

# Getting Ahead with Purify

[support@pureatria.com](mailto:support@pureatria.com)  
<http://www.pureatria.com>



## **IMPORTANT NOTICE**

### **DISCLAIMER OF WARRANTY**

Pure Atria makes no representations or warranties, either express or implied, by or with respect to anything in this manual, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose or for any indirect, special or consequential damages.

### **COPYRIGHT NOTICE**

Copyright © 1996-1997 Pure Atria. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of Pure Atria. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, Pure Atria assumes no responsibility for errors or omissions. This publication and features described herein are subject to change without notice.

The program and information contained herein are licensed only pursuant to a license agreement that contains use, reverse engineering, disclosure and other restrictions; accordingly, it is "Unpublished — rights reserved under the copyright laws of the United States" for purposes of the FARs.

### **RESTRICTED RIGHTS LEGEND**

Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c) (1) (a) of the Rights in Technical Data and Computer Software clause of the DFARs 252.227-7013 and FAR 52.227-19(c) and any successor rules or regulations.

### **TRADEMARKS**

Purify is a U. S. registered trademark of Pure Atria. Pure Atria, the Pure Atria logo, and PowerCheck are trademarks of Pure Atria in the United States and in other countries.

Covered by one or more of U.S. Patent Nos. 5,193,180, 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.

All other products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Printed in the U.S.A.

## Contents

### **Welcome to Purify**

Check every component in your program .....	5
Find errors before they occur .....	6
Don't wait—use Purify early and often .....	6

### **Getting started**

Instrumenting and running a program .....	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 view data .....	16

### **Using Purify's power features**

Integrating Purify into Microsoft Developer Studio 97 .....	17
Customizing error detection .....	18
Using just-in-time debugging .....	19
Extending error checking with Purify API functions .....	20
Using Purify in an existing test environment .....	20
Sharing Purify data across the team .....	21

<b>Index</b> .....	23
--------------------	----



## 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 is the fastest and most comprehensive run-time error detection tool available. It automatically integrates into Microsoft Developer Studio 97, it requires no special builds, and it lets you customize error detection for each component in your program.

### 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 Classes (MFC)
- C/C++ components embedded within Visual Basic applications, Internet Explorer, Netscape Navigator, or any Microsoft Office application
- Excel and Word plug-ins

## Find errors before they occur

Purify reports these and many other memory errors *before* they actually occur, so you can correct them quickly:

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

Purify also checks calls to Windows APIs, validating parameters for every memory handle and pointer passed through the API. These include GDI, Internet services, system registry, and COM and OLE interface APIs.

**More information?** For a complete list of the errors Purify finds, select **Help > Purify Messages** in the Purify main window.

## 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:

- Instrument and run an executable program
- Analyze error messages
- Correct your source code
- Rerun the program to verify your corrections

You can use Purify with programs built in either release mode or debug mode. However, in order for Purify to report the exact location of errors, you should build your program in debug mode.

More information? Look up *debug data* in the online Help index.

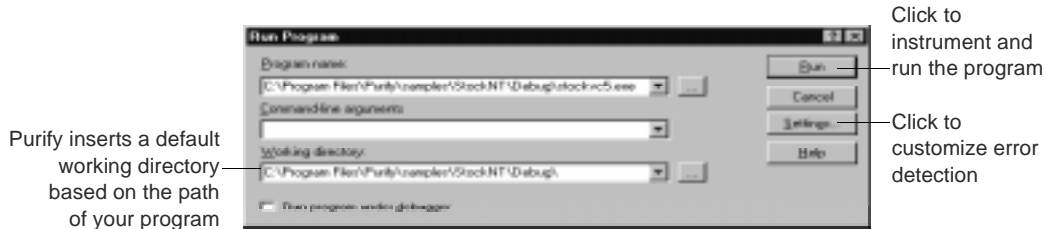
### Instrumenting and running a program

To start Purify, double-click 



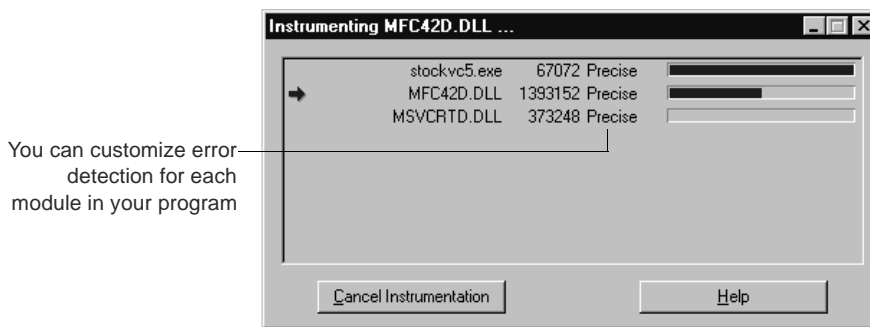
Click **Run** to begin.

Select the program you want to instrument.



Purify first copies the program and each library it calls, then instruments these modules using Object Code Insertion (OCI) technology. This inserts checking instructions that validate every read, write, allocation, and freeing of memory.

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



Purify caches the instrumented modules. When you rerun a program, Purify uses the cached modules, re-instrumenting only the modules that have changed since the previous run.

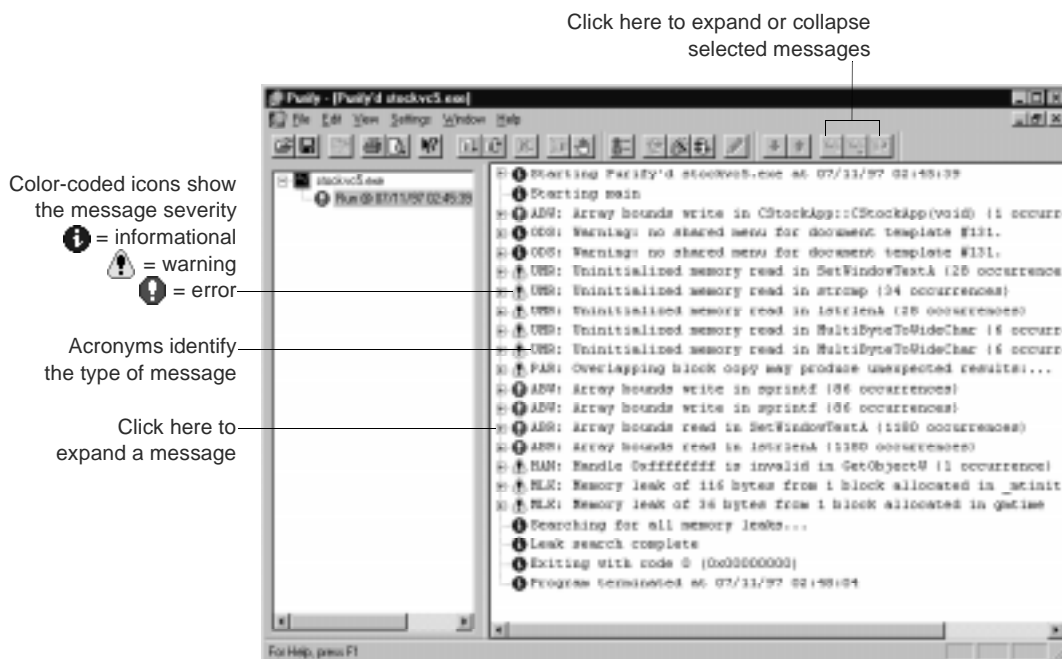
**More information?** Read “Customizing error detection” on page 18 of this manual.



## Seeing all your errors at a glance

After instrumenting the program, Purify automatically starts it. As you run the program, Purify displays run-time errors and memory leaks in an Error View window.

The error view's condensed outline makes it easy to identify the critical errors in your program. You can expand the outline and see detailed diagnostic information.



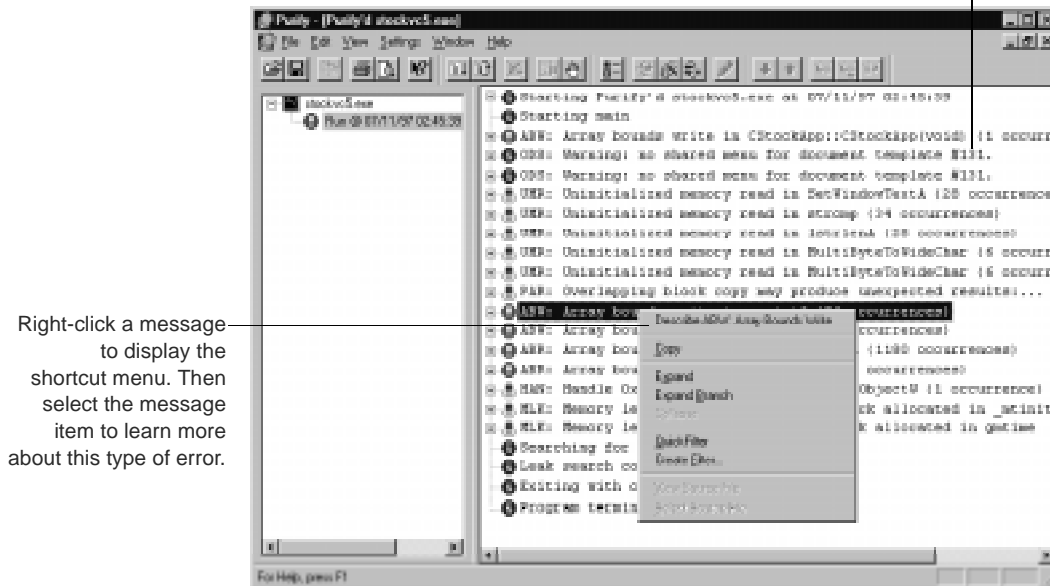
When you exit the program, Purify reports memory leaks. You can set Purify to also report memory in use and handles in use at exit.

**More information?** Look up *memory in use*, *exit* and *handles in use, exit* in the online Help index.

## When identical errors repeat

Often identical errors repeat many times in a program, such as when the error is inside a loop. To make error views easy to scan, Purify, by default, displays a message the first time the error occurs, then updates a counter each time the error repeats.

Purify reports the number of times identical errors occur



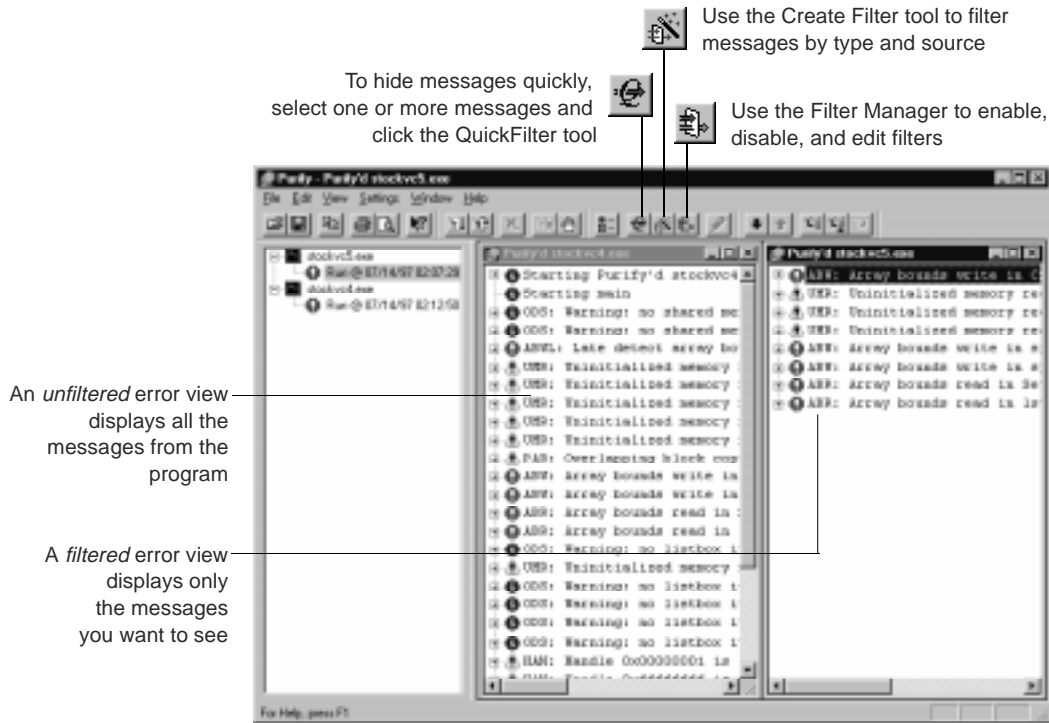
**More information?** You can also display each occurrence of a message individually. Look up *repeat count* in the online Help index.

## Focusing on critical errors first

A large program can generate hundreds of error messages. To quickly focus on the most critical ones, you can create filters to temporarily hide the other messages from the Error View window.

You can filter messages based on their type and source. For example, you might want to hide all informational messages, or hide all messages that originate in a specific file.

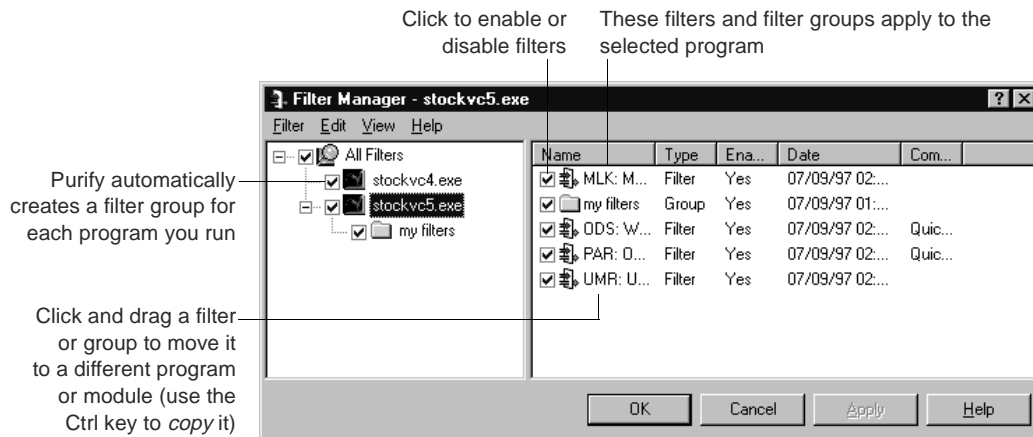
Once you're ready to deal with the hidden messages, just disable the filters to redisplay them.



Filters apply to the current run and to all future runs of the program until you disable them.

## Working with filters

Purify filters are very flexible. You can create individual filters or groups of filters, then use Purify's Filter Manager to apply them to specific programs or modules. You can also create global filters that apply to every program.



You can also share filters with other members of your development team.

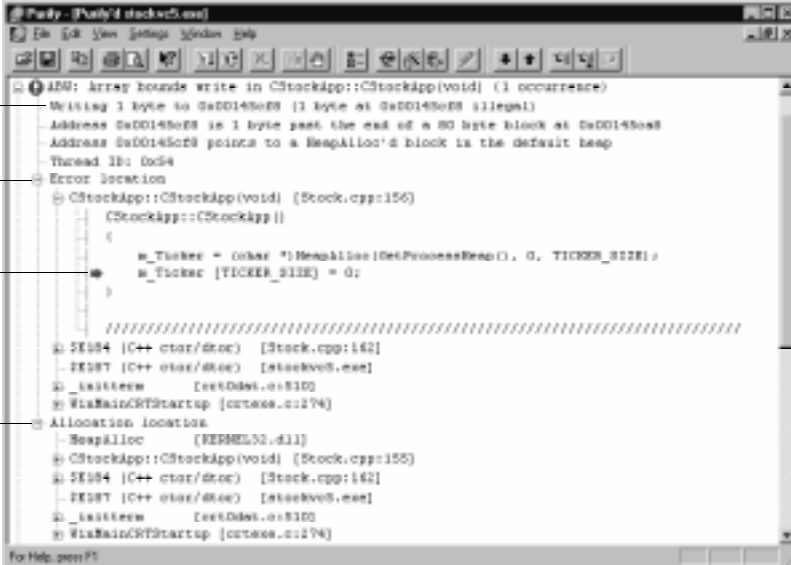
Another way to focus on errors in specific modules is to use Purify's PowerCheck™ options to customize the level of error detection for each module.

**More information?** Look up *filters*, *filters sharing*, and *PowerCheck* in the online Help index. Read “Customizing error detection” on page 18 of this manual.

## Analyzing messages

Purify's messages pinpoint *where* errors occur and provide the diagnostic information you need to analyze *why* they occur.

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



The location in memory where the error occurs

This call stack shows the function calls leading to the error

Purify flags the line where the error occurs

This call stack shows the function calls leading to the allocation of the memory block involved in the error

```
Purify - (Purify'd stackC5.exe)
[File Edit View Settings Window Help]
[Icons]
ABW: array bounds write in CStockApp::CStockApp(void) (1 occurrence)
Writing 1 byte to 0a001450e8 (1 byte at 0a001450e8 illegal)
Address 0a001450e8 is 1 byte past the end of a 80 byte block at 0a00145048
Address 0a001450e8 points to a HeapAlloc'd block in the default heap
Thread ID: 0c54
Error location
  CStockApp::CStockApp(void) [Stock.cpp:155]
    CStockApp::CStockApp()
    {
      m_Ticker = coxax "HeapAlloc (DetProcessHeap(), 0, TICKER_SIZE);
      m_Ticker [TICKER_SIZE] = 0;
    }
    ///////////////////////////////////////////////////
  a: FE54 (C++ ctor/dtor) [Stack.cpp:162]
  - FE57 (C++ ctor/dtor) [StackNv05.exe]
  a: _initteem [testDblt.o:810]
  @ WinMainCRTStartup [crtexe.o:2174]
Allocation location
  - HeapAlloc [HEAPMEM0.dll]
  @ CStockApp::CStockApp(void) [Stock.cpp:155]
  a: FE54 (C++ ctor/dtor) [Stack.cpp:162]
  - FE57 (C++ ctor/dtor) [StackNv05.exe]
  a: _initteem [testDblt.o:810]
  @ WinMainCRTStartup [crtexe.o:2174]
For Help, press F1
```

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.

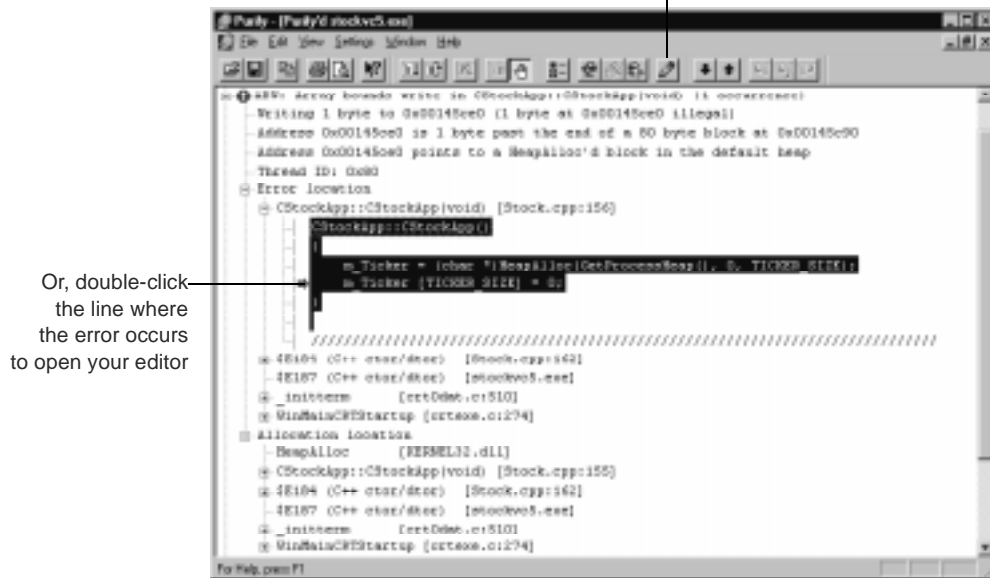
**More information?** Look up *preferences*, *source code*, and *messages*, *customizing* in the online Help index. For general tips on how to Purify your code, look up *Purify, using* in the online Help index.

## Correcting errors


Purify makes it easy to correct errors. Just double-click the line where the error occurs, and Purify opens the source code in your editor, positioned at the exact location of the error.

If you integrate Purify into Microsoft Developer Studio 97 during installation, Purify starts Developer Studio. You can, however, specify an editor of your choice.

Click to open your editor and correct the error



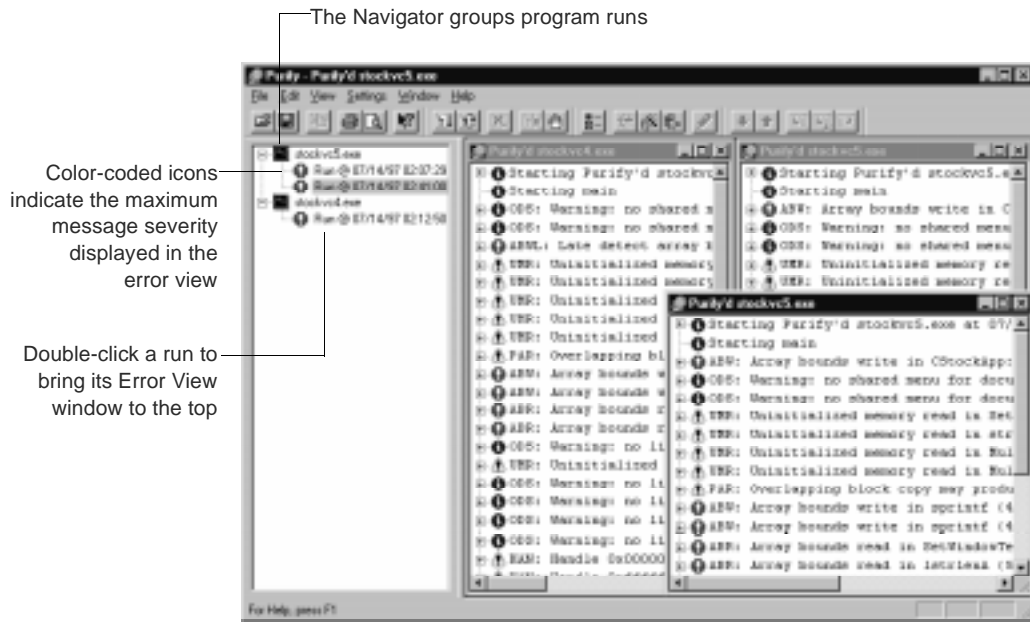
Or, double-click the line where the error occurs to open your editor

After correcting errors, rebuild your program. Then click Purify's Run Again tool  to rerun the program. Purify saves instrumentation time by re-instrumenting only the modules that have changed since the previous run.

**More information?** Look up *editor, selecting* in the online Help index.

## Comparing program runs

After correcting errors and rerunning your program, you can easily compare runs to verify your corrections. Purify's Navigator window helps you keep track of multiple programs and runs.




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

## Checking multi-process applications

Because Purify supports multiple Error View windows, it's easy to debug client/server and multi-process applications. You can debug several processes during a session and see the error reports for each running application simultaneously.

## Saving error view data

You can click the Save Copy As tool  to save data from an error view in any of several formats. For example, you can save it as a Purify data file (.pfy), with or without any messages you filtered out. Later, you can open the saved data file in Purify to analyze it or to compare it to future program runs. You can also save Purify data to an ASCII text (.txt) file.

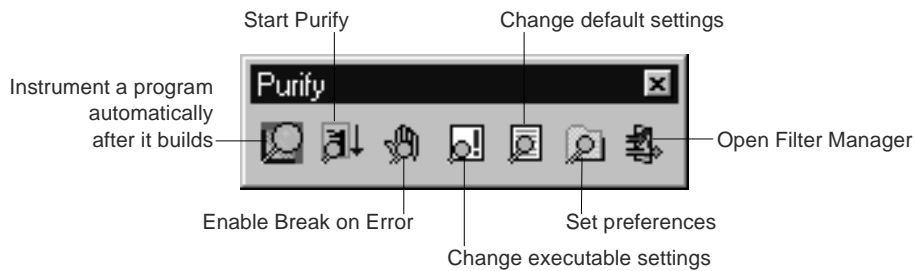
**More information?** Read “Sharing Purify data across the team” on page 21 of this manual. Also, look up *data, saving* in the online Help index.



## Using Purify's power features

### Integrating Purify into Microsoft Developer Studio 97

During installation, Purify is automatically integrated into Microsoft Developer Studio 97, providing one-stop error detection and correction. Purify adds a menu and a toolbar to Developer Studio to provide instant access to Purify's functionality.



You can build your program and run Purify on it from within Developer Studio. When Purify finds an error, just double-click the error line to open the source file in your editor at the exact location of the error.

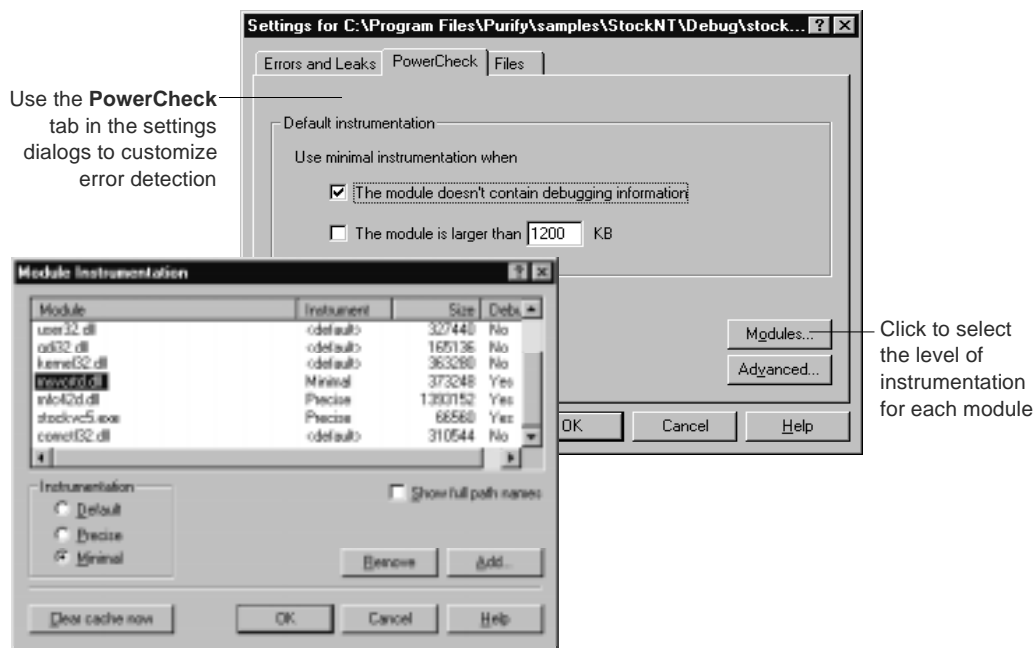
You can also add Purify toolbar icons to other Developer Studio toolbars or menus.

**More information?** Look up *Developer Studio, integrating* in the online Help index.

## Customizing error detection

Purify's default error detection is based on the size of each module in your program and the availability of debugging information. However, you can customize error detection for each module.

You can select *precise* instrumentation to get full run-time error detection, including Windows API checking, and to pinpoint problems in any component in your program. You can select *minimal* instrumentation for very quick instrumentation times.



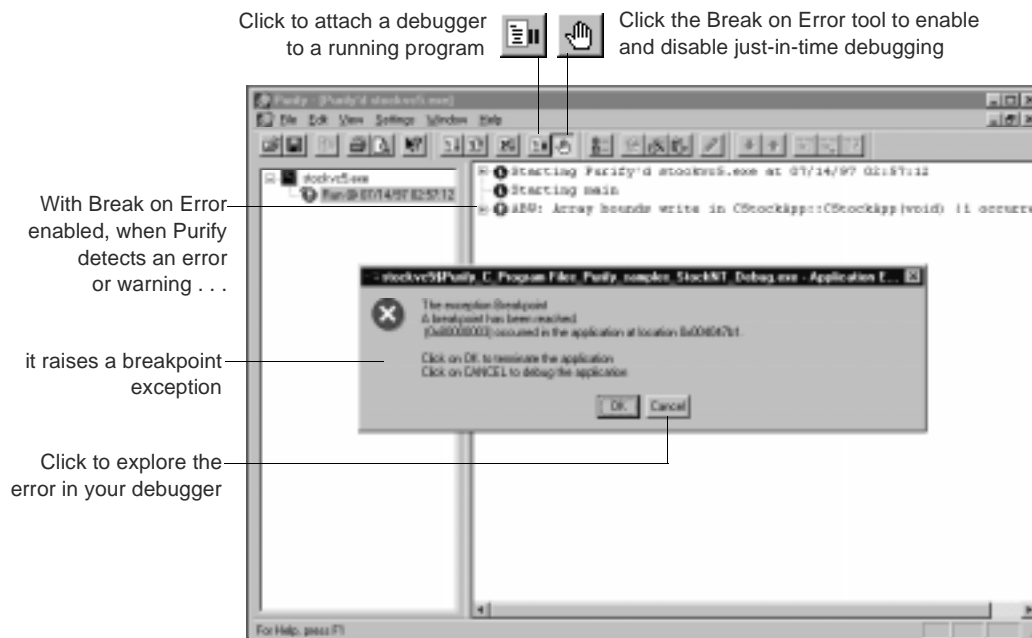
You might begin by using the *precise* setting for the most critical modules in your program and the *minimal* setting for the others. Later, you can change the minimal settings to precisely check the rest of the modules.

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

## 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. You can enable Break on Error to have Purify stop your program just before an error executes and let you start your debugger.

You can also attach your debugger to a running program at any time, or run a Purify'd program directly under your debugger.



To quickly debug *only* the most critical errors in your program, use Break on Error with Purify filters. First, filter out all the other messages, then enable Break on Error. Purify does not break for the filtered errors, only for the critical ones. When you're ready to debug the remaining errors, just disable the filters.

**More information?** Look up *debugger*, *using Break on Error* and *filters* in the online Help index.

## Extending error checking with Purify API functions

Purify includes a set of Application Programming Interface (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 your debugger. For example, you can call them interactively from the QuickWatch window in Microsoft Developer Studio 97.

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

## Using Purify in an existing test environment

Using Purify's command-line interface, you can use Purify with existing makefiles, batch files, or Perl scripts. For example, if you have a test script that runs a program, you can easily modify the script to run an instrumented version of the program. To do so, you might add this line to the beginning of your test script:

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

This line instructs Purify to save the original `Exename.exe` to a `.bak` file, then instrument `Exename.exe` without actually running it. Your test script then runs the instrumented 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.

## Sharing Purify data across the team

Purify saves you time during testing by making it easy to share information with other team members. For example, you can communicate program status more effectively by copying sections of error views and pasting them into email messages or bug reports, or by saving the information in error views as Purify data files (.pfy) or ASCII text files (.txt).

If your email program supports the MAPI interface standard, you can automatically send email messages containing Purify data files from within Purify.

You can also share filters, such as those for hiding messages about errors in common modules or third-party code for which you don't have the source files.



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 20  
  Windows API checking 6

### B

batch files 20  
Break on Error 19  
building programs in debug mode 7

### C

cache files 8  
call stack 13  
client/server applications 15  
code  
  *See* source code  
COM support 5  
command-line interface 20  
components  
  *See* modules  
count, repeat error 10  
Create Filter tool 11  
customizing  
  error detection 8, 18  
  message format 13

### D

data  
  files 21  
  saving 16  
debugging  
  data 7  
  just-in-time 19  
Developer Studio integration 17  
displaying hidden messages 11

### E

editor, opening from Purify 14  
email, and Purify data 21  
error detection, customizing 8, 18  
Error View window, overview 9  
errors  
  breaking on 19  
  correcting 14  
  display of repeated 10  
  *See also* messages  
exit messages 9

### F

files  
  caching instrumented 8  
  .pfy 16, 21  
  .txt 16, 21  
filters  
  Filter Manager 12  
  overview 11  
  sharing 12, 21  
functions, Purify API 20

### G

groups, filter 12

### H

handles  
  in use at exit 9  
  leaks 20  
hiding messages  
  *See* filters

### I

instrumentation  
  customizing 18  
  overview 8

## J

just-in-time debugging 19

## L

leaks

*See* memory

## M

mail

*See* email

makefiles 20

memory

leaks reported at exit 9

PurifyNewLeaks API function 20

menu, shortcut 10

messages

analyzing 13

customizing format 13

expanding 13

filtering 11

redisplaying hidden 11

*See also* errors

showing first only 10

Microsoft Developer Studio  
integration 17

minimal instrumentation 18

modules

custom error detection for 8, 18

support for 5

multi-process applications 15

## N

Navigator window, overview 15

## P

Perl scripts 20

.pfy file 16, 21

PowerCheck

tab 18

using with filters 12

precise instrumentation 8, 18

programs

instrumenting 7

rerunning 14

running from command line 20

running under debugger 19

## Q

QuickFilter tool 11

## R

release mode 7

repeat error count 10

rerunning a program 14

running programs 7

runs, comparing multiple 15

## S

saving data

from an error view 16

/SaveTextData option 20

sharing

data files 21

filters 12, 21

shortcut menu 10

source code

displayed in messages 13

editing 14

stack, call 13

## T

tests, using Purify in 20

threaded application support 5

toolbar, Purify 17

.txt file 16, 21

## W

windows

Error View 9

multiple error views 15

Navigator 15

Windows API checking 6