DPVC Quick Reference

Default Options (Visual Studio .NET) 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 .NET).
	-	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 .NET).
	Off	Generate NLB files dynamically
API Call Reporting	Off	Enable API call reporting. *All category selections are unavailable until you check 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 category selections are unavailable until you check 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*
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 requrested 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*

Category		Settings
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*

Category		Settings
	-	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
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

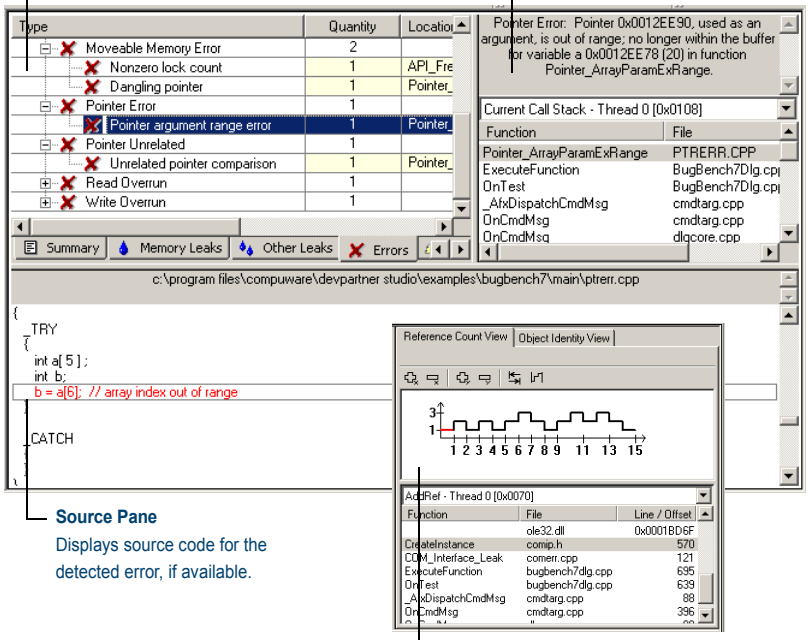
BoundsChecker Window

Results Pane

Summary, Memory Leaks, Other Leaks, Errors, .NET Performance, Modules, Transcript tabs provide overview and detail about detected errors.

Details Pane

Displays long description of detected error; call stack information; reference count graph (see inset below).



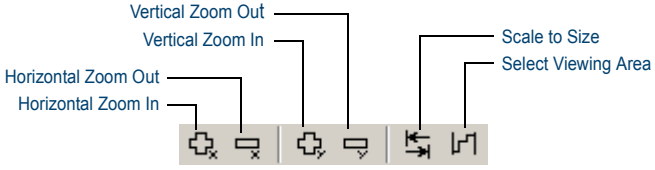
Source Pane

Displays source code for the detected error, if available.

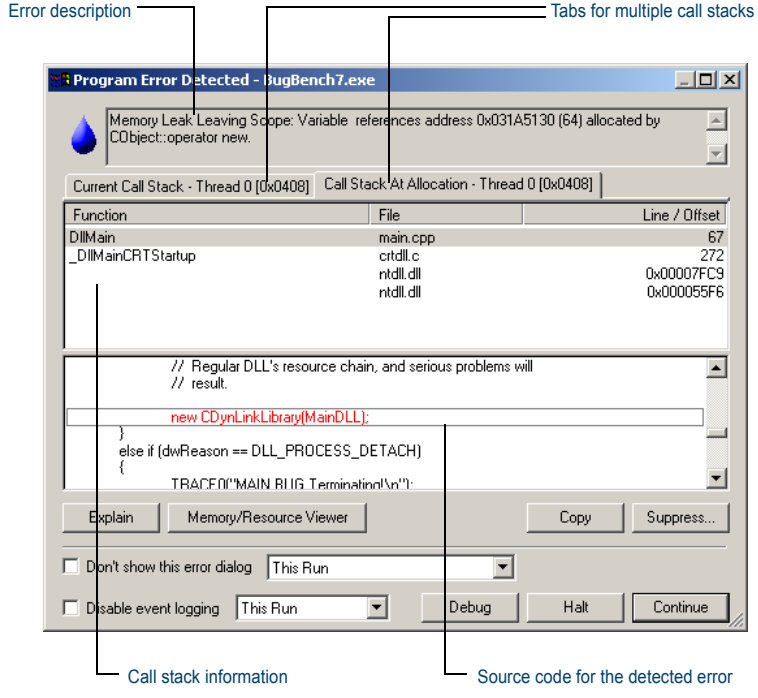
Details Pane - Reference Count Graph

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

Reference Count Graph Toolbar



Program Error Detected Dialog Box

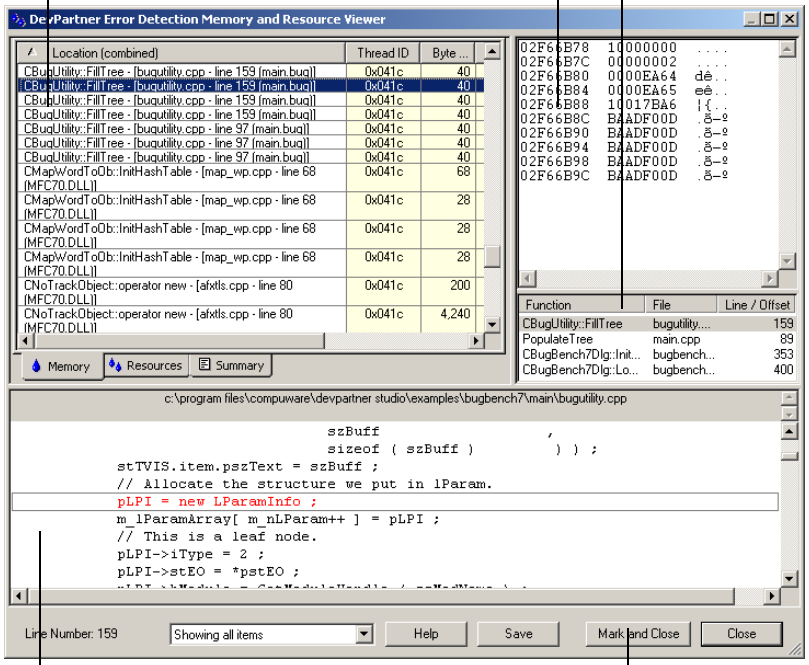


Memory and Resource Viewer Dialog Box

Results pane
Displays Memory, Resource, and Summary tabs.

Memory Contents Pane
Right-click to select display type.

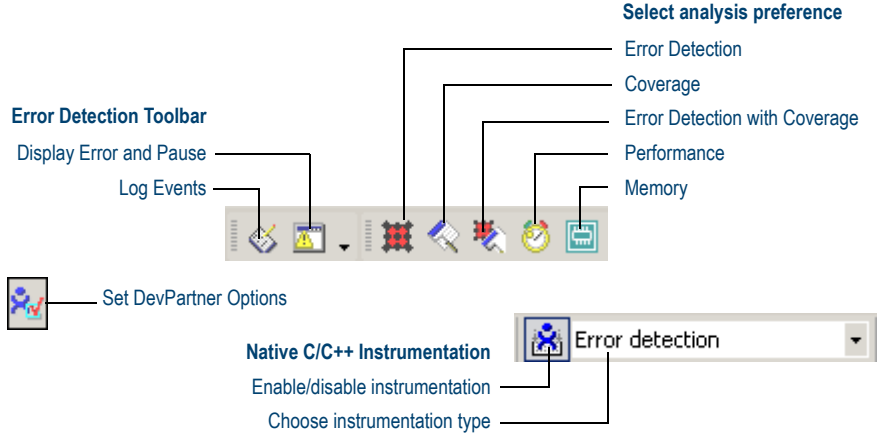
Stack Pane



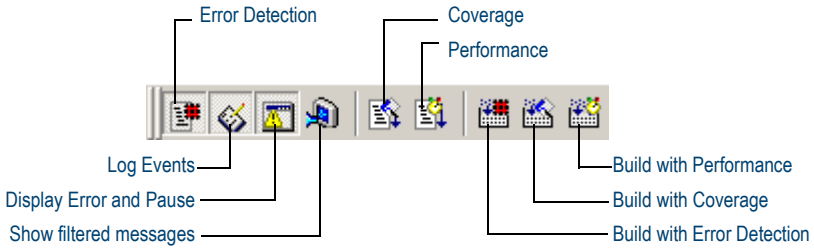
Source Pane
Displays source code for the detected error, if available.

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

BoundsChecker Toolbar in Visual Studio .NET



BoundsChecker Toolbar in Visual C++ 6.0



Icons Used in the Results Pane

Icon	Description	Appears in...
	Memory Leaks	Summary, Memory Leaks, Transcript tabs
	Other Leaks	Summary, Other Leaks, Transcript tabs
	Errors	Summary, Errors, Transcript tabs
	Warning	Summary, Errors, Transcript tabs
	Debug String	Transcript tab
	.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

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 - 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 *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

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...	To...
/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.
/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...	To...
/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.
/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