display error and pause		-	Category: Handle and pointer arguments - Default when active: On
	Off	Prompt to save program results		-	Category: Flag, range and enumeration arguments - Default when
	Off	Show memory and resource viewer when application exits			active: On
	On	Source file search path - based on the location of the .EXE (standalone), .DSW (Visual C++), or .SLN (Visual Studio .NET).		-	Check statically linked C run-time library APIs - <i>Default when active: On</i>
	-	Override symbol path - Default: empty			when active: All items selected
	-	Working directory (standalone only) based on the location of the .EXE	COM Call Reporting	Off	Enable COM method call reporting on objects that are implemented in
	-	Command line arguments (standalone only) - Default: empty			the selected modules
Data Collection	On	Call parameter coding depth = 1		-	Report COM method calls on objects implemented outside of the listed modules - <i>Default when active: On</i>
	On	Maximum call stack depth on allocation = 5		-	All components tree view - Default when active: All selected
	On On	Maximum call stack depth on error = 20	COM Object Tracking	Off	Enable COM object tracking
		NLB file directory is based on the location of the .EXE (standalone), .DSW (Visual C++), or .SLN (Visual Studio .NET).	com object maxing	-	All COM classes tree view - Default when active: All selected
	Off	Generate NLB files dynamically	Deadlock Analysis	Off	Enable deadlock analysis
API Call Reporting	Off	Enable API call reporting. All category selections are unavailable until you		-	Assume single process - Default when active: On
		check this item.		-	Enable watcher thread - Default when active: Off
	-	Collect window messages - Default when active: Off		-	Generate errors when: A critical section is re-entered - Default when
	-	Collect API method calls and returns Default when active: On			active: Off
	-	View only modules needed by this application - Default when active: On		-	Generate errors when: A wait is requrested on an owned mutex -
	-	All modules (tree view) Default when active: All selected			Default when active: On
Call Validation	Off	Off Enable call validation. All category selections are unavailable until you		-	Number of historical events per resource - <i>Default when active: 10</i>
		check this item		-	Report synchronization API timeouts - Default when active: Off
	-	Enable memory block checking - Default when active: Off		-	Report wait limits or actual waits exceeding (seconds) - Default when active: 60
	-	Fill output argument before call - Default when active: Off		-	Synchronization Naming Rules - Default when active: Don't warn about
	-	COM failure codes - Default when active: On		-	resource naming

Category

Settings

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

Category		Settings	Category		Settings
Memory Tracking	On	Enable memory tracking		-	Finalizer monitoring - Default when active: On
	On	Report leaks immediately		-	COM interop monitoring - Default when active: On
	Off	Show leaked allocation blocks		-	PInvoke interop monitoring - Default when active: On
	Off	Enforce strict reallocation semantics		-	Interop reporting threshold - Default when active: 1
	On	Enable FinalCheck	.NET Call Reporting	Off	Enable .NET method call reporting
	On	Enable guard bytes; Pattern = FC; Count = 4 bytes		-	All types (tree view node) - Default when active: Selected
	-	Check heap blocks at runtime: On free		-	.NET User Assemblies (tree view node) - Default when active: Selected
	On	Enable fill on allocation; Pattern = FB		-	.NET System Assemblies (tree view node) - Default when active: Not
	On	Check uninitialized memory; Size = 2 bytes			selected
	On	Enable poison on free; Pattern = FD	Resource Tracking	On	Enable resource tracking
.NET Analysis	Off	Enable .NET analysis		On	Resources tree view. All listed resources are selected by default
	-	Exception monitoring - Default when active: On	Modules and Files	On	Modules and files tree view. All listed modules are selected by default.
				Off	Show leaks and errors only if source code is available

BoundsChecker User Interface

BoundsChecker Window

Results Pane Details Pane Summary, Memory Leaks, Other Leaks, Errors, .NET Displays long description of detected Performance, Modules, Transcript tabs provide overview error; call stack information; reference and detail about detected errors. count graph (see inset below). Pointer Error: Pointer 0x0012EE90, used as an Quantity Locatio: 🔺 ype argument, is out of range; no longer within the buffer - X Moveable Memory Error 2 for variable a 0x0012EE78 (20) in function 🗶 Nonzero lock count API Fre 1 Pointer_ArrayParamExRange. 🗶 Dangling pointer Pointer 1 🚊 💥 Pointer Error 1 Current Call Stack - Thread 0 [0x0108] ٠ X Pointer argument range error Pointer Function . File 🗄 💥 Pointer Unrelated -1 Pointer ArrayParamExRange PTRERR.CPP 🗶 Unrelated pointer comparison 1 Pointer ExecuteFunction BugBench7Dlg.cp 🗄 💥 Read Overrun 1 OnTest BugBench7Dlg.cpj 🗄 💥 Write Overrun _AfxDispatchCmdMsg cmdtarg.cpp -OnCmdMsg cmdtarg.cpp OnCmdMsa dlacore.cpp 🗉 Summary 💧 Memory Leaks 🐴 Other Leaks 🗶 Errors 💈 🕨 c:\program files\compuware\devpartner studio\examples\bugbench7\main\ptrerr.cpp TBY Reference Count View Object Identity View inta[5]; 양 며 양 며 붙 네 int b; b = a[6]; // array index out of range CATCH 123456789 11 13 15 AddRef - Thread 0 [0x0070] -Line / Offset 🔺 Source Pane tion File ole32.dll 0x0001BD6F Displays source code for the ateInstance comip.h 570 121 695 M_Interface_Leak comerr.cpp detected error, if available. bugbench7dlg.cpp ecuteFunction est bugbench7dlg.cpp 639 xDispatchCmdMsg cmdtarg.cpp 88 CmdMsg cmdtarg.cpp 396 -Details Pane - Reference Count Graph

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

Reference Count Graph Toolbar



Program Error Detected Dialog Box

Error description		Tabs for multiple call stacks
Program Error D stected - BugBench7.exe Memory Leak Leaving Scope: Variable rel	e erences address 0x031A	5130 (64) allocated by
CObject::operator new. Current Call Stack - Thread 0 [0x0408] Call Sta	ck At Allocation - Thread	0 [0x0408]
Function	File	Line / Offset
DIIMain	main.cpp	67
_DIMainCRTStartup	crtdll.c	272
	ntdll.dll	0x00007FC9
// Regular DLL's resource chair // result.	n, and serious problems w	ill 🔺
new CDynLinkLibrary(MainDLL);		
} else if (dwReason == DLL_PROCESS_D {	ETACH)	
J TBACEO("MAIN BLIG Terminatin	ial/n"):	
Explain Memory/Resource Viewer		Copy Suppress
Don't show this error dialog This Run	•	
Disable event logging This Run	Debug	Halt Continue
Call stack information	Source	e code for the detected error

Results pane Memory Contents Pane Right-click to select display type. Displays Memory, Resource, and Summary tabs. Stack Pane); DevPartner Error Detection Memory and Resource Viewer _ 🗆 🗵 Thread ID 12F6 6B78 10000000 Location (combined) Byte . . B7C 000002 12F6 CBupUtility::FillTree - [bugutility.cpp - line 159 (main.bug)] 0x041c 40 6B80 00EA64 02F(dê CBu gUtility::FillTree - [bugutility.cpp - line 159 (main.bug)] 0x041c 40 02F66B84 00EA65 eê. CBubUtility::FillTree - [bugutility.cpp - line 159 (main.bug)] 0x041c 40 02F66B88 017BA6 |{.. 8-2 1 0x041c CBugUtility::FillTree - [bugutility.cpp - line 159 (main.bug]] 40 02F66B8C B ADFOOD CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)] 0x041c 40 .ã_2 02F66B90 BAADFOOD CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)] 0x041c 40 02F66B94 BAADFOOD ð-2 CBugUtility::FillTree - [bugutility.cpp - line 97 (main.bug)] 0x041c 40 02F66B98 BAADFOOD ð-2 CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 0x041c 68 02F66B9C BAADFOOD ă_2 (MFC70.DLL)] CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 28 0x041c (MFC70.DLL)] 28 CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 0x041c [MFC70.DLL]] CMapWordToOb::InitHashTable - [map_wp.cpp - line 68 0x041c 28 [MFC70.DLL]] CNoTrackObject:operator new - [afxtls.cpp - line 80 200 0x041c (MFC70.DLL) Function File Line / Offset 4.240 CNoTrackObject::operator new - [afxtls.cop - line 80 0x041c CBugUtility::FillTree bugutility.. 159 IMFC70.DLL11 PopulateTree main.cpp 89 CBugBench7Dlg::Init... bugbench.. 353 ♦ Resources 🗉 Summary o Memory 400 CBugBench7Dlg::Lo... bugbench. c:\program files\compuware\devpartner studio\examples\bugbench7\main\bugutility.cpp * szBuff sizeof (szBuff))); stTVIS.item.pszText = szBuff ; // Allocate the structure we put in 1Param. pLPI = new LParamInfo ; m lParamArray[m nLParam++] = pLPI ; $/\overline{/}$ This is a leaf node. pLPI->iType = 2 ; pLPI->stEO = *pstEO ; Mark and Close Close Line Number: 159 Showing all items • Help Save Source Pane

Memory and Resource Viewer Dialog Box

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 Resouce viewer reappears.



Select analysis preference Error Detection Coverage Error Detection Toolbar Error Detection with Coverage Display Error and Pause Performance Log Events -Memory Set DevPartner Options Error detection Native C/C++ Instrumentation Enable/disable instrumentation Choose instrumentation type BoundsChecker Toolbar in Visual C++ 6.0 Error Detection Coverage Performance S 🖸 Δ. (<u>الج</u> Log Events Build with Performance

Display Error and Pause -Build with Coverage Show filtered messages Build with Error Detection

Icons Used in the Results Pane

lcon	Description	Appears in
۵	Memory Leaks	Summary, Memory Leaks, Transcript tabs
•	Other Leaks	Summary, Other Leaks, Transcript tabs
×	Errors	Summary, Errors, Transcript tabs
4	Warning	Summary, Errors, Transcript tabs
1	Debug String	Transcript tab
	.NET Performance	Summary, .NET Performance tabs
م رچ	Module Load Event	Summary, Modules and Transcript tabs
٠	Subroutine call	Transcript tab
<u>ش</u>	Garbage Collection Event	Transcript tab
Ŧ	Event Begins	Transcript tab
*	Event Resumes	Transcript tab
Ŧ	Event Ends	Transcript tab

Icons Used in the Details Pane

lcon	Description
٠	Subroutine call
(L)	Entry Parameters
(†)	Exit Parameters
0+	Return Value
	Property (default) for data types
<i>.</i>	Property for data types

ActiveCheck and FinalCheck Error Detection

ActiveCheck

.NET Errors

Finalizer errors

code

GC.Suppress finalize not called

Unhandled native exception passed to managed

Dispose attributes errors

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	

Pointer and Leak Errors

Interface leak

Memory leak Resource leak

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 - Deepest Error Detection

FinalCheck[™] enables BoundsChecker 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

Keyboard Commands

List of Available Keyboard Commands - Visual Studio .NET

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

List of Available Keyboard Commands - Visual C++ 6.0

Command	Action
Ctrl+F	Edit > Find
Ctrl+H	Edit > Replace
Ctrl+G	Edit > Go To
Alt+F2	Edit > Bookmarks
Alt+F9	Edit > Breakpoints
Ctrl+Alt+T	Edit > List Members
Ctrl+Shift+space	Edit > Parameter Info
Ctrl+Space	Edit > Complete Word
Ctrl+W	View > ClassWizard
Alt+0	View > Workspace
Alt+2	View > Output
Alt+Enter	View > Properties
Ctrl+F7	Build > Compile <i>filename</i>
F7	Build > Build 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

Command Line Reference

Command Line Reference

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	То
/NMhelp or /?	Display help text
/NMignore:source-file	Specify a source file that should not be instrumented
/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)
/NMlog:log-file	Specify a log file for NMCL messages (default: stdout)
/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.
/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.

Use	То
/NMbcpath:bc-path	Specify the directory location of bcinterf.lib if you do not have the directory that contains NMCL on your path.
/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.
/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.
/NMbcOn	Use Error Detection instrumentation. This is the default setting.
/NMtxOn	Specifies instrumentation for performance and coverage analysis.
/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.
/NMtxInlines	Instruments methods that are marked as inlineable if inline optimizations are enabled (using the /O1, /O2, /Ob1, or /Ob2 option)

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	То
/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.
/NMbcOn	Use BoundsChecker instrumentation. If you do not specify this option, BoundsChecker cannot instrument your code.
/NMtxOn	Specifies instrumentation for performance and coverage analysis.

Use	То
/NMbcpath:bc-path	Specify the directory location of bcinterf.lib if you do not have the directory that contains NMCL on your path.
/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
/NMpass	Specify pass-through mode, which instructs NMLINK to call LINK without intervention.

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