

Getting Ahead with Purify

support@rational.com
<http://www.rational.com>

RATIONAL
SOFTWARE CORPORATION

IMPORTANT NOTICE

COPYRIGHT NOTICE

Copyright © 1996, 1998 Rational Software Corporation. All rights reserved.

THIS DOCUMENT IS PROTECTED BY COPYRIGHT AND CONTAINS INFORMATION PROPRIETARY TO RATIONAL. ANY COPYING, ADAPTATION, DISTRIBUTION, OR PUBLIC DISPLAY OF THIS DOCUMENT WITHOUT THE EXPRESS WRITTEN CONSENT OF RATIONAL IS STRICTLY PROHIBITED. THE RECEIPT OR POSSESSION OF THIS DOCUMENT DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISTRIBUTE ITS CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE, IN WHOLE OR IN PART, WITHOUT THE SPECIFIC WRITTEN CONSENT OF RATIONAL.

U.S. GOVERNMENT RIGHTS NOTICE

U.S. GOVERNMENT RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational License Agreement and in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct 1988), FAR 12.212(a) 1995, FAR 52.227-19, or FAR 52.227-14, as applicable.

TRADEMARK NOTICE

Rational, the Rational logo, Purify, ClearQuest, and Rational Visual Test are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries.

Visual C++, Windows NT, Developer Studio, and Microsoft are trademarks or registered trademarks of the Microsoft Corporation. All other names are used for identification purposes only and are trademarks or registered trademarks of their respective companies.

U.S. PATENT NOTICE

U.S. Registered Patent Nos. 5,193,180 and 5,335,344 and 5,535,329. Licensed under Sun Microsystems Inc.'s U.S. Pat. No. 5,404,499. Other U.S. and foreign patents pending.

Printed in the U.S.A.

Contents

Welcome to Purify	5
Check every component in your program	5
Find errors before they occur	6
Don't wait—use Purify early and often	6
Getting started	7
Running a program with Purify	7
Seeing all your errors at a glance	9
Focusing on critical errors first	11
Analyzing messages	13
Correcting errors	14
Comparing program runs	15
Saving error data	16
Using Purify's power features	17
Customizing error detection	17
Using just-in-time debugging	18
Extending error checking with Purify API functions	19
Using Purify standalone	19
Testing with Purify's command-line interface	21
Using Purify in a highly integrated environment	22
Index	25

Welcome to Purify

Today's competitive software development is component based. To deliver quality applications on time, you not only need to make sure your own code is error free, you also need a way to check the components your software uses—even when you don't have the source code.

That's where Purify[®] can help you get ahead. Purify provides the fastest and most comprehensive run-time error detection available for Visual C++ programs. Purify automatically integrates into Microsoft Developer Studio 97 and later, and requires no special builds, so you can use Purify without changing the way you work.

Check every component in your program

Purify thoroughly checks every component in your program, even in complex multi-threaded, multi-process applications, including:

- COM-enabled applications using OLE and ActiveX controls
- DLLs, including Windows DLLs and Microsoft Foundation Class Library DLLs
- Visual C/C++ components embedded within Visual Basic applications, Internet Explorer, Netscape Navigator, or any Microsoft Office application
- Microsoft Excel and Microsoft Word plug-ins
- Applications running in Windows CE Emulation Mode on Windows NT

Purify also checks calls to Windows API functions, validating parameters such as memory handles and pointers. Included are GDI, Internet services, system registry, and COM and OLE interface API functions.

Find errors before they occur

Run-time errors and memory leaks are among the most difficult errors to locate and the most important to correct. The symptoms of incorrect memory use are unpredictable, and typically appear far from the cause of the error. The errors often remain undetected until triggered by some random event, so that a program can seem to work correctly when in fact it's only working by accident.

Purify detects the following kinds of memory errors, and many others, before they actually occur, so you can resolve them before they do any damage:

- Array bounds errors
- Accesses through dangling pointers
- Uninitialized memory reads
- Memory allocation errors
- Memory leaks

More information? For a complete list of the errors Purify detects, select Purify Messages from the Purify Help menu.

Don't wait—use Purify early and often

For maximum benefit, start using Purify as soon as your code is ready to run and continue using it regularly throughout your development cycle, especially for:

Acceptance tests: Validate third-party code or code from other development groups before incorporating it into your application.

Code check-in: Reduce the risk that bugs in your code might impact other code modules.

Nightly builds: Incorporate Purify into your test harness to verify that modules work together and to expose code dependencies and collisions.

By using Purify early and often, you'll release clean, reliable products—on time.

Getting started

With Purify, you can deliver cleaner code in a few easy steps:

- 1 Run your program with Purify in Microsoft Developer Studio 97 or later.
- 2 Analyze error messages.
- 3 Correct your source code.
- 4 Rerun the program to verify your corrections.

You can also use Purify independently of Developer Studio. Read “Using Purify standalone” on page 19 of this guide, and “Testing with Purify’s command-line interface” on page 21.

Running a program with Purify

Open your project in Developer Studio, then engage Purify from the Purify toolbar.

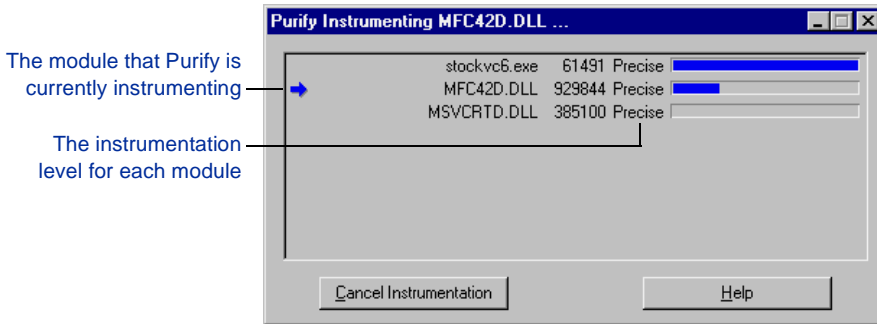
Click to engage Purify



Build and execute your program using commands from the Developer Studio **Build** menu. (To get the maximum level of detail in Purify error reports, build your program with debug and relocation data. For more information, look up *debug data, locating* in the Purify online Help index.)

Purify copies the program and each library it calls, then *instruments* the copies using Object Code Insertion (OCI) technology. The instrumentation process inserts instructions that validate every read, write, and allocation and deallocation of memory.

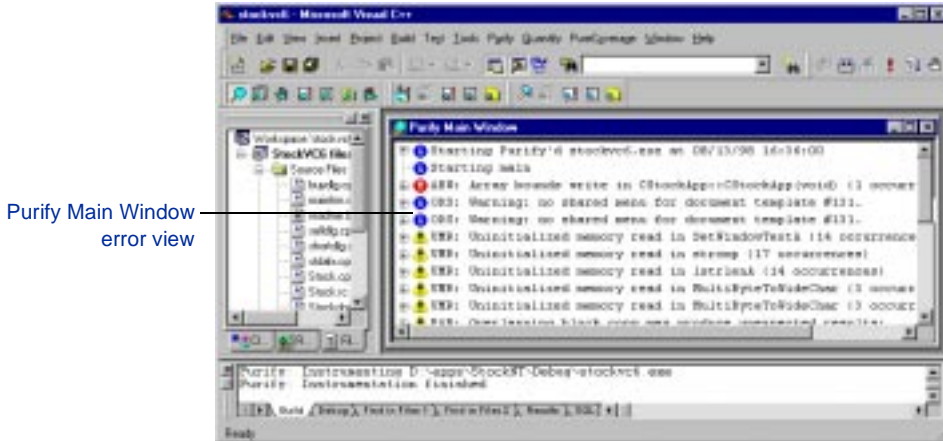
You can follow Purify's progress as it instruments each module.



You can customize Purify's instrumentation level to provide more or less detail for special cases. For more information, read "Customizing error detection" on page 17 of this guide.

Purify caches the instrumented modules. When you rerun a program, Purify saves time and resources by using the cached modules, re-instrumenting only those that have changed since the previous run.

As you exercise your program, Purify detects run-time errors and memory leaks and displays them in an error view in the Purify Main Window.



Seeing all your errors at a glance

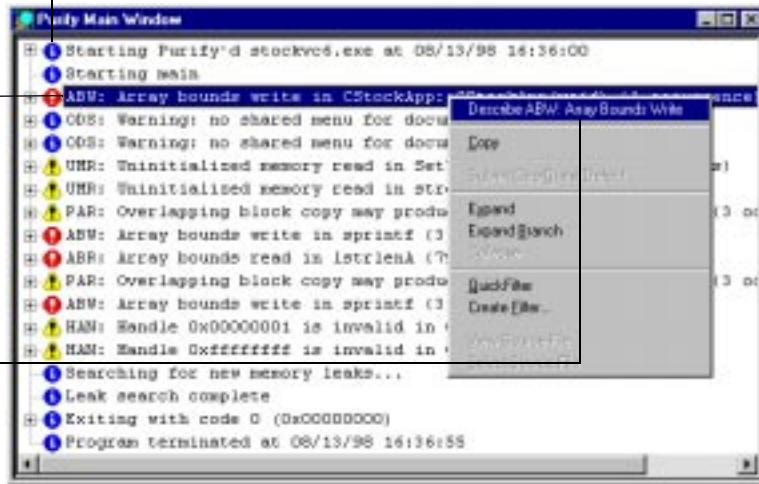
Purify displays informational messages about the state of your program's execution, as well as messages about run-time errors and memory leaks.

Color-coded icons show message severity:

 informational  warning  error

Acronyms like ABW identify message type

For a description of a message, right-click the message, then select Describe



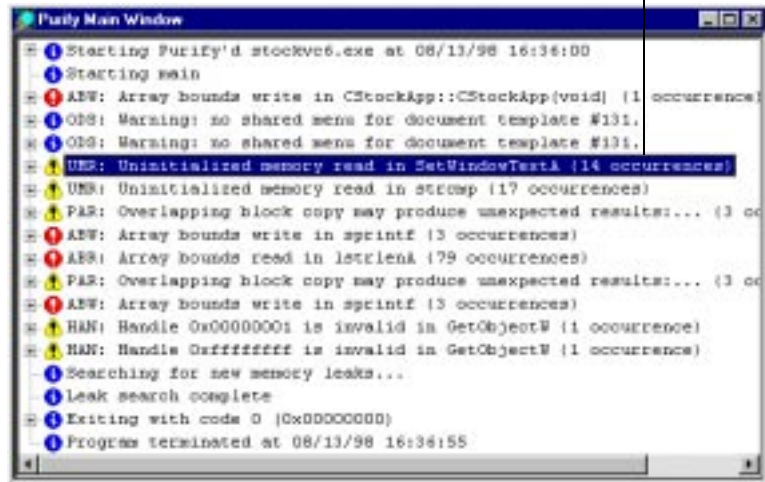
When you exit the program, Purify reports memory leaks. In addition to memory leaks, you can set Purify to report memory in use at exit and handles in use at exit.

More information? Look up *error and leak settings* in the online Help index.

When identical errors repeat

An error often repeats many times in a program, particularly if it occurs inside a loop. To provide a succinct overview of a program's errors, Purify by default displays each error message only once, the first time an error occurs, and then updates a counter whenever the error repeats.

This Uninitialized memory read (UMR) occurred 14 times



```
Purify Main Window
- Starting Purify'd stockvc6.exe at 08/13/98 16:36:00
- Starting main
- ABW: Array bounds write in CStackApp::CStackApp(void) (1 occurrence)
- ODS: Warning: no shared menu for document template #131.
- ODS: Warning: no shared menu for document template #131.
- UMR: Uninitialized memory read in SetWindowText (14 occurrences)
- UMR: Uninitialized memory read in strcpy (17 occurrences)
- PAR: Overlapping block copy may produce unexpected results:... (3 oc
- ABW: Array bounds write in sprintf (3 occurrences)
- ABR: Array bounds read in strlen (79 occurrences)
- PAR: Overlapping block copy may produce unexpected results:... (3 oc
- ABW: Array bounds write in sprintf (3 occurrences)
- HAN: Handle 0x00000000 is invalid in GetObjectW (1 occurrence)
- HAN: Handle 0xffffffff is invalid in GetObjectW (1 occurrence)
- Searching for new memory leaks...
- Leak search complete
- Exiting with code 0 (0x00000000)
- Program terminated at 08/13/98 16:36:55
```

More information? If you want Purify to display each occurrence of a message individually, instead of reporting counts, you can change the default setting. Look up *error and leak settings* in the online Help index.

Focusing on critical errors first

A large program can generate hundreds of messages. To focus on the most critical error messages quickly, create filters to hide all other messages from the display.

You can filter messages individually, or you can filter them based on their type and source. Consider hiding all informational messages, for example, or all messages originating from a specific file.


An *unfiltered* error view displays all the messages from the program

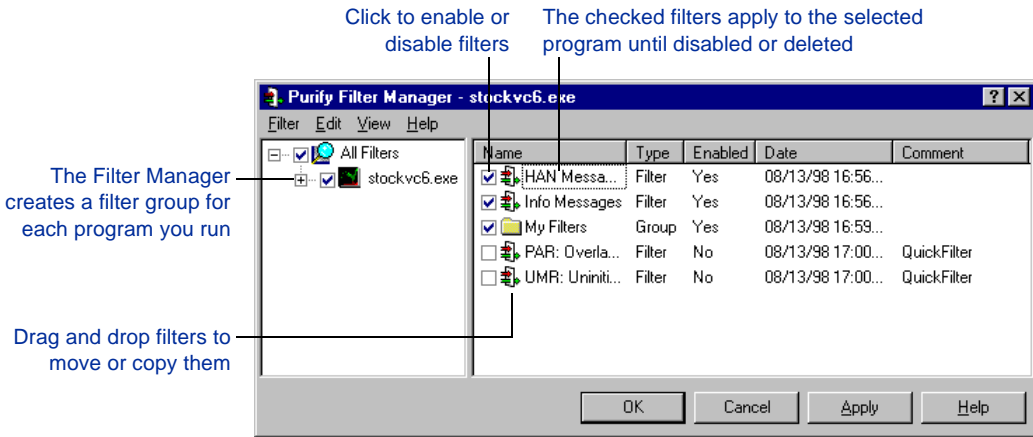
A *filtered* error view displays only the messages you want to see



Once created, filters apply to the current run and to all future runs of the program until you disable them. Disabling a filter causes hidden messages to be redisplayed in the error view.

Working with filters

Purify filters are very flexible. Click the Filter Manager tool  to create individual filters or groups of filters, and to apply them to specific programs or modules. You can also create global filters that apply to all programs and modules. And you can share filters, which Purify saves as `.pft` files, with other members of your team.



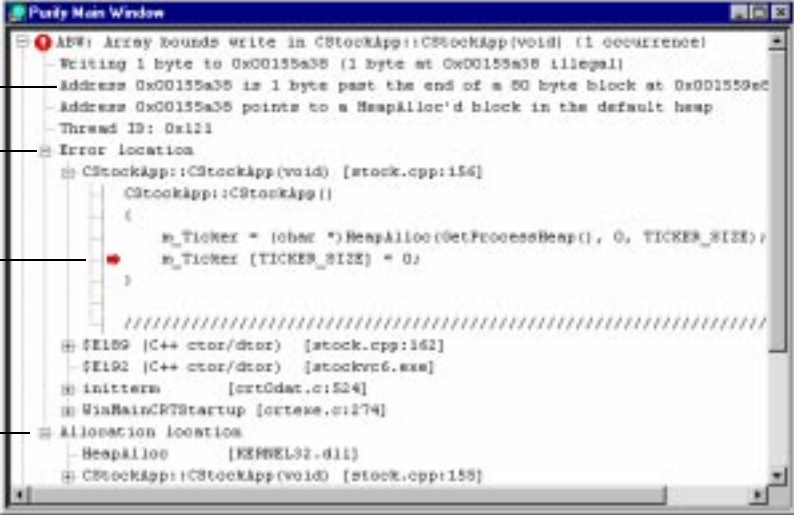
More information? Look up *filters, about* in the online Help index.

In addition to filtering, Purify offers another way of focusing on the errors in specific modules and simultaneously minimizing instrumentation time—the PowerCheck feature, which allows you to optimize the level of error detection for each module. For more information, read “Customizing error detection” on page 17 of this guide.

Analyzing messages

You can expand Purify's messages to pinpoint *where* errors occur and to obtain diagnostic information for understanding *why* they occur.

Here's an example of an expanded ABW (Array Bounds Write) error message:



The location in memory where the error occurs

Call stack showing the function calls leading to the error

Flag indicating the line where the error occurs

Call stack showing the function calls leading to the allocation of the memory block associated with the error

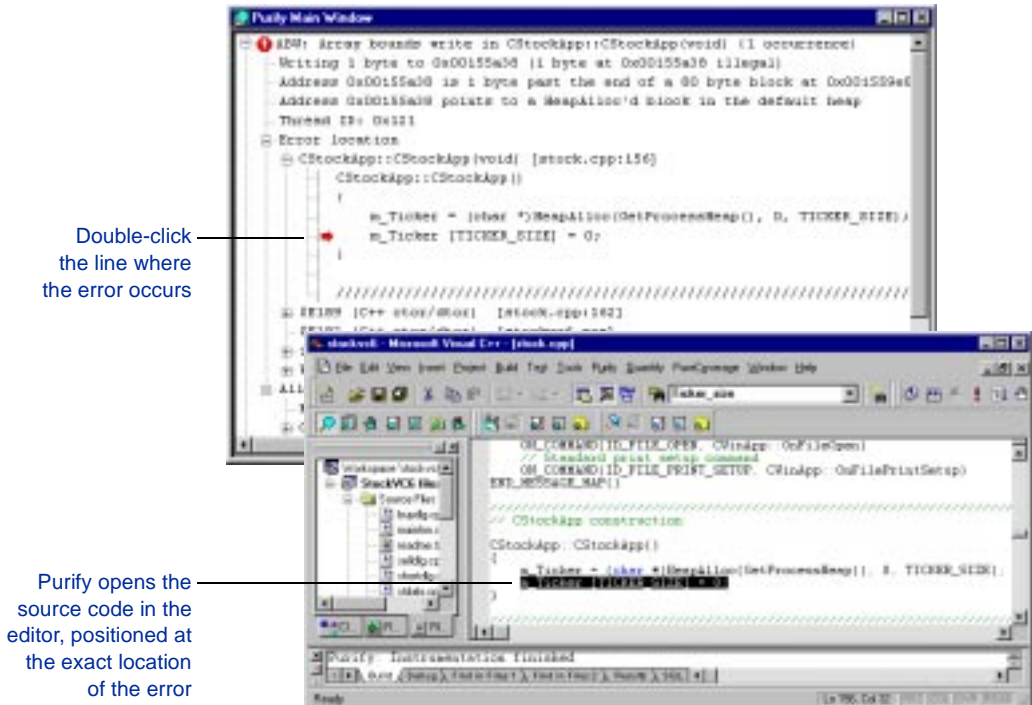
```
Purify Main Window
- ABW: Array Bounds Write in CStockApp::CStockApp(void) (1 occurrence)
- Writing 1 byte to 0x00155a38 (1 byte at 0x00155a38 illegal)
- Address 0x00155a38 is 1 byte past the end of a 50 byte block at 0x001559e0
- Address 0x00155a38 points to a HeapAlloc'd block in the default heap
- Thread ID: 0x121
- Error location
  CStockApp::CStockApp(void) [stock.cpp:156]
  CStockApp::CStockApp()
  {
    m_Ticker = (ohsc *)HeapAlloc(GetProcessHeap(), 0, TICKER_SIZE);
    m_Ticker [TICKER_SIZE] = 0;
  }
  ///////////////////////////////////////////////////////////////////
  @E189 [C++ ctor/dtor] [stock.cpp:162]
  @E192 [C++ ctor/dtor] [stockv6.exe]
  @InitTerm [crt0dat.c:524]
  @WinMainCRTStartup [crtexe.c:274]
- Allocation location
  HeapAlloc [KERNEL32.dll]
  CStockApp::CStockApp(void) [stock.cpp:156]
```

The level of detail provided in call stacks depends on the availability of debug and relocation data. Even if you build your program in release mode, you can still get the highest possible level of detail. For more information, look up *debug data, for release builds* in the online Help index.

You can customize the format of Purify's messages. For example, you can increase the number of lines of source code that are displayed, or include instruction pointers and offsets to make locating errors easier. For more information, look up *preferences, source code* in the online Help index.

Correcting errors

Purify makes it easy to correct errors.



After correcting the errors, rebuild your program. Then rerun the program with Purify engaged.

Comparing program runs

After rerunning your corrected program, you can easily compare runs to verify your corrections. Purify's Navigator window, which you can display from the Purify View menu, helps you keep track of multiple runs and multiple programs.

The Navigator window groups runs by program

A color-coded icon indicates the maximum message severity displayed in the error view for the run



More information? You can customize the information displayed in the Navigator window. Look up *navigator window* in the online Help index.

Checking multi-process applications

If you're debugging client/server and multi-process applications, you can debug several processes and see the error reports for each running application simultaneously. To do this, run each process in a separate instance of Developer Studio with Purify engaged. Alternatively, you can use the standalone Purify user interface. See "Using Purify standalone" on page 19.

Saving error data

You can save Purify error data from a run in order to analyze it later, share it with other members of your team, or include it in reports. You can save Purify error data in two formats:

- Purify data files (.pfy), with or without any messages you filtered out. Later, you can open the saved file to analyze it or to compare it to future program runs.
- ASCII text files (.txt), for use in spreadsheet and word-processing programs.

More information? Look up *error view data, saving* in the online Help index.

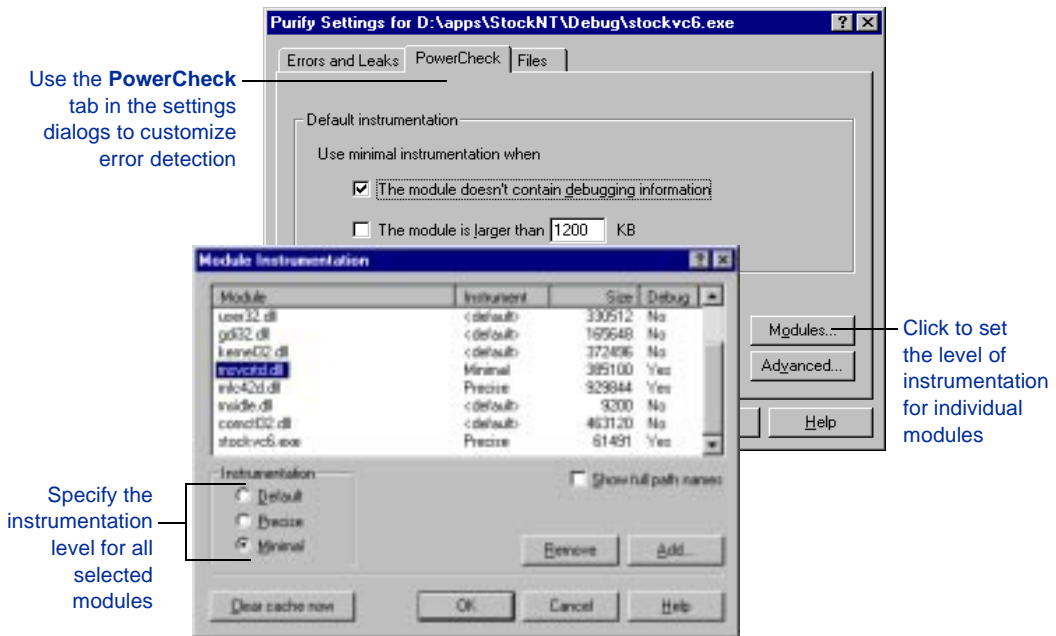
Using Purify's power features

Customizing error detection

You can control the level of error detection for each module in a program by selecting one of Purify's two instrumentation levels:

- *Precise* instrumentation provides full run-time error detection, to pinpoint problems in any component in your program.
- *Minimal* instrumentation improves Purify's performance while providing a basic level of error detection.


Purify sets a default level for each module based on module size and the availability of debug and relocation data. However, you can override the default and specify instrumentation levels for each module to meet your own requirements.

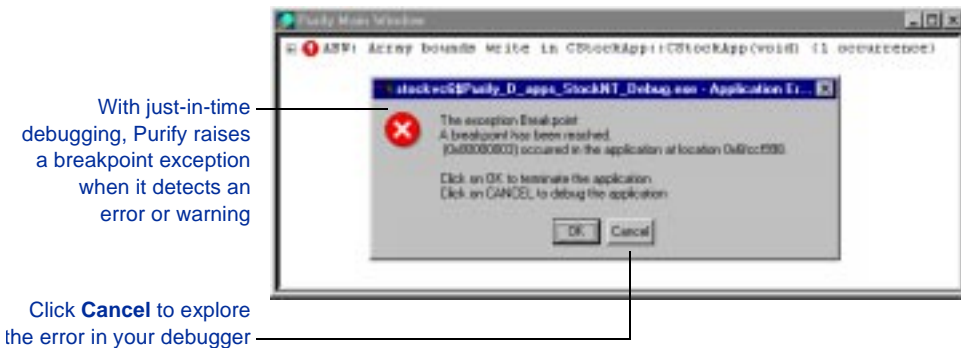


Try using the *precise* setting for the most critical modules in your program and the *minimal* setting for the others. Later, you can change the minimal settings to precise for a thorough check of the other modules.

More information? Look up *powercheck* in the online Help index. For information about how to provide debug and relocation data when you are using Purify on release builds, look up *debug data, for release builds*.

Using just-in-time debugging

Purify's just-in-time debugging support provides instant access to your debugger when you need to solve tough problems. Click  to enable Break on Error, so that Purify stops your program just before an error executes and lets you start your debugger if you're not already running Purify inside Microsoft Developer Studio 97 or later. You can also run a Purify'd program directly under the debugger.



To quickly debug *only* the most critical errors in your program, use Break on Error together with Purify filters. First, filter out all the less critical messages, then enable Break on Error. Purify breaks only for the unfiltered messages. When you're ready to debug the remaining errors, just disable the filters.

More information? Look up *break on error tool* and *filters* in the online Help index.

Extending error checking with Purify API functions

Purify includes a set of API functions that extend Purify's error checking capabilities and give you greater control over tracking errors.

Using Purify's API functions, you can set and test memory state, and search for memory and handle leaks. For example, by default Purify reports memory leaks only when you exit your program. However, if you call the API function `PurifyNewLeaks` at key points throughout your program, Purify reports any new memory leaks it has detected since the last time the function was called. This periodic checking enables you to track memory leaks more closely.

You can call Purify API functions from your program or from the QuickWatch dialog in the Developer Studio debugger.

More information? Look up *api functions, list* and *api functions, using* in the online Help index.

Using Purify standalone


When you don't need all of Microsoft Developer Studio's resources, you can use Purify standalone. Purify's independent user interface provides the same intuitive, efficient error-detection capabilities as when you use Purify integrated with Developer Studio.

Note: You can also use Purify's independent user interface while continuing to work integrated with Developer Studio by deselecting **Embed Error Views** in the Purify Settings menu.

To start Purify as a standalone application, double-click 



Purify instruments your code and opens an Error View window to display the results.

More information? For information about a user interface item such as a tool or menu command, click  and then click the item.

Testing with Purify's command-line interface

Using Purify's command-line interface, you can use Purify with existing makefiles, batch files, and Perl scripts. For example, if you have a test script that runs a program, you can easily modify the script to instrument and run the program. To do this, change the line that runs *Exename.exe* to:

```
purify Exename.exe
```

Alternatively, to run the instrumented version of *Exename.exe* consistently throughout your tests, add this line to the beginning of your test script:

```
purify /Replace /Run=no Exename.exe
```

This line instructs Purify to save the original *Exename.exe* to a *.bak* file, and to instrument *Exename.exe* but not to run it at this time. Now, whenever your test script runs *Exename.exe*, it runs the instrumented version of the program, providing Purify's detailed diagnostics.

You can run Purify without the graphical interface by using the */SaveTextData* option. This option saves Purify's diagnostic messages to a text output file. You can use the error and warning messages in this file as additional criteria for your test results.

More information? Look up *command line* in the online Help index.

Using Purify in a highly integrated environment

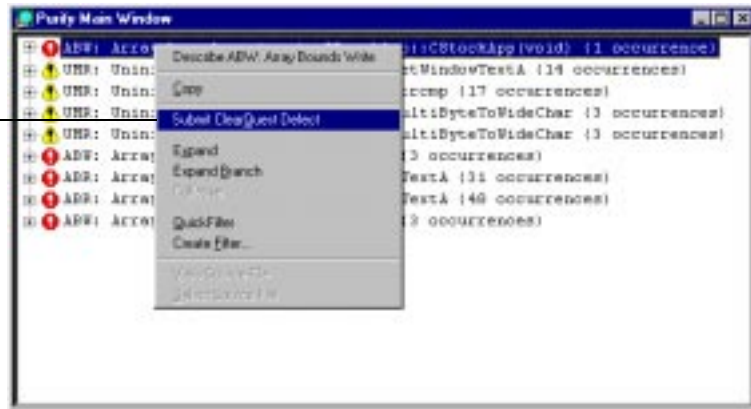
Rational Software tools integrate into your working environment to help you do your job more effectively and efficiently. Without leaving Purify or Developer Studio, you can make full use of:

- ClearQuest™, Rational's change request management tool
- Rational Visual Test®, Rational's automated test scripting tool

Using Purify with ClearQuest

If you have ClearQuest installed, you can submit a defect as soon as Purify detects an error or warning.

Right-click on an error message and select Submit ClearQuest Defect



Purify automatically supplies entries for a number of fields in the submit form and specifies the category of error. You can easily attach Purify data files (.pfy) to further document the error.

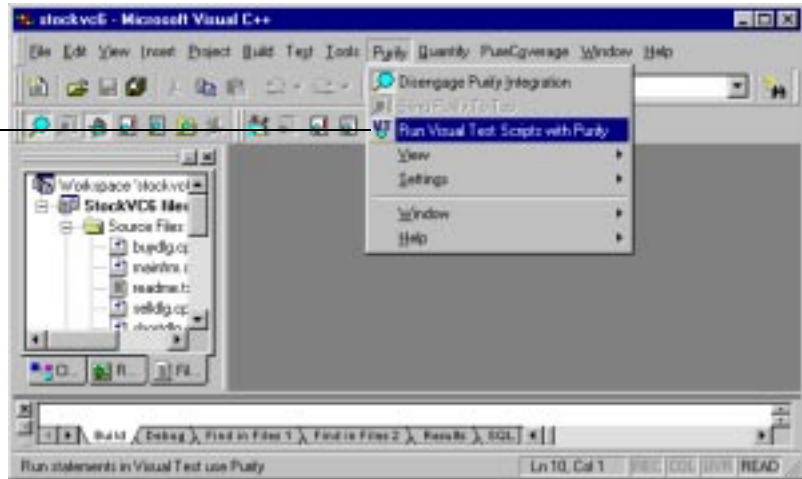
Using Purify with Visual Test

If you have Rational Visual Test 6.0 installed, you can easily Purify the program that Visual Test is exercising. To do this, you must include the file `TOOLS.INC` in your Visual Test script

file (.mst), select Purify > Run Visual Test Scripts with Purify, and run your script as usual in the Developer Studio interface.

Update your Visual Test script, then select Run Visual Test Scripts with Purify

Then run your script in Visual Test



If you are using a test harness to run Visual Test scripts, you can easily modify it to run Purify automatically as it exercises the program.

More information? Look up *clearquest* and *visual test* in the online Help index, and refer to the ClearQuest and Visual Test documentation.



Now you're ready to put Purify to work. Remember that Purify's online Help contains detailed information to assist you.

Index

A

ABW (Array Bounds Write) error 13
 API
 Purify functions 19
 Windows API checking 5

B

batch files 21
 Break on Error tool 18

C

cache files 8
 call stack 13
 ClearQuest, integrated with
 Purify 22
 client/server applications 15
 code, editing 14
 COM support 5
 command-line interface 21
 components
 See modules
 Create Filter command 11
 customizing error detection 17

D

data, saving 16
 debug data, and instrumentation 7,
 17
 debugging, just-in-time 18
 Developer Studio, integration with
 Purify 7
 displaying filtered messages 12

E

Embed Error Views command 19
 error detection, customizing 17
 error view 8, 15, 16

errors

 breaking on 18
 correcting 14
 See also messages
 exit messages 9

F

files

 caching after instrumentation 8
 .mst 23
 .pft 12
 .pfy 16
 .txt 16

filters

 filter groups 12
 Filter Manager 12
 overview 11
 saved in .pft files 12
 sharing 12
 functions, Purify API 19

G

groups, filter 12

H

handles

 in use at exit 9
 leaks 19
 hiding messages
 See filters

I

instrumentation
 customizing 17
 defined 7
 minimal 17
 precise 17

- integration
 - ClearQuest 22
 - Microsoft Developer Studio 97 or later 7–16
 - Rational Visual Test 22–23

- J**
- just-in-time debugging 18

- L**
- leaks
 - See* memory

- M**
- Main window 8
- makefiles 21
- memory
 - leaks reported at exit 9
 - PurifyNewLeaks API function 19
- menu, shortcut 9
- messages
 - analyzing 13
 - expanding 13
 - filtering 11
 - redisplaying filtered 12
 - See also* errors
- Microsoft Developer Studio, integration with Purify 7
- minimal instrumentation 17
- modules
 - custom error detection for 17
 - support for 5
- multi-process applications 15

- N**
- Navigator, overview 15

- P**
- Perl scripts 21
- .pft files 12
- .pfy files 16
- PowerCheck feature 17
- precise instrumentation 17
- programs
 - rerunning 14
 - running from command line 21
 - running under debugger 18

- Q**
- QuickFilter command 11

- R**
- Rational Visual Test, integrated with Purify 22–23
- relocation data, and instrumentation 7, 17
- rerunning a program 14
- runs, comparing multiple 15

- S**
- saving data
 - from an error view 16
 - /SaveTextData option 21
- sharing filters 12
- shortcut menu 9
- source code, editing 14
- stack, call 13
- standalone use of Purify 19

- T**
- tests, using Purify in 21, 22
- threaded application support 5
- .txt file 16

- U**
- unembedding the Purify interface from Developer Studio 19

- V**
- Visual Test, integrated with Purify 22–23

- W**
- windows
 - Error View 8, 15, 16
 - Main 8
 - Navigator 15
 - Windows API checking 5