BoundsChecker Basics

BoundsChecker 5

Visual C++ Edition

Windows® 95
Windows NT®

NuMega™
Technologies
Making Software Work

March 1997

Information in this document is subject to change without notice and does not represent a commitment on the part of NuMega Technologies, Inc. The software described in this document is furnished under the software license agreement distributed with the product. The software may be used or copied only in accordance with the terms of the license. The purchaser may make one copy of the software for a backup, but no part of this user manual may be reproduced, stored in a retrieval system, or transmitted in any form or by any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use, without prior written permission from NuMega Technologies, Inc.

Software License Agreement

You are purchasing a license to use NuMega Technologies, Inc. software. The software is owned by and remains the property of NuMega Technologies, Inc., is protected by international copyrights, and is transferred to the original purchaser and any subsequent owner of the software media for their use only on the license terms set forth below. Opening the package and/or using the software indicates your acceptance of these terms. If you do not agree to all of the terms and conditions, or if after use you are dissatisfied with the software, return the software, manuals and any partial or whole copies within thirty days of purchase to the party from who you received it for a refund, subject to our restocking fee.

Use Of The Software: NuMega Technologies, Inc. ("NuMega"), grants the original purchaser ("Licensee") the limited rights to possess and use the NuMega Technologies, Inc. Software and User Manual ("Software") for its intended purposes. Licensee agrees that the Software will be used solely for Licensee's internal purposes, and that at any one time, the Software will be installed on a single computer only. If the Software is installed on a networked system, or on a computer connected to a files server or other system that physically allows shared access to the Software, Licensee agrees to provide technical or procedural methods to prevent use of the Software by more than one user.

One machine-readable copy of the Software may be made for BACK UP PURPOSES ONLY, and the copy shall display all proprietary notices, and be labeled externally to show that the back-up copy is the property of NuMega, and that use is subject to this License. Documentation may not be copied in whole or part.

Use of the Software by any department, agency or other entity of the U.S. Federal Government is limited by the terms of the following "Rider for Governmental Entity Users."

Licensee may transfer its rights under this License, PROVIDED that the party to whom such rights are transferred agrees to the terms and conditions of this Licensee, and written notice is provided to NuMega. Upon such transfer, Licensee must transfer or destroy all copies of the Software.

Except as expressly provided in this License, Licensee may not modify, reverse engineer, decompile, disassemble, distribute, sub-license, sell, rent, lease, give or in any way transfer, by any means or in any medium, including telecommunications, the Software. Licensee will use its best efforts and take all reasonable steps to protect the Software from unauthorized use, copying or dissemination, and will maintain all proprietary notices intact.

Limited Warranty And Indemnification: NuMega warrants the Software media to be free of defects in workmanship for a period of ninety days from purchase. During this period, NuMega will replace at no cost any such media returned to NuMega, postage prepaid. This service is NuMega's sole liability under this warranty.

NuMega agrees to indemnify and hold Licensee harmless from all loss, claim or damage to Licensee arising out of any claim, legal action, or suit alleging that the Software infringes a United States patent, copyright, or trade secret. NuMega will defend, at its own expense, any such claim or action against Licensee, and will pay all reasonable costs, expenses, and damages incurred by Licensee in connection therewith, including reasonable attorneys' fees and expenses incurred by Licensee, on condition that NuMega shall be immediately notified in writing by Licensee of any notice of such claim or action or pending claim or action known to Licensee prior to the time when the failure to deliver such notice has injured NuMega; and that NuMega shall have control of the defense against any such claim or action and all negotiations toward the settlement or compromise thereof. In such event, Licensee shall have the right to participate in any such action at NuMega's cost. If the Software becomes the subject of any claim, suit, or proceeding for infringement of any United States patent, copyright or any other right of a third party, or in the event of any adjudication that the Software infringes upon any United States patent, copyright or any other third party, or if the use or license of such Software is enjoined, in addition to other liability which may arise under this provision, NuMega shall at its sole expense and discretion either (a) obtain a license or any rights necessary to make the warranty contained herein true and correct, or (b) refund to Licensee any amounts paid to NuMega for the Software. Specifically excluded from the above covenant are any claims or actions based upon, or portions of claims or actions based upon, those portions of the Software modified or developed by Licensee or third parties.

Disclaimer: LICENSE FEES FOR THE SOFTWARE DO NOT INCLUDE ANY CONSIDERATION FOR ASSUMPTION OF RISK BY NUMEGA, AND NUMEGA DISCLAIMS ANY AND ALL LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR OPERATION OR INABILITY TO USE THE SOFTWARE, EVEN IF ANY OF THESE PARTIES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. FURTHERMORE, LICENSEE INDEMNIFIES AND AGREES TO HOLD NUMEGA HARMLESS FROM SUCH CLAIMS. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY THE LICENSEE. THE WARRANTIES EXPRESSED IN THIS LICENSE ARE THE ONLY WARRANTIES MADE BY NUMEGA AND ARE IN LIEU OF ALL OTHER WARRANTIES EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE.

THIS WARRANTY GIVES YOU SPECIFIED LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF WARRANTIES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Term: This License is effective as of the time Licensee receives the Software, and shall continue in effect until Licensee ceases all use of the Software and returns or destroys all copies thereof, or until automatically terminated upon the failure of Licensee to comply with any of the terms of this License.

General: This License is the complete and exclusive statement of the parties' agreement. Should any provision of this License be held to be invalid by any court of competent jurisdiction, that provision will be enforced to the maximum extent permissible, and the remainder of the License shall nonetheless remain in full force and effect. This License shall be controlled by the laws of the State of New Hampshire, and the United States of America.

Rider For U.S. Government Entity Users

This is a Rider to the above Software License Agreement, ("License"), and shall take precedence over the License where a conflict occurs.

1.The Software was: developed at private expense; no portion was developed with government funds; is a trade secret of NuMega and its licensor for all purposes of the Freedom of Information Act; is "commercial computer software" subject to limited utilization as provided in any contract between the vendor and the government entity; and in all respects is proprietary data belonging solely to NuMega and its licensor.

2.For units of the DOD, the Software is sold only with "Restricted Rights" as that term is defined in the DOD Supplement to DFAR 252.227-7013 (b)(3)(ii), and use, duplication or disclosure is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Manufacturer: NuMega Technologies, Inc. P.O. Box 7780, Nashua, New Hampshire 03060-7780 USA.

3.If the Software was acquired under a GSA Schedule, the Government has agreed to refrain from changing or removing any insignia or lettering from the Software or Documentation or from producing copies of manuals or disks (except for back up purposes) and; (1) Title to and ownership of the Software and Documentation and any reproductions thereof shall remain with NuMega and its licensor; (2) use of the Software shall be limited to the facility for which it is acquired; and (3) if the use of the Software is discontinued at the original installation and the Government wishes to use it at another location, it may do so by giving prior written notice to NuMega, specifying the new location site and class of computer.

4.Government personnel using the Software, other than under a DOD contract or GSA Schedule, are hereby on notice that use of the Software is subject to restrictions that are the same or similar to those specified above.

Contents

Some great men owe most of their greatness to the ability of detecting in those they destine for their tools the exact quality of strength that matters for their work.

◊ Joseph Conrad

1 The BoundsChecker Solution

The need for software development teams to produce quality software is greater than ever before. The complexity of software has grown geometrically and the opportunity for problems to develop is immense. Software defects can cripple a product, cause lengthy schedule delays, and ultimately cost the engineer, the development team, and the company dearly.

Developers spend most of their debugging time tracking down and repairing elusive bugs that were introduced early in development. The quality assurance staff does the bulk of the feature testing late in the development process when the schedule allows little time. Unfortunately, testing at this late stage usually focuses only on the outward functionality of the product. Testers run a GUI regression test bed, achieve exit criteria, and declare the product ready for shipping.

All too often, the product still possesses many hidden bugs that conventional testing techniques failed to identify. These bugs create customer dissatisfaction and poor product reputation. Updates, patches, and other expensive, embarrassing retroactive fixes cost time and money that could be spent more profitably and creatively on product improvement and new product development.

Check Early, Check Often—The BoundsChecker Philosophy

The solution to this problem is simple and quality assurance circles have been aware of it for years. To increase software quality, developers must thoroughly test their code early in the development process. Bugs must be caught and resolved as they are introduced to avoid surprises during integration, quality assurance, beta testing, and production. Briefly stated, "check early, check often."

Before BoundsChecker, this was easier said than done. Most developers need to spend the majority of their time writing code if they are to release a product on schedule. Unfortunately, few developers have the time or resources to test their products thoroughly as they develop them.

BoundsChecker provides the solution to this dilemma. BoundsChecker automates the crucial process of error-detection and analysis, identifies elusive bugs that are beyond the reach of traditional debugging and testing techniques, and adds little or no time to the development process.

Industry figures show that 50 percent of the development effort on an average project is spent on debugging. Regularly using BoundsChecker will significantly reduce the amount of time needed to debug your applications.

The Benefits of Using BoundsChecker

BoundsChecker is the most comprehensive, automated debugging solution available for C, and C++ development. As such, both individuals and organizations developing 32-bit Windows applications benefit from the following features:

- Comprehensive Error Detection
- Flexible Debugging Environment
- Integration Into the Visual C++ Debugger
- Advanced Error Analysis
- Windows Compliance Assurance
- Open Error-detection Architecture

The following sections highlight these benefits.

Comprehensive Error Detection

Unlike ordinary heap checkers that are limited to finding common memory errors, BoundsChecker is a sophisticated error-detection tool that validates the latest windows APIs including ActiveX, DirectX, OLE, COM, and ODBC. Additionally, BoundsChecker detects errors in executable files, dynamic link libraries, third-party modules, and OLE components. Of course, BoundsChecker also pinpoints static, stack, and heap errors, as well as, memory and resource leaks.

Best of all, using ActiveCheck technology, BoundsChecker does all this without requiring you to recompile or relink your program. Simply run your program under BoundsChecker and it automatically analyzes the internals of your program as it runs. BoundsChecker monitors your program's API calls, memory allocations and deallocations, windows messages, and other significant events, then uses this data to detect errors and to provide a complete trace of your program's execution. You can even check programs that do not have source code available. Since ActiveCheck requires no compilation or relinking overhead, you can use it daily to detect the following types of errors in Windows NT and Windows 95 programs:

API and OLE Errors	Memory Errors	Pointer and Leak Errors
• Windows function failed • Windows function not implemented • OLE Interface method failure • Invalid argument • Invalid OLE interface method argument • Questionable use of thread	• 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	• Interface leak • Memory leak • Resource leak • Unallocated pointer

In addition to ActiveCheck, BoundsChecker provides an error-detection technology called FinalCheck. A superset of ActiveCheck, FinalCheck finds all the errors ActiveCheck finds, plus these additional errors:

Memory Errors	Pointer and Leak Errors
• Reading overflows memory • Reading uninitialized memory • Writing overflows memory	• Array index out of range • Assigning pointer out of range • Expression uses dangling pointer • Expression uses unrelated pointers • Function pointer is not a function • Memory leaked due to free • Memory leaked due to reassignment • Memory leaked leaving scope • Returning pointer to local variable

To find these additional errors, FinalCheck uses a technique known as instrumentation. Instrumentation inserts error-detection code into an intermediate form of your program when you compile it, so BoundsChecker can view the structure of the application. Use FinalCheck for key project milestones and for detecting errors that are difficult to find.

Flexible Debugging Environment

BoundsChecker provides a flexible debugging environment. You can run BoundsChecker:

- With Microsoft Visual C++ as an integrated part of Microsoft Developer Studio

  Lets you benefit from all the features BoundsChecker provides while you continue to work completely within Microsoft Developer Studio. Work as you normally do; BoundsChecker works in the background. You can configure BoundsChecker settings, check your program, and review detected errors.

  BoundsChecker takes full advantage of the user interface within Microsoft Developer Studio 97, so you can use its existing Open, Close, Find, Save, and Print menu commands; toggle the BoundsChecker toolbar from the Toolbars tab; and access BoundsChecker Help from InfoView.

- As an independent application

  Lets you run BoundsChecker independent of Microsoft Developer Studio.

- From a DOS command line

  Lets you start BoundsChecker from a command line or automate a series of tests from a batch file.

Integration Into the Visual C++ Debugger

BoundsChecker provides a feature called Smart Debugging that automatically integrates BoundsChecker with the Visual C++ debugger within Microsoft Developer Studio. Smart Debugging enhances your debugger, so you no longer have to locate bugs yourself. Instead, BoundsChecker actively monitors all events and looks for errors as you step through your code. When BoundsChecker finds a problem, it displays the nature of the error. You can either position the debugger at the source line that contains the problem and view the error immediately or you can continue debugging and view the errors BoundsChecker found later.

Advanced Error Analysis

Windows is an event-driven environment in which much of your program is executed in response to Windows messages and other events. BoundsChecker intercepts control when events occur and logs them, so you can use them to see a complete history of events that led to a problem. BoundsChecker logs the following events:

- Windows messages and hooks. These events show how your program reacted to Windows messages.
- API calls and API returns along with argument information. These events define the order in which procedures are executed in your program.
- Output debug string messages from the program you are checking.
- Error messages, including all information BoundsChecker recorded in the event log.

Windows Compliance Assurance

To assure your program's ability to run on all Win32 variants, BoundsChecker creates compliance reports that identify calls specific to Windows NT, Windows 95, and Win32s. BoundsChecker also checks for undocumented Windows calls and displays your program's use of the C and C++ Run-Time Library, highlighting calls not supported by ANSI C.

Open Error-Detection Architecture

You can easily extend the 2500 standard API and OLE functions BoundsChecker validates to include APIs you create yourself. When you extend BoundsChecker to test your APIs, BoundsChecker automatically validates their parameters, validates their return values, and logs their trace data, so you can analyze it.

Where to Go From Here

This manual provides an overview of BoundsChecker and explains how to use its most commonly-used features. These include:

- Checking code
- Viewing data
- Setting error detection and reporting
- Checking compliance
- Logging and validating your own DLLs

TIP: If you use the Query feature within the Help system for Developer Studio 97, it automatically searches through the Help for BoundsChecker.

For detailed information, see the BoundsChecker on-line Help. To access a list of Help topics, do one of the following:

- If you are using Microsoft Developer Studio 97, use InfoView.
- If you are using Microsoft Developer Studio 4.x, click Tools, and then click BoundsChecker Help.

BoundsChecker also provides a hands-on tutorial to guide you through the process of checking code and analyzing data from within MicroSoft Developer Studio 97. On the Start menu, click programs, point to NuMega BoundsChecker, and then click BoundsChecker Tutorial to start the tutorial.

It is of the highest importance in the art of detection to be able to recognize out of a number of facts which are incidental and which are vital.

◊ Sir Arthur Conan Doyle

2 Checking and Analyzing Programs

BoundsChecker provides ActiveCheck technology to make checking your code so easy that every member of your development team can use BoundsChecker daily to test his or her code. ActiveCheck analyzes your executable image, DLLs, and OCXs as they execute, so you do not need to recompile or relink your program. Simply run the program under BoundsChecker, which works in the background to detect the following types of errors automatically.

Error	ActiveCheck Error Detection
API	ActiveCheck validates the parameters passed to and the values returned from over 2,500 API functions. The APIs ActiveCheck supports include DirectX, ODBC, Win32, WinSock, the C Run-Time Library, and internet APIs.
OLE	ActiveCheck detects invalid parameters and return codes for OLE Interface methods. ActiveCheck supports over 70 different OLE interfaces including the ActiveX interfaces. Additionally, ActiveCheck detects errors in reference counting. This is useful for detecting errors that occur when interfaces are not properly released after they are instantiated.
Pointer	Bad pointers frequently cause errors. To help you eliminate them, ActiveCheck checks for: Operations on null pointers. Operations on pointers that do not point to valid data. Attempts to free handles without unlocking them.

Error	ActiveCheck Error Detection
Leaks	ActiveCheck detects memory and resource leaks. Memory leaks occur when memory is allocated, but never freed. ActiveCheck detects memory leaks by using Windows memory allocation functions, such as HeapAlloc, GlobalAloc, and LocalAlloc, and standard C and C++ allocations including malloc and new. Resource leaks occur when windows specific resources, such as HMENU, HKEY, and HCURSOR, are allocated by your program, but not released back to the system. Resource leaks can consume excess memory and degrade system performance.
Memory	ActiveCheck detects overwriting and underwriting of memory in dynamically allocated memory, local or stack memory, and global or static memory.

Checking programs is easy, however, the specific steps to take depend on how you use BoundsChecker. The following sections detail how to check programs from:

- Microsoft Developer Studio
- the BoundsChecker application
- a DOS command line

Checking Programs Within Microsoft Developer Studio

BoundsChecker works with Visual C++ as an integrated part of Microsoft Developer Studio. Thus, enabling you to benefit from the advanced error-detection capabilities BoundsChecker provides without leaving your development environment.

Additionally, BoundsChecker enhances the Visual C++ debugger through Smart Debugging. Smart Debugging monitors all events and looks for errors as you step through code. When it encounters a problem, it displays the error and lets you choose whether to position the debugger at the source line that contains the error and view it immediately or continue debugging and view the errors later.

BoundsChecker integrates with Microsoft Developer Studio 4.0 and greater, with some variations between major versions. The following sections describe how to check code within Microsoft Developer Studio 97 and 4.x.

Using Microsoft Developer Studio 97

To use BoundsChecker to check a program from within Microsoft Developer Studio 97, do the following:

1 On the File menu, click Open Workspace to locate and open the program you want to check.

2 On the BoundsChecker menu, click Integrated Debugging if BoundsChecker integrated debugging is not enabled.

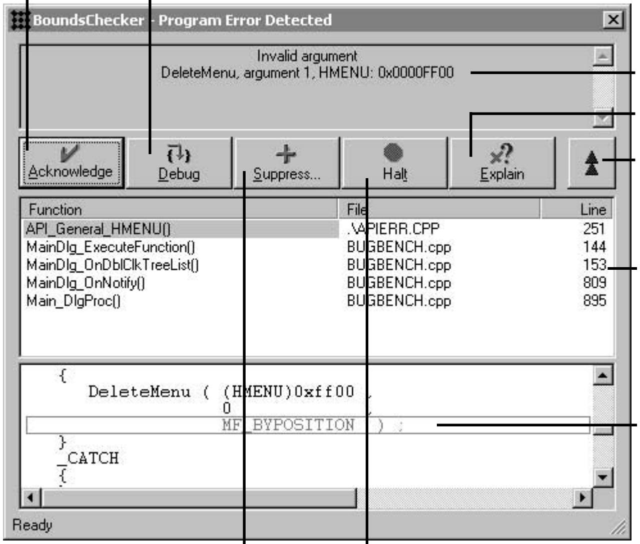
3 Use the standard debug commands to check your program. (For example, on the Build menu, point to Start Debug, and then click Go.)

BoundsChecker displays errors and events in the Program Results window as it detects them.

4 As you use your application, BoundsChecker works in the background. When BoundsChecker detects an error, it displays detailed information about the error.

Acknowledge the error.

Debug the error.



Invalid argument
DeleteMenu, argument 1, HMENU: 0x0000FF00

Function	File	Line
API_General_HMENU()	.\APIERR.CPP	251
MainDlg_ExecuteFunction()	BUGBENCH.cpp	144
MainDlg_OnDblClkTreeList()	BUGBENCH.cpp	153
MainDlg_OnNotify()	BUGBENCH.cpp	809
Main_DlgProc()	BUGBENCH.cpp	895

```
    {
      DeleteMenu ( (HMENU)0xff00 ,
                   0 ,
                   MF_BYPOSITION ) ;
    }
    _CATCH
    {
```

Bug description

Get Help on the error.

Hide the call stack and source code.

Call stack

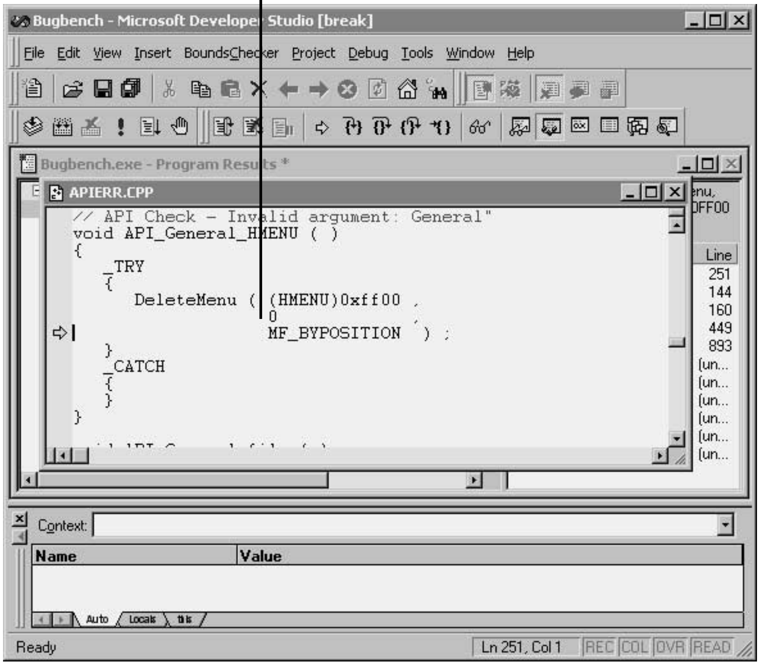
Source code that contains the error.

Stop your program.

Suppress the error from being reported.

Do one of the following:

- Click Debug to break into the debugger at the point in which the error occurred.



BoundsChecker automatically locates and displays the error.

Bugbench - Microsoft Developer Studio [break]
File Edit View Insert BoundsChecker Project Debug Tools Window Help
Bugbench.exe - Program Results *
APIERR.CPP
Line
251
144
160
449
893
(un...
(un...
(un...
(un...
(un...
(un...
Context:
Name
Value
Auto
Locals
this
REC COL OVR READ

```
// API Check - Invalid argument: General"
void API_General_HMENU ( )
{
   _TRY
   {
      DeleteMenu ( (HMENU)0xff00 ,
                   0             ,
                   MF_BYPOSITION ) ;
   }
   _CATCH
   {
   }
}
```

Ready

Ln 251, Col 1

On the Debug menu, click Go to resume your debugging session.

- Click Acknowledge to continue checking your program.
- Click Suppress if you do not want BoundsChecker to report the error again. BoundsChecker then lets you choose the circumstances under which it suppresses the error (within the function, within the source file, within the .EXE or DLL, or anywhere it occurs) and lets you add a remark. You can also save suppression information for future runs of the program.

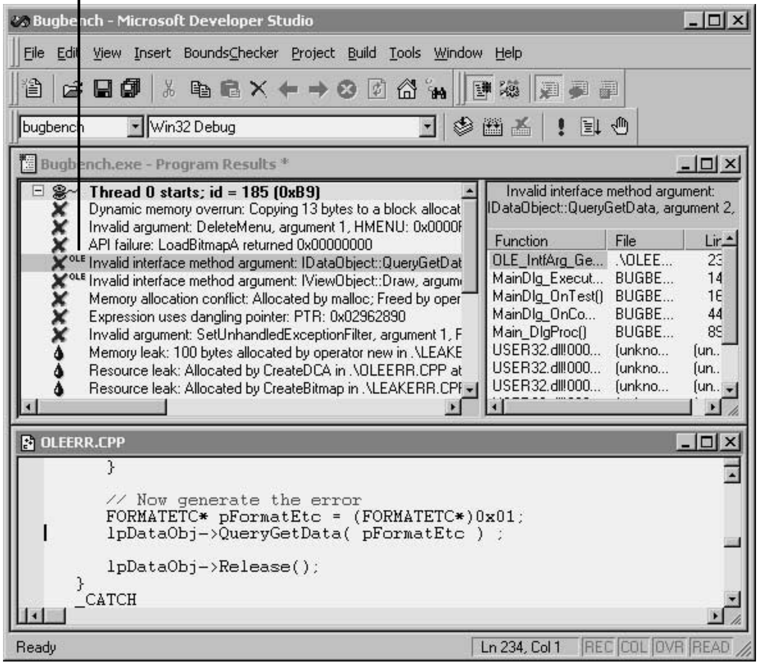
  There are two main reasons you might want to suppress an error:

  ◊ The error was generated by code belonging to another developer or by a third-party DLL or OCX.

  ◊ Your code properly handles the error. For example, BoundsChecker might detect an API failure that your code is able to handle.

When you finish checking your program, use the data in the Program Results window to analyze it. For example, double-click an error in the Program Results window to position the cursor at the location of the error. See *Viewing the Results of Your Error-detection Session* on page 24.

Double-click an error or leak to display the line of code that contains the error.


Bugbench - Microsoft Developer Studio
File Edit View Insert BoundsChecker Project Build Tools Window Help
bugbench
Win32 Debug
Bugbench.exe - Program Results *
OLEERR.CPP
Ready
Ln 234, Col 1
REC COL OVR READ

```
Thread 0 starts; id = 185 (0xB9)
Dynamic memory overrun: Copying 13 bytes to a block allocat
Invalid argument: DeleteMenu, argument 1, HMENU: 0x0000F
API failure: LoadBitmapA returned 0x00000000
OLE Invalid interface method argument: IDataObject::QueryGetDat
OLE Invalid interface method argument: IViewObject::Draw, argum
Memory allocation conflict: Allocated by malloc; Freed by oper
Expression uses dangling pointer: PTR: 0x02962890
Invalid argument: SetUnhandledExceptionFilter, argument 1, F
Memory leak: 100 bytes allocated by operator new in .\LEAKE
Resource leak: Allocated by CreateDCA in .\OLEERR.CPP at
Resource leak: Allocated by CreateBitmap in .\LEAKERR.CPF
```

Invalid interface method argument: IDataObject::QueryGetData, argument 2,

Function	File	Lir
OLE_IntfArg_Ge...	.\OLEE...	23
MainDlg_Execut...	BUGBE...	14
MainDlg_OnTest()	BUGBE...	16
MainDlg_OnCo...	BUGBE...	44
Main_DlgProc()	BUGBE...	89
USER32.dll!000...	(unkno...	(un..
USER32.dll!000...	(unkno...	(un..
USER32.dll!000...	(unkno...	(un..

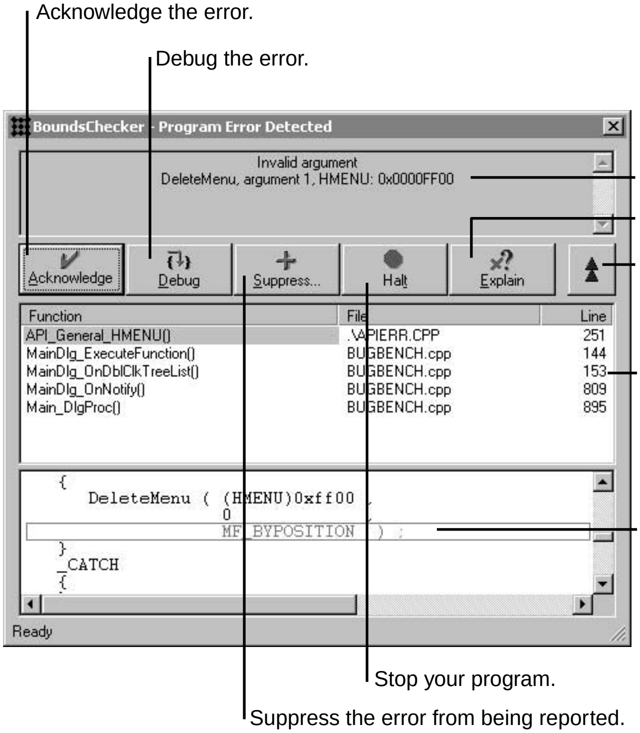
```
        }

        // Now generate the error
        FORMATETC* pFormatEtc = (FORMATETC*)0x01;
        lpDataObj->QueryGetData( pFormatEtc ) ;

        lpDataObj->Release();
    }
    _CATCH
```

Using Microsoft Developer Studio 4.x

To use BoundsChecker to check a program from within Microsoft Developer Studio 4.x, do the following:

1. On the File menu, click Open Workspace to locate and open the program you want to check.
2. If BoundsChecker integrated debugging is not enabled, click Tools, and then click BoundsChecker Integrated Debugging.
3. Use the standard debug commands to check your program. (For example, on the Build menu, point to Debug, and then click Go.)

4 As you use your application, BoundsChecker works in the background. When BoundsChecker encounters an error, it displays detailed information about the error.



BoundsChecker - Program Error Detected

Invalid argument
DeleteMenu, argument 1, HMENU: 0x0000FF00

Function	File	Line
API_General_HMENU()	.\APIERR.CPP	251
MainDlg_ExecuteFunction()	BUGBENCH.cpp	144
MainDlg_OnDblClkTreeList()	BUGBENCH.cpp	153
MainDlg_OnNotify()	BUGBENCH.cpp	809
Main_DlgProc()	BUGBENCH.cpp	895

```
    {
       DeleteMenu ( (HMENU)0xff00 ,
                    0             ,
                    MF_BYPOSITION  ) ;
    }
    _CATCH
    {
```

Acknowledge the error.

Debug the error.

Bug description

Get Help on the error.

Hide the call stack and source code.

Call stack

Source code that contains the error.

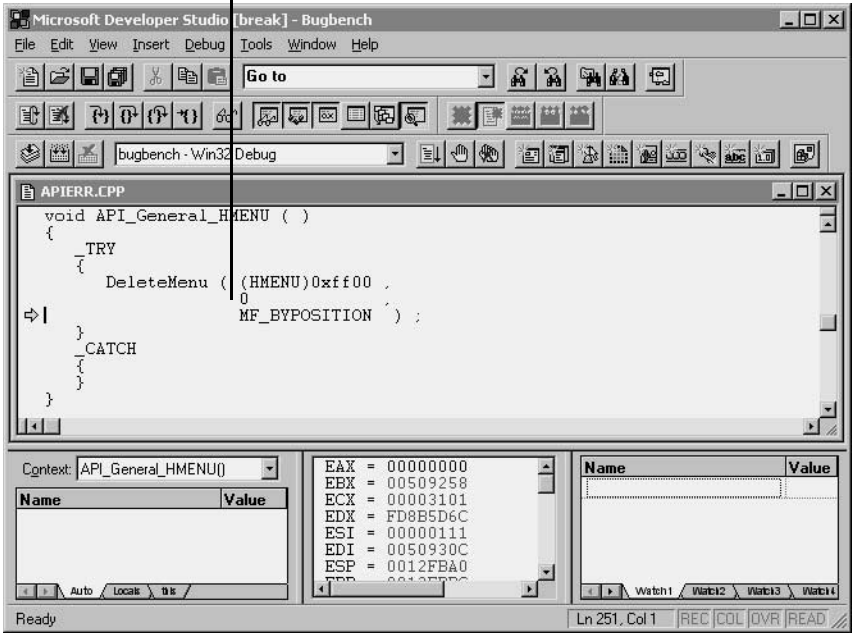
Stop your program.

Suppress the error from being reported.

Do one of the following:

- Click Debug to break into the debugger at the point in which the error occurred.

BoundsChecker automatically locates and displays the error.



```
void API_General_HMENU ( )
{
   _TRY
   {
      DeleteMenu ( (HMENU)0xff00 ,
                   0             ,
                   MF_BYPOSITION ) ;
   }
   _CATCH
   {
   }
}
```

```
EAX = 00000000
EBX = 00509258
ECX = 00003101
EDX = FD8B5D6C
ESI = 00000111
EDI = 0050930C
ESP = 0012FBA0
```

On the Debug menu, click Go to resume your debugging session.

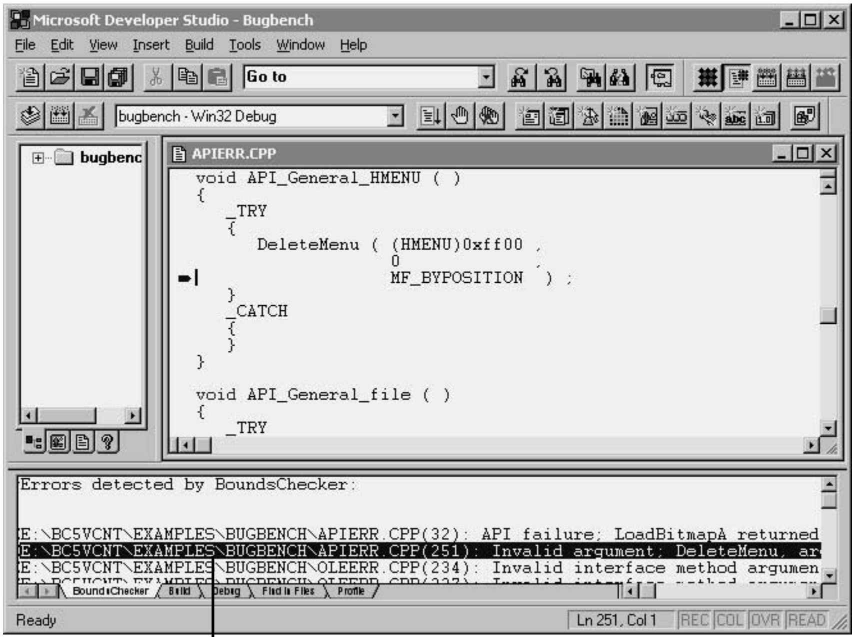
- Click Acknowledge to continue checking your program.
- Click Suppress if you do not want BoundsChecker to report the error again. BoundsChecker then lets you choose the circumstances under which it suppresses the error (within the function, within the source file, within the .EXE or DLL, or anywhere it occurs) and lets you add a remark. You can also save suppression information for future runs of the program.

  There are two main reasons you might want to suppress an error:

  ◊ The error was generated by code belonging to another developer or by a third-party DLL or OCX.

  ◊ Your code properly handles the error. For example, BoundsChecker might detect an API failure that your code is able to handle.

In the BoundsChecker tab within the Output window, BoundsChecker lists entries for all the errors and leaks it detects. To display the code that contains a particular error or leak, double-click the entry.



```
void API_General_HMENU ( )
{
   _TRY
   {
      DeleteMenu ( (HMENU)0xff00 ,
                   0          ,
                   MF_BYPOSITION  ) ;
   }
   _CATCH
   {
   }
}

void API_General_file ( )
{
   _TRY
```

```
Errors detected by BoundsChecker:

E:\BC5VCNT\EXAMPLES\BUGBENCH\APIERR.CPP(32): API failure; LoadBitmapA returned
E:\BC5VCNT\EXAMPLES\BUGBENCH\APIERR.CPP(251): Invalid argument; DeleteMenu, ar
E:\BC5VCNT\EXAMPLES\BUGBENCH\OLEERR.CPP(234): Invalid interface method argumen
```

Double-click an error or leak to display the line of code that contains the error.

BoundsChecker also reports program events, so you can analyze your program. See *Viewing the Results of Your Error-detection Session* on page 24 for more information about analyzing events.

Checking Programs With BoundsChecker

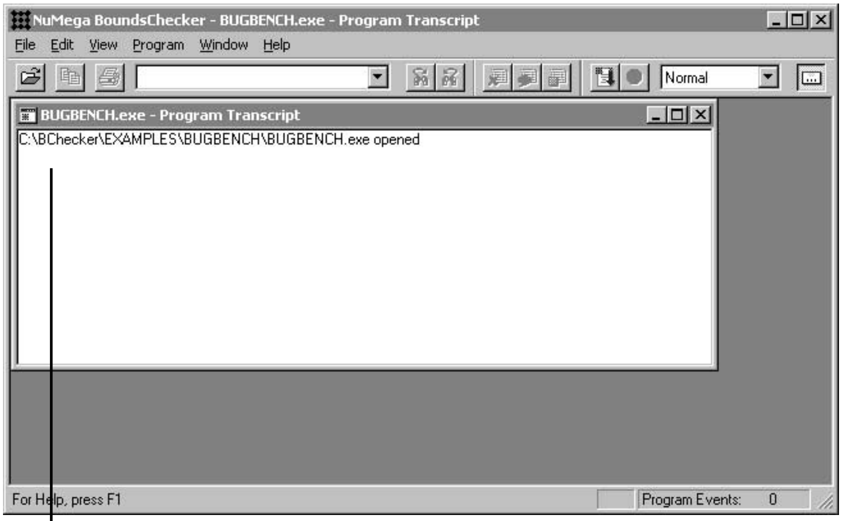
To check your program from the BoundsChecker application, do the following:

1 Click the Start button, and then point to Programs. Point to the folder that contains BoundsChecker, and then click BoundsChecker.

2 On the File menu, click Open.

3 Select the file you want to load and click Open.

BoundsChecker displays the Program Transcript window to log debugging events for the program you opened. This log is useful for determining which DLLs load when your program runs and for tracking output debug string messages to determine what the program does.


NuMega BoundsChecker - BUGBENCH.exe - Program Transcript
File
Edit
View
Program
Window
Help
Normal
BUGBENCH.exe - Program Transcript
For Help, press F1
Program Events: 0

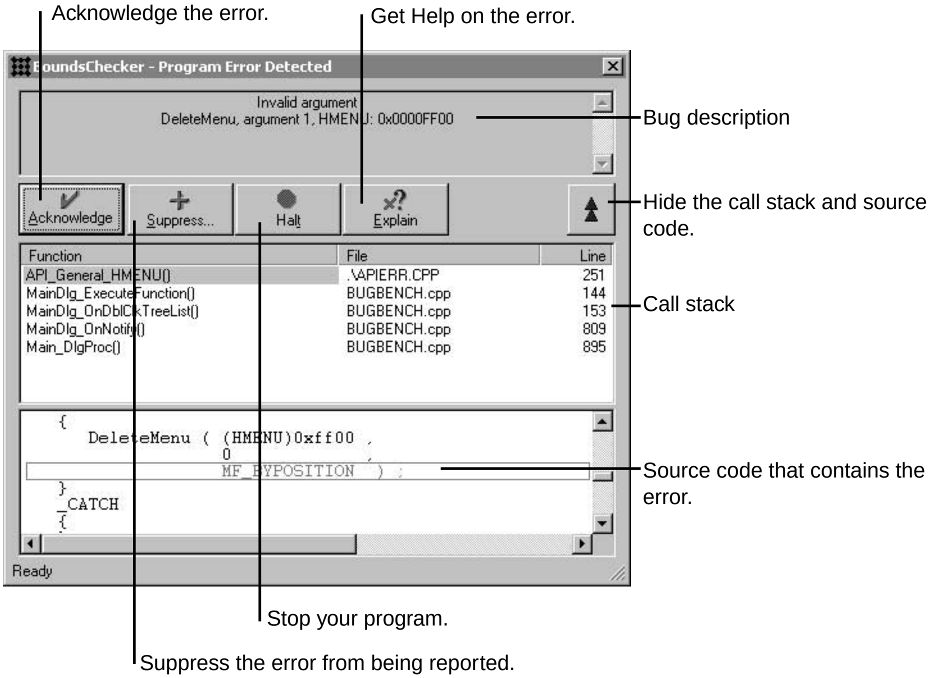
```
C:\BChecker\EXAMPLES\BUGBENCH\BUGBENCH.exe opened
```

Displays debugging events for your program.

4 On the Program menu, click Run.

BoundsChecker displays the Program Results window and starts your program. The Program Results window displays the errors and events BoundsChecker detects.

5 As you use your application, BoundsChecker works in the background. When BoundsChecker detects an error, it displays detailed information about the error.



Acknowledge the error.

Get Help on the error.

Bug description

Hide the call stack and source code.

Call stack

Source code that contains the error.

Stop your program.

Suppress the error from being reported.

Invalid argument
DeleteMenu, argument 1, HMENU: 0x0000FF00

Function	File	Line
API_General_HMENU()	.\APIERR.CPP	251
MainDlg_ExecuteFunction()	BUGBENCH.cpp	144
MainDlg_OnDblClkTreeList()	BUGBENCH.cpp	153
MainDlg_OnNotify()	BUGBENCH.cpp	809
Main_DlgProc()	BUGBENCH.cpp	895

```
    {
        DeleteMenu ( (HMENU)0xff00 ,
                     0             ,
                     MF_BYPOSITION  ) ;
    }
    _CATCH
    {
```

Do one of the following:

- Click Acknowledge to continue checking your program.
- Click Suppress if you do not want BoundsChecker to report the error again. BoundsChecker then lets you choose the circumstances in which it suppresses the error (within the function, within the source file, within the .EXE or DLL, or anywhere it occurs) and lets you add a remark. You can also save suppression information for future runs of the program.

  There are two main reasons you might want to suppress an error:

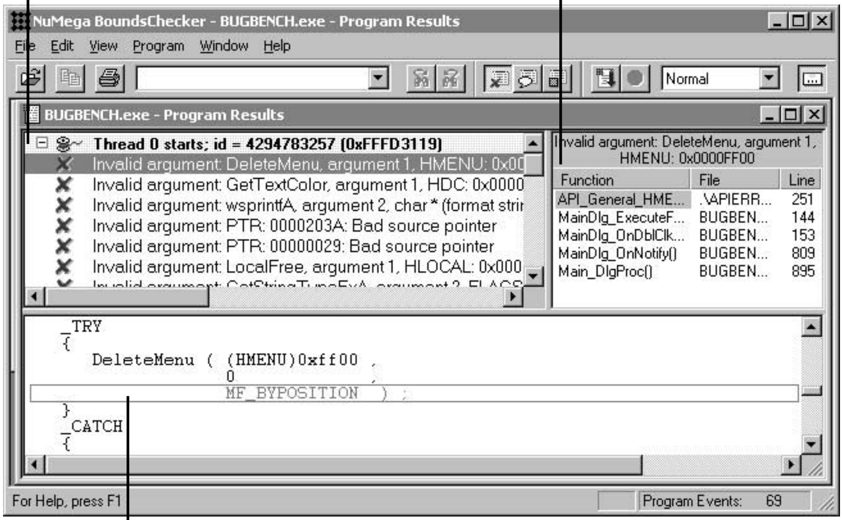
  ◊ The error was generated by code belonging to another member of the development team or by a third-party DLL or OCX.

  ◊ Your code properly handles the error. For example, BoundsChecker might detect an API failure that your code is able to handle.

When you are done checking your program, use the data in the Program Results window to analyze your program. For example, click an error in the Program Results window to display the line of code in which BoundsChecker detected the error. See *Viewing the Results of Your Error-detection Session* on page 24.

Displays errors and events that occur in your program.

Displays call stack data for the selected error.


NuMega BoundsChecker - BUGBENCH.exe - Program Results
File Edit View Program Window Help
Normal
BUGBENCH.exe - Program Results
For Help, press F1
Program Events: 69

Thread 0 starts; id = 4294783257 (0xFFFD3119)
Invalid argument: DeleteMenu, argument 1, HMENU: 0x00
Invalid argument: GetTextColor, argument 1, HDC: 0x0000
Invalid argument: wsprintfA, argument 2, char * (format stri
Invalid argument: PTR: 0000203A: Bad source pointer
Invalid argument: PTR: 00000029: Bad source pointer
Invalid argument: LocalFree, argument 1, HLOCAL: 0x000

Invalid argument: DeleteMenu, argument 1, HMENU: 0x0000FF00

Function	File	Line
API_General_HME...	.\APIERR...	251
MainDlg_ExecuteF...	BUGBEN...	144
MainDlg_OnDblClk...	BUGBEN...	153
MainDlg_OnNotify()	BUGBEN...	809
Main_DlgProc()	BUGBEN...	895

```
_TRY
{
   DeleteMenu ( (HMENU)0xff00 ,
                0            ,
                MF_BYPOSITION  ) ;
}
_CATCH
{
```

Displays source code for the selected error.

Starting BoundsChecker From the DOS Command Line

Start BoundsChecker from a DOS command line when you want to:

- Pass a file to BoundsChecker to open at initialization.
- Automate a series of tests from a batch file.

Once you are familiar with BoundsChecker, use the BC command from within a DOS session to start BoundsChecker. You can use the BC command with .BCP, .BCE, and .EXE files as follows:

```
BC [foo.bcp]

BC [foo.bce]

BC [foo.exe [argument1 argument2]]
```

BoundsChecker provides these optional switches.

Switch	Description
/B logfile	Run BoundsChecker in batch mode. All operations are executed with no user input required. The results are saved in "logfile." This switch overrides /L, /M, and /S.
/L	Disable start-up splash screen.
/M	Start BoundsChecker minimized.
/S	Disable immediate error reporting.
/W<dir>	Specify the working directory. The directory path must immediately follow the /w argument. Do not use a space to separate the directory path from the argument.

Viewing the Results of Your Error-detection Session

BoundsChecker intercepts control when errors or events occur and logs them to the Program Results window. After checking your program, use the Program Results window to see a complete history of the events that led to a problem.

If you are using either Microsoft Developer Studio 97 or the BoundsChecker application, the Program Results window displays automatically. If you are using Microsoft Developer Studio 4.x, do the following to display the Program Results window:

1. On the Tools menu, click BoundsChecker to start the BoundsChecker application.

   You will use the Program Results window within the BoundsChecker application to review the results of your error-detection session.

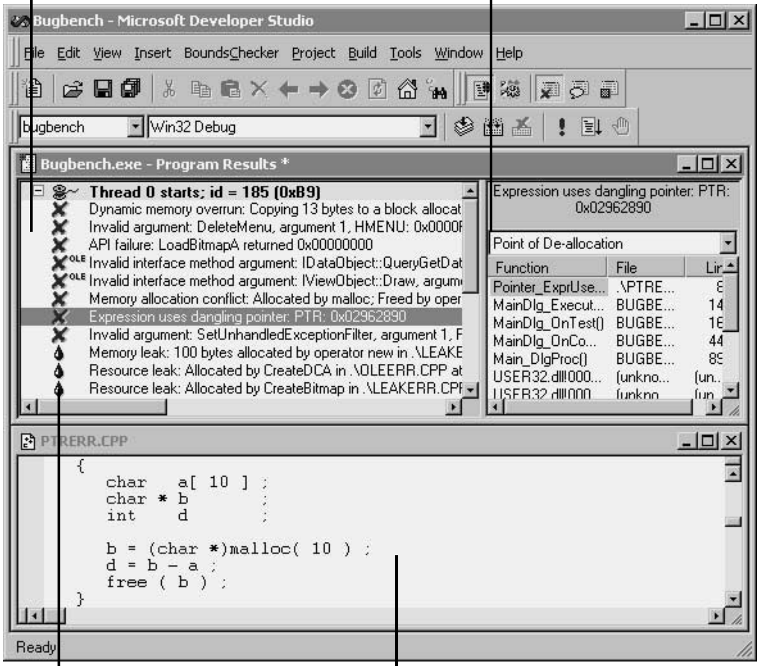
2. On the File menu within BoundsChecker, click Open and select your program's corresponding .BCE file.

   The .BCE file is the error and event file BoundsChecker created when it checked your program. By default, the .BCE file is located in the directory with your program's .EXE.

The following figure shows a typical Program Results window in Microsoft Developer Studio 97.

Lists detected errors and events.

Lists call stacks for the selected error.


Bugbench - Microsoft Developer Studio
File Edit View Insert BoundsChecker Project Build Tools Window Help
bugbench
Win32 Debug
Bugbench.exe - Program Results *
PTRERR.CPP
Ready

Thread 0 starts; id = 185 (0xB9)
Dynamic memory overrun: Copying 13 bytes to a block allocat
Invalid argument: DeleteMenu, argument 1, HMENU: 0x0000F
API failure: LoadBitmapA returned 0x00000000
OLE Invalid interface method argument: IDataObject::QueryGetDat
OLE Invalid interface method argument: IViewObject::Draw, argum
Memory allocation conflict: Allocated by malloc; Freed by oper
Expression uses dangling pointer: PTR: 0x02962890
Invalid argument: SetUnhandledExceptionFilter, argument 1, F
Memory leak: 100 bytes allocated by operator new in .\LEAKE
Resource leak: Allocated by CreateDCA in .\OLEERR.CPP at
Resource leak: Allocated by CreateBitmap in .\LEAKERR.CPI

Expression uses dangling pointer: PTR: 0x02962890

Point of De-allocation

Function	File	Lir
Pointer_ExprUse...	.\PTRE...	8
MainDlg_Execut...	BUGBE...	14
MainDlg_OnTest()	BUGBE...	16
MainDlg_OnCo...	BUGBE...	44
Main_DlgProc()	BUGBE...	89
USER32.dll!000...	(unkno...	(un..
USER32.dll!000	(unkno	(un

```
{
   char   a[ 10 ] ;
   char * b       ;
   int    d       ;

   b = (char *)malloc( 10 ) ;
   d = b - a ;
   free ( b ) ;
}
```

Indicates the type of error.

Displays source code for the selected error.

Examining Errors

BoundsChecker places a wealth of information at your fingertips. With BoundsChecker you can easily view the following data about each error:

- The line in the source code in which BoundsChecker detected the error.
- The error's corresponding call stack.
- The source code for any function in the call stack.
- The point at which memory is allocated (for errors that involve a memory block that is allocated elsewhere).
- On-line Help for the error

Displaying Source Code and Call Stack Data

To display the following source code and call stack information for an error, click the error in the Program Results window. If you are using Microsoft Developer Studio 97, double-click the error in the Program Results window.

- The source code in which BoundsChecker detected the error.

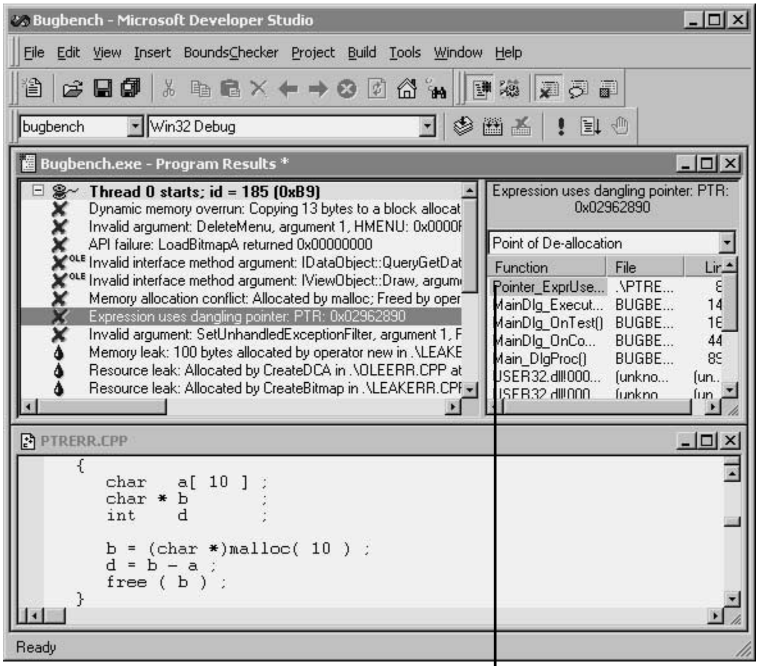
  BoundsChecker highlights the line that contains the error by framing it and displaying it in red.

- The error's call stack.

  BoundsChecker lists each function in the stack, the file in which the function is located, and the line on which the function is found.

Using the Call Stack

The stack frame lets you display the source code for any function in the stack. This is useful for seeing the events that led to the error. If the error involves a memory block that is allocated elsewhere, the stack frame also lets you view the point at which the memory is allocated



```
Thread 0 starts; id = 185 (0xB9)
    Dynamic memory overrun: Copying 13 bytes to a block allocat
    Invalid argument: DeleteMenu, argument 1, HMENU: 0x0000F
    API failure: LoadBitmapA returned 0x00000000
OLE Invalid interface method argument: IDataObject::QueryGetDat
OLE Invalid interface method argument: IViewObject::Draw, argum
    Memory allocation conflict: Allocated by malloc; Freed by oper
    Expression uses dangling pointer: PTR: 0x02962890
    Invalid argument: SetUnhandledExceptionFilter, argument 1, F
    Memory leak: 100 bytes allocated by operator new in .\LEAKE
    Resource leak: Allocated by CreateDCA in .\OLEERR.CPP at
    Resource leak: Allocated by CreateBitmap in .\LEAKERR.CPF
```

Expression uses dangling pointer: PTR: 0x02962890

Point of De-allocation

Function	File	Lir
Pointer_ExprUse...	.\PTRE...	8
MainDlg_Execut...	BUGBE...	14
MainDlg_OnTest()	BUGBE...	16
MainDlg_OnCo...	BUGBE...	44
Main_DlgProc()	BUGBE...	89
USER32.dll!000...	(unkno...	(un..
USER32.dll!000	(unkno	(un

```
{
   char   a[ 10 ] ;
   char * b       ;
   int    d       ;

   b = (char *)malloc( 10 ) ;
   d = b - a ;
   free ( b ) ;
}
```

Double-click a function to view its corresponding source code.

To view a particular function, click the corresponding function in the stack. If you are using Microsoft Developer Studio 97, double-click the function. If the error deals with memory that was allocated either from the heap or from earlier on the call stack, you can choose one of the following before selecting a function:

- Location of Error

  Lists the functions that led to the error.
- Point of Allocation

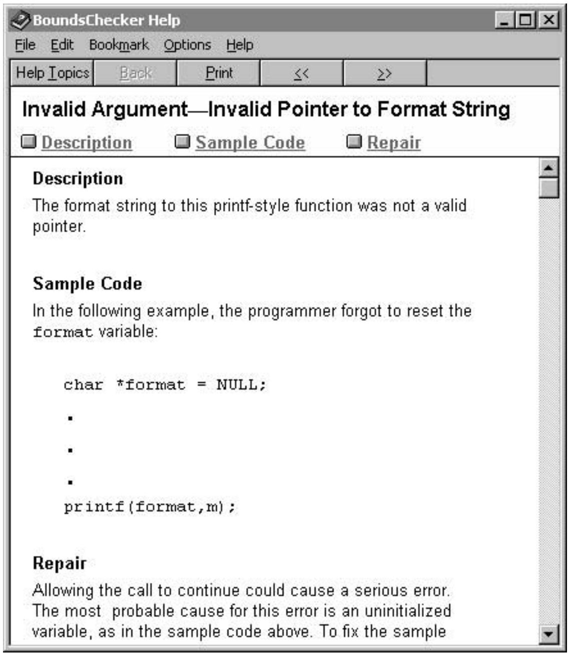
  Lists the functions where memory is allocated.
- Point of Deallocation

  Lists the functions where memory is freed.

Displaying Help for the Error

BoundsChecker provides the following Help for each type of error it detects:

- A complete description of the error.
- Sample error code.
- Suggestions for correcting the error



Invalid Argument—Invalid Pointer to Format String

Description

The format string to this printf-style function was not a valid pointer.

Sample Code

In the following example, the programmer forgot to reset the `format` variable:

```
char *format = NULL;
.
.
.
printf(format,m);
```

Repair

Allowing the call to continue could cause a serious error. The most probable cause for this error is an uninitialized variable, as in the sample code above. To fix the sample

To display Help for a particular error, do the following:

1. Click the error on which you need Help.
2. Click the right mouse button, and then click Explain.

Suppressing Errors

You can suppress an error while you check your program or after you analyze it in the Program Results window. Suppressing an error prevents BoundsChecker from reporting it again. You might want to suppress an error if:

- The error was generated by code from another developer or from a third-party DLL or OCX.
- Your code properly handles the error.

To suppress an error, do the following:

1. Click the error you want to suppress.
2. Click the right mouse button, and then click Suppress.
3. Select one of the following suppression options:
   - Suppress this Error Only When it Occurs in This Function
   - Suppress this Error Only When it Occurs in This Source File
   - Suppress this Error Only When it Occurs in This EXE or DLL
   - Suppress this Error Regardless of Where it Occurs
4. If you want BoundsChecker to suppress the error automatically the next time you check the program, select Save Suppression Information. Otherwise, BoundsChecker only suppresses this error when you display the results of this error-detection session.
5. If you want to add a notation to the error you are suppressing, add a remark in the text box.

When you suppress an error, it appears dimmed in the Program Results window. BoundsChecker adds the suppressed error to the list it maintains in the Error Suppression tab within the program settings. See *Customizing Error Suppression Settings* on page 37 for information about removing errors from the suppression list.

Changing the Results View

By default, the Program Results window displays errors, threads, and leaks only. However, you can change the type of data it displays by changing its view. If you are using BoundsChecker, click one of the following settings on the View menu to change the Results view. If you are using Microsoft Developer Studio 97, click BoundsChecker, point to View, and then click one of the following settings:

- Show Errors and Leaks Only

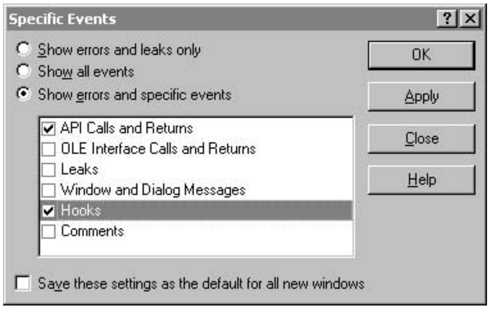
  Displays errors, threads, and leaks.

- Show All Events

  Displays errors, threads, leaks and all events.

- Show Errors and Specific Events

  Displays errors, threads, and specific events. To determine the specific events it displays, click Specific Events on the View menu and select the events you want to view. The following example instructs the Program Results window to display API calls and returns and hooks in addition to errors.



Specific Events

○ Show errors and leaks only
○ Show all events
● Show errors and specific events

☑ API Calls and Returns
☐ OLE Interface Calls and Returns
☐ Leaks
☐ Window and Dialog Messages
☑ Hooks
☐ Comments

☐ Save these settings as the default for all new windows

Note: The Error Detection and Event Reporting program settings determine the type of errors and events that BoundsChecker detects and reports. See *Chapter 3: Customizing Error Detection and Reporting* on page 33.

The Program Results window uses the following icons to represent errors and events:

Icon	Error or Event Type	Description
◆	Call-return	BoundsChecker adds a Call-return event to the event log when your program makes an API or OLE call and then returns from the call.
◆ OLE	OLE Call-return	Call-returns that contain nested events are designated with a plus sign. To see the nested events, click the plus sign to expand the event. When you expand a Call-return event, BoundsChecker uses Expanded Call and Expanded Return icons to designate the individual calls and returns.
➡	Expanded Call	
➡ OLE	OLE Expanded Call	
⬅	Expanded Return	
⬅ OLE	OLE Expanded Return	
✎	Comment	BoundsChecker adds a Comment event to the event log when your program makes a call to OutputDebugString.
✖	Error	BoundsChecker adds an Error event to the event log when it catches an error in your program.
✖ OLE	Ole Error	
⚓	Hook	BoundsChecker adds a Hook event to the event log when the program processes a Windows hook call. The function name and arguments are included on the line.
💧	Leak	BoundsChecker adds a Leak event to the event log for each memory, resource, or interface method leak it finds. The message describes the leak.
💧 OLE	OLE Leak	
⊞	Message	BoundsChecker adds a Message event to the event log when the program processes a dialog or Windows message.
∿	Start of Thread	BoundsChecker adds a start of thread event to the event log when it detects the creation of a thread.
⇅	Thread Context Switch	BoundsChecker adds a Thread Context Switch event to the event log when it detects that your program has switched from one thread to another.

Printing Your Results

On the File menu, click Print to print the contents of the Program Results window.

Saving Your Results

To save the results of your error-detection session to view later, do the following:

1. If your application is running, quit your application.
2. On the File menu, click Save As.
3. Enter a file name and select the location in which you want to save the file.

   By default, BoundsChecker saves the file in the directory that contains the executable.

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore, all progress depends on the unreasonable man.

◊ George Bernard Shaw

3 Customizing Error Detection and Reporting

BoundsChecker provides a series of program settings that let you determine how it detects and reports errors and events. These program settings control the following:

Program Setting	Description
Error Detection	Determines the types of errors BoundsChecker detects and reports.
Event Reporting	Determines if BoundsChecker collects and reports data about calls your program makes to libraries and Windows APIs.
Program Info	Determines the program search path and directory BoundsChecker uses to locate your program files and establishes program parameters to pass as command-line arguments.
Error Suppression	Determines if BoundsChecker reports errors in specific libraries and instances.
Modules and Files	Determines the modules BoundsChecker checks.

Customizing Program Settings

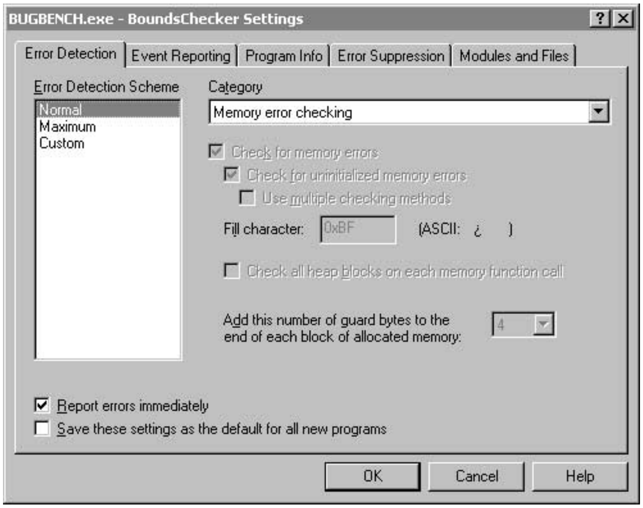
To modify the program settings, do the following:

1. Do one of the following:
   - If you are using Microsoft Developer Studio 97, click BoundsChecker, and then click Settings.
   - If you are using Microsoft Developer Studio 4.x, click Tools, and then click BoundsChecker Settings.
   - If you are using the BoundsChecker application, click Program, and then click Settings.
2. Click the tab for the settings you want to modify.

   The sections that follow highlight the settings for each of these tabs.
3. When you finish modifying the settings, click OK to save your changes.

Customizing Error Detection Settings

The following Error Detection settings determine how BoundsChecker detects and reports errors.



BUGBENCH.exe - BoundsChecker Settings

Error Detection Scheme: Normal

Maximum

Custom

Category: Memory error checking

☑ Check for memory errors

☑ Check for uninitialized memory errors

□ Use multiple checking methods

Fill character: 0xBF (ASCII: ¿)

□ Check all heap blocks on each memory function call

Add this number of guard bytes to the end of each block of allocated memory: 4

☑ Report errors immediately

□ Save these settings as the default for all new programs

Error Detection Scheme

To get you up and running as quickly as possible while offering you optimum flexibility, BoundsChecker provides three Error Detection Schemes: Normal, Maximum, and Custom. Normal and Maximum preset the level of error detection BoundsChecker provides and Custom lets you specify your preferred level of error detection. Use these schemes as follows:

TIP: To see the difference between Normal and Maximum, switch between the modes and observe how the options are set.

- Normal

  Performs core error detection for all modules that have debug information. To maximize performance, Normal collects only the information essential for diagnosing the problem. Select Normal when you do not want a high level of detail or when you need to maximize performance. Normal is the default.

- Maximum

  Performs the highest level of error detection. Maximum even checks third-party code, including modules that do not contain debug information. Maximum also reports all instances of errors that are a result of other errors. Select Maximum when you want to:

  - Collect as much information as possible about an error.
  - Find all occurrences of an error.
  - See errors in third-party modules that do not have debug information.

- Custom

  Provides the greatest flexibility by letting you specify the types of errors BoundsChecker detects and the extent to which they are reported. By default, the value of the Custom options are set to Normal. Change these options as needed.

For information about controlling the events BoundsChecker reports, see*Customizing Event Reporting Settings* on page 36.

Report Errors Immediately

Determines if BoundsChecker automatically displays the Program Error Detected window each time it encounters an error in your program. Displaying the Program Error Detected window is useful for seeing errors in context. If you prefer, clear this setting to check your program without interruption. BoundsChecker always maintains a log of your error-detection session, so you can see your program's errors and events at your convenience.

Save These Settings as the Default for all New Programs

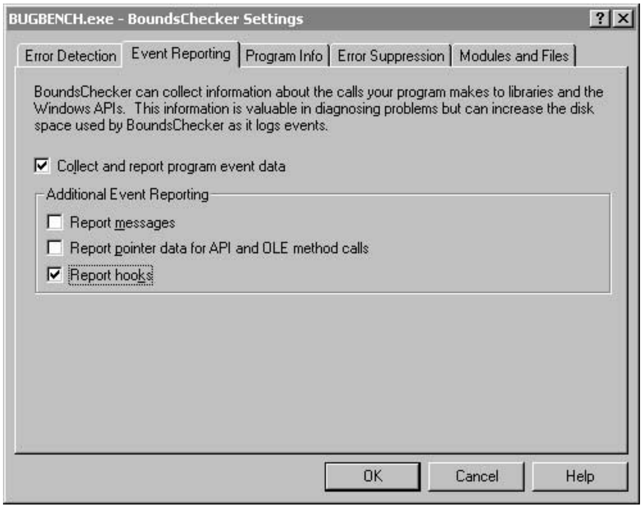
Depending on your development environment, you may want to permanently modify your changes to the Error Detection settings. Select this setting to apply your modifications to all subsequent programs you check with BoundsChecker.

Customizing Event Reporting Settings

Event reporting instructs BoundsChecker to collect all the Windows API calls, parameters, and messages your program sends and receives. This helps you debug your program, by showing you what is happening. Use event reporting to solve the following problems:

Problem Area	Suggested Analysis
Sequences	Examine messages and how your program responds to them. For instance, did the messages come in the order you expected? Check the API calls your program made in response to messages.
Performance	Look for indications of wasted time. For instance, is your program painting a window twice in succession on two different messages? Your program may be making hundreds or even thousands of unnecessary memory allocations or file reads. You can block these allocations into a few big operations to improve performance.
Threads	Look at the thread-switching and thread interaction. This helps you debug multi-thread problems with semaphores in critical sections.
API failures	Look at the arguments passed to APIs. When pointers are passed, trace the data to which they point.

By default, BoundsChecker does not enable event reporting, because collecting detailed event data can increase the amount of disk space BoundsChecker uses and can affect system performance. If you want BoundsChecker to collect this data, select Collect and Report Program Event Data, and then select the types of events you want to collect.



BoundsChecker can collect information about the calls your program makes to libraries and the Windows APIs. This information is valuable in diagnosing problems but can increase the disk space used by BoundsChecker as it logs events.

☑ Collect and report program event data

Additional Event Reporting

□ Report messages

□ Report pointer data for API and OLE method calls

☑ Report hooks

Customizing Program Information Settings

Use the Program Info settings to establish the following for your program:

- Working Directory
- Command Line Arguments
- Source File Search Path

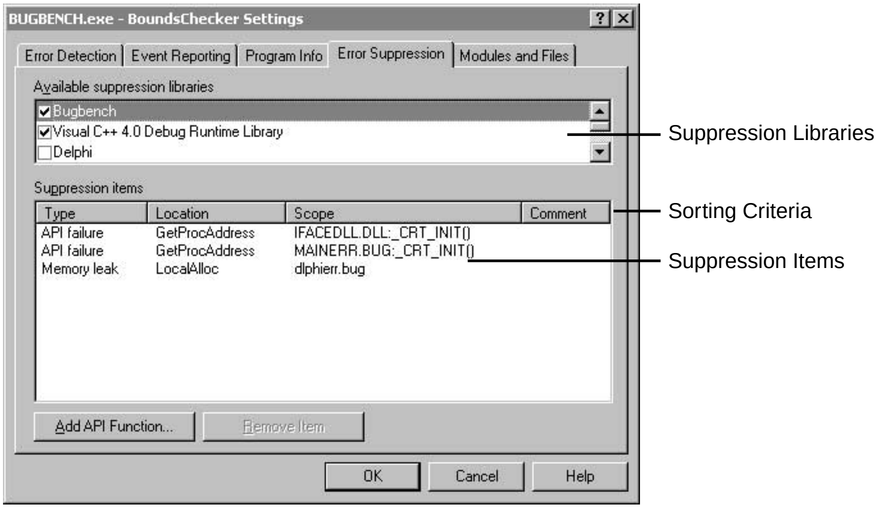
Customizing Error Suppression Settings

Each time you suppress an error, BoundsChecker adds the suppression information to the suppression library. BoundsChecker uses the suppression library to determine which errors to suppress for future runs of the program. In addition to the individual program libraries, BoundsChecker supplies suppression libraries for common DLLs, including MFC and OWL, and the Delphi VCL.

As your work progresses, you may want to delete a suppression item, or you may want BoundsChecker to check your program without referring to the library. Perform the actions in the following table to view and modify libraries and suppression items.

To	Do this
Display the list of suppression items for a particular library.	Select the name of the library.
Enable or disable a library.	Select or clear the checkbox for the library.
Sort the suppression items in a library.	Click a sorting criterion above the list of items.
Delete a suppression item.	Select the item from the list, and then click Remove.

Note: You can only delete suppression items from your program library.



BUGBENCH.exe - BoundsChecker Settings

Available suppression libraries

☑ Bugbench
☑ Visual C++ 4.0 Debug Runtime Library
□ Delphi

Suppression items

Type	Location	Scope	Comment
API failure	GetProcAddress	IFACEDLL.DLL:_CRT_INIT()	
API failure	GetProcAddress	MAINERR.BUG:_CRT_INIT()	
Memory leak	LocalAlloc	dlphierr.bug	

Suppression Libraries

Sorting Criteria

Suppression Items

Customizing Modules and Files Settings

BoundsChecker automatically checks all the source files for your .EXE and its related static and run-time DLLs and OCXs. However, you might want to check only a specific portion of your code. For example, you might want to limit error detection to a specific module or source files that comprise a module.

To limit the code BoundsChecker checks, clear the modules or source files you do not want to check. Note that run-time DLLs and OCXs are not listed, but are automatically checked. To avoid checking these modules, click Add to locate and add them to the list of modules, then clear their associated check boxes.

Every man at the bottom of his heart believes that he is a born detective.

◊ John Buchan

4 Detecting Errors With FinalCheck

In addition to ActiveCheck, BoundsChecker provides an even more exhaustive error-detection technology called FinalCheck. A superset of ActiveCheck, FinalCheck finds all the errors ActiveCheck finds, plus these additional errors:

Memory Errors	Pointer and Leak Errors
• Reading overflows memory • Reading uninitialized memory • Writing overflows memory	• Array index out of range • Assigning pointer out of range • Expression uses dangling pointer • Expression uses unrelated pointers • Function pointer is not a function • Memory leaked due to free • Memory leaked due to reassignment • Memory leaked leaving scope • Returning pointer to local variable

To find these particularly elusive and deeply-rooted errors, FinalCheck uses a technique known as instrumentation. Instrumentation inserts error-detection code into your program when you compile it. To accomplish this, FinalCheck intercepts the immediate language generated by the compiler's front end, adds additional error-detection code, and passes it to the back end. BoundsChecker performs this process transparently. When you run your program, BoundsChecker uses this error-detection code to find the additional errors.

To benefit from the additional error detection FinalCheck provides, you must recompile your program. FinalCheck can also cause your program to run a little slower as it checks for additional errors. Therefore, it is recommended that you use ActiveCheck regularly during the development cycle and reserve FinalCheck for key project milestones and for detecting elusive errors.

Using FinalCheck

To use FinalCheck, do the following:

- Create a new project configuration from within Microsoft Developer Studio
- Build the program from within Microsoft Developer Studio or the command line.

The following sections describe how to create a project configuration and how to build a program.

Creating a Project Configuration

Although it is not required, NuMega recommends that you create an additional project configuration (target) for BoundsChecker before you build your project. Using a project configuration dedicated to BoundsChecker lets you use:

- A debug configuration to reduce compile time when you do not want to instrument modules in compilation.
- An instrumentation configuration to fully instrument modules in compilation.
- A release configuration to produce the final product.

To create a project configuration for BoundsChecker, do the following:

1. On the File menu, open a project in Visual C++.
2. On the Build menu, click Configurations.
3. Click Add to create a new project configuration.
4. In the Configuration field, type a name for the configuration.
5. In the Copy Settings From: list, select the existing debug target and click OK.

   The existing debug target is typically indicated as follows: *program name* - Win32 Debug.
6. Click Close.

Building a Program

After you create a project configuration for BoundsChecker, build your program. The following sections describe how to build your program from within Microsoft Developer Studio and from a command line.

Building a Program From Microsoft Developer Studio

To build your program from Microsoft Developer Studio, do the following:

1. In the File menu, open the project you want to build.

2. If you are using Microsoft Developer Studio 97, click BoundsChecker, and then select Instrument Builds. If you are using Microsoft Developer Studio 4.x, go to Step 3.

3. Select your BoundsChecker project configuration.

   In Microsoft Developer Studio 97, your project is listed in the Active Project list and in Microsoft Developer Studio 4.x it is listed in the Default Project Configuration list.

4. If you want to refine the build process, BoundsChecker provides several NMCL command-line options that you can use within Microsoft Developer Studio. See *Building a Program for FinalCheck From the Command Line* on page 41 for a complete list of options. To use an option:

   - On the build menu click settings, and then click C/C++.
   - Type the option in the Project Options field.

5. Build your program, as follows:

   - If you are using Microsoft Developer Studio 97, click Build, and then click Rebuild All.
   - If you are using Microsoft Developer Studio 4.x, click Tools and then click BoundsChecker Rebuild All.

   In subsequent builds, use the Build option to instrument and compile only files changed or added since the last build.

In the Build tab within the Output window, BoundsChecker lists entries for all the errors and leaks it detects. To display the source code that contains a particular error or leak, double-click the entry.

Building a Program for FinalCheck From the Command Line

From a DOS session, use NMAKE and the following BoundsChecker components to build your program for FinalCheck:

BoundsChecker Component	Description
NMCL.EXE	BoundsChecker compiler driver. Use this driver in place of the CL.EXE.
NMLINK.EXE	BoundsChecker linker driver.
BCINTERF.LIB	BoundsChecker library file. All instrumented programs require this file.

The instrumentation process automatically substitutes the BoundsChecker NMCL compile driver for the standard Visual C++ compiler driver. The following sections describe how to use the Microsoft and Win32 SDK makefiles to build your program for FinalCheck.

Building a Program With a Microsoft Makefile

To use a standard Microsoft makefile to build your program for FinalCheck, specify CPP=NMCL.EXE on the NMAKE command line.

See, "Using NMCL Command-line Options," for a list of options you can use to refine the build process.

The following example assumes that your path variable includes NMCL and NMLINK:

```
NMAKE /f TEST.MAK CPP=NMCL.EXE LINK32=NMLINK.EXE
```

Note: If your path variable does not include NMCL and NMLINK, specify the full directory path to these programs (located in the BoundsChecker installation directory).

Building a Program With a Win32 SDK Makefile

If your makefile is based on the original Win32 SDK, the substitution line is slightly different, and you need to add it after the line that includes the NTWIN32.MAK file:

```
!include <ntwin32.mak>
CC=NMCL.EXE
LINK=NMLINK.EXE
```

See the following section for a list of options you can use to refine the build process.

Using NMCL Command-line Options

To apply NMCL command-line options to an entire project, specify NMCL and the appropriate option in quotes.

The following example shows the correct format for specifying an NMCL option:

```
NMAKE /f TEST.MAK CPP="NMCL /NMopt:TEST.INI"
```

You can also use the NMCL environment variable to specify NMCL command-line options. This is useful for specifying global options.

The following example prevents a particular source file from being instrumented:

```
SET NMCL=/NMignore:STDAFX.CPP
```

The following table lists the available NMCL options.

NMCL Option	Description
/NMproj:*project-name*	Specifies a BoundsChecker project. This option is typically used from within the Visual C IDE.
/NMignore:*source-file[:function]*	Specifies a source file or function that should not be instrumented.
/NMonly:*source-file[:function]*	Specifies a source file or function that should be instrumented.
/NMopt:*option-file* or /NM@*option-file*	Specifies an option file.
/NMlog:*log-file*	Specifies a log file for NMCL messages (default:stdout).
/NMpass	Specifies pass-through mode, which instructs NMCL to call CL without intervention. No instrumentation occurs.
/NMstoponerror	Stops NMCL if an error occurs during instrumentation.
/NMclpath:*cl-path*	Specifies the directory location of CL.EXE. Use this option if MSDEV is not installed or to bypass the MSDEV installed location.
/NMbcpath:*bc-path*	Specifies the location of the install directory for BoundsChecker. If necessary, use this option to instruct NMCL where to find BCINTERF.LIB.
/NMnogm	Ignores the CL /Gm (minimal build) option if it appears on the command line. Use this option to avoid a known conflict between the NMAKE /A and CL /Gm options.
/NMhelp	Displays help text.

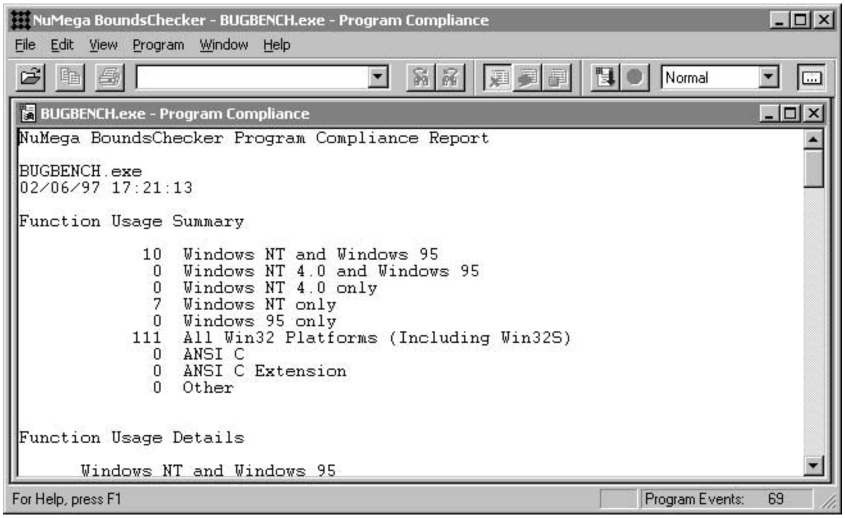
Knowledge is what we get when an observer, preferably a scientifically trained observer, provides us with a copy of reality that we can all recognize.

◊ Christopher Lasch

5 Checking Compliance

Microsoft provides a collection of 32-bit application programming interfaces (APIs) called Win32. Win32 is implemented under Windows NT and Windows 95. A portion of Win32 is also implemented on Windows 3.1 as Win32s.

Although many of the APIs within Win32 support both Windows NT and Windows 95, some are platform specific. You can unknowingly use a call or set of calls that are available on one platform, but not another. To assure that your program is compliant across both Windows platforms and Win32s, BoundsChecker provides compliance reports that categorize your program's APIs. Use these reports to determine if your program's APIs are available on all Win32 platforms or just a subset of Win32 platforms. Additionally, BoundsChecker categorizes your program's use of C Run-Time Library calls into ANSI and non-ANSI. The following example illustrates a compliance report.


NuMega BoundsChecker - BUGBENCH.exe - Program Compliance
File Edit View Program Window Help
Normal
BUGBENCH.exe - Program Compliance
For Help, press F1
Program Events: 69

```
NuMega BoundsChecker Program Compliance Report

BUGBENCH.exe
02/06/97 17:21:13

Function Usage Summary

             10  Windows NT and Windows 95
              0  Windows NT 4.0 and Windows 95
              0  Windows NT 4.0 only
              7  Windows NT only
              0  Windows 95 only
            111  All Win32 Platforms (Including Win32S)
              0  ANSI C
              0  ANSI C Extension
              0  Other


Function Usage Details

      Windows NT and Windows 95
```

Checking API Compliance

The best strategy for ensuring API compliance is to keep compliance in mind as you design a program and to use BoundsChecker compliance checking early in the development process. Otherwise, finding and fixing API compliance-related problems can be a long and tedious task, especially if you postpone the job until after a program is completed.

BoundsChecker provides two ways for checking compliance:

- Program Compliance

  Lists all functions to which your program's .EXE file refers.

- Event Compliance

  Lists only those functions that were called when you checked the program.

Checking Program Compliance

Program Compliance categorizes and lists all functions to which your .EXE file refers. BoundsChecker does not list API calls made by DLLs to which the EXE points. To check DLLs, use Event Compliance.

The following sections describe how to produce a Program Compliance report from Microsoft Developer Studio and the BoundsChecker application.

Checking Program Compliance From Microsoft Developer Studio

The method for checking program compliance varies between Microsoft Developer Studio 97 and 4.x.

To create a Program Compliance report from within Microsoft Developer Studio 97:

1. Open the program for which you want to produce the report.
2. On the BoundsChecker menu, click Check Program Compliance.

To create a Program Compliance report from within Microsoft Developer Studio 4.x:

1. Open the program for which you want to produce the report.
2. On the Tools menu, click BoundsChecker to start the BoundsChecker application. Use the BoundsChecker application to generate compliance reports.
3. On the Program menu within the BoundsChecker application, click Check Program Compliance to generate the compliance report.

Checking Program Compliance From BoundsChecker

To create a Program Compliance report from within the BoundsChecker application:

1. Open the program for which you want to produce the report.
2. On the Program menu, click Check Program Compliance.

Checking Event Compliance

Event Compliance uses the results of your error-detection session to categorize and list only those Win32 and C Run-Time Library functions that were actually called when you ran the program. Event Compliance includes calls made by DLLs to which your program points.

To produce an Event Compliance report, you need to set the Event Reporting program settings before you run your program. The following steps explain how to set the Event Reporting program settings and produce the compliance report from within Microsoft Developer Studio and the BoundsChecker application.

Checking Event Compliance From Microsoft Developer Studio

The method for checking event compliance varies between Microsoft Developer Studio 97 and 4.x.

To create an Event Compliance report from within Microsoft Developer Studio 97:

1. Open the program for which you want to produce the report.
2. On the BoundsChecker menu, click Settings.
3. In the BoundsChecker Settings window, click the Event Reporting tab.
4. Select Collect and Report Detailed Event Data and click OK.
5. Check the program, using all the functions you want included in the compliance report.
6. On the BoundsChecker menu, click Check Event Compliance to produce the report.

To create an Event Compliance report from within Microsoft Developer Studio 4.x:

1. Open the program for which you want to produce the report.
2. On the Tools menu, click BoundsChecker Settings.
3. In the BoundsChecker Settings window, click the Event Reporting tab.
4. Select Collect and Report Detailed Event Data and click OK.
5. Check the program, using all the functions you want included in the compliance report.
6. On the Tools menu, click BoundsChecker to start the BoundsChecker application, which generates compliance reports.
7. On the Program menu within the BoundsChecker application, click Check Event Compliance to generate the compliance report.

Checking Event Compliance From BoundsChecker

To produce an Event Compliance report:

1. On the Program menu, click Settings.
2. In the BoundsChecker Settings window, click the Event Reporting tab.
3. Select Collect and Report Detailed Event Data and click OK.
4. Check the program, using all the functions you want included in the compliance report.
5. On the Program menu, click Check Event Compliance to produce the report.

An ounce of prevention is worth a pound of cure.

◊ Proverb

6 Validating Your Own APIs

Application Programming Interfaces (APIs) are the most popular model on Windows for defining how DLLs work together. APIs are also one of the most error-prone areas for programmers, often resulting in bugs that are difficult to catch. Additionally, APIs can cause strange behavior that is difficult to reproduce.

BoundsChecker excels at finding API-related errors for a pre-defined group of function sets. The following table lists these sets.

Supported API Functions	Supported OLE Functions
CRTL	Direct X
MAPI	Active X
ODBC	
WIN32	
WINSOCK	

For each function within the API, BoundsChecker automatically validates:

- Every parameter for the function, including parameters specific to Windows, such as hWnds and hMenu. See *Default Parameter and Return Types* on page 52.
- The parameter’s return value. This validation finds errors in parameter types and ranges.

The following table lists the types of API and OLE errors BoundsChecker detects.

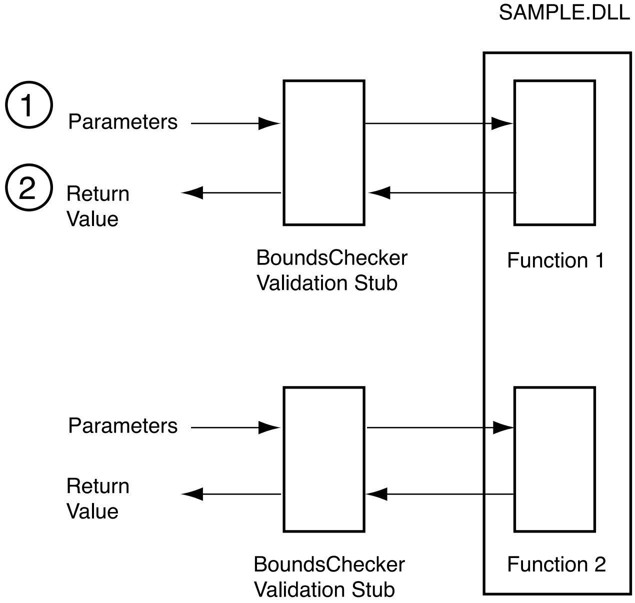
API and OLE Errors BoundsChecker Detects
API failure: Windows function failed
API failure: Windows function not implemented
Interface method failure
Invalid argument
Invalid argument: At least one format specifier is illegal
Invalid argument: Bad destination pointer
Invalid argument: Bad handle
Invalid argument: Bad source pointer
Invalid argument: Conflicting combination of flags
Invalid argument: Format string is not followed by valid arguments
Invalid argument: Invalid pointer to format string
Invalid argument: Not enough arguments for this format string
Invalid argument: Out of range
Invalid argument: Structure size field is not initialized
Invalid argument: Too many arguments for this format string
Invalid argument: Undefined or illegal flags
Invalid interface method argument
Invalid interface method argument: Conflicting combination of flags
Invalid interface method argument: Out of range
Invalid interface method argument: Structure size field is not initialized
Invalid interface method argument: Undefined or illegal flags
Questionable use of thread

You can add your own API function sets to the function sets BoundsChecker validates. When you extend BoundsChecker to test your APIs, BoundsChecker automatically validates parameters, traces and validates return values, and lists the trace data in the Program Results window, where you can analyze it.

How BoundsChecker Validates APIs

To validate parameters and return values, BoundsChecker uses a validation routine to patch a piece of code for each function into the memory image of the application under test. This piece of code is called a validation stub: it is responsible for intercepting each API call, validating it, logging it, and updating the Program Results window.

The following figure illustrates how a validation stub intercepts and validates parameters and return values for a function.



1 The validation stub intercepts parameters as they are passed to the function. The validation stub logs and validates each parameter against a set of known values or conditions.

2 On return from the API call, the validation stub logs and validates the return value, if one exists.

Creating an API Validation Module

To validate your own APIs, create a validation module for each .DLL (or .DEF) you want to test. BoundsChecker provides an API wizard to simplify this process. When you create a validation module, the API wizard analyzes your .DLL and automatically generates the source code for the validation module, including the validation stubs.

The following general steps explain how to create and use a validation module for your DLL. See the On-line Help for more information about options you can use to refine this process.

1 Click Start, point to NuMega BoundsChecker, and then click Generate API Validation Module to start the API wizard (BCAPIWIZ.exe).

   The wizard automatically analyzes your .DLL to determine the parameters it uses and the calls it exports. Then, the wizard uses this information to generate a source (.CPP) file for the validation module. Additionally, the wizard generates a .MAK file and a header (.H) file.

2 Review the .CPP file to verify that the code BoundsChecker generated for the .DLL's parameters and return calls is complete.

   C++ functions use decorated names that contain parameter information, so BoundsChecker can automatically provide the parameter validation code for any C++ function exported from a DLL. However, you need to create your own C or C++ validation code for C functions exported from a DLL or for parameter types that BoundsChecker does not recognize. BoundsChecker indicates where you need to add code by placing "TODO:" comments in the generated source file.

3 From the command prompt, run NMAKE or MAKE on the .MAK file, as follows:

   ```
   [N]MAKE -f makefile-name
   ```

   Example: `NMAKE -f FOO.MAK`

   NMAKE uses the compiler and linker to build an .API validation file. Then, NMAKE places the file into the BoundsChecker APICheck directory.

4 Check your application as you normally would.

   BoundsChecker automatically uses the validation module to validate your .DLL.

Default Parameter and Return Types

By default, BoundsChecker generates validation and logging code for the following parameter types. To add your own parameter types, see the On-line Help.

Parameter Type	Action	Parameter Type	Action
ATOM	Validate and log	HSZ	Log as DWORD value
BOOL	Log as DWORD value	HTHREAD	Validate and log
bool	Log as DWORD value	HWND	Validate and log
BYTE	Log as BYTE value	IDHOOK	Validate and log
char *	Validate and log	int	Log as DWORD value
char **	Validate	KERNELHANDLE	Validate and log
CHAR	Log as BYTE value	LCID	Log as DWORD value

Parameter Type	Action	Parameter Type	Action
char	Log as BYTE value	LCTYPE	Log as DWORD value
COLORREF	Log as DWORD value	long	Log as DWORD value
COORD	Log as DWORD value	LONG	Log as DWORD value
double	Log	LPARAM	Log as DWORD value
DWORD	Log	LPCODE	Validate and log
FILE *	Validate and log	LPCSTR	Validate and log
float	Log	LPCWSTR	Validate and log
GLOBALATOM	Validate and log	LPSTR	Validate and log
HACCEL	Validate and log	LPWSTR	Validate and log
HANDLE	Validate and log	PACL	Validate and log
HCONV	Log as DWORD value	PSID	Validate and log
HCONVLIST	Log as DWORD value	REGSAM	Log as DWORD value
HCURSOR	Validate and log	SECURITY_INFORMATION	Log as DWORD value
HDBC	Validate and log	SERVICE_STATUS_HANDLE	Log as DWORD value
HDDEDATA	Log as DWORD value	short	Log as DWORD value
HDWP	Validate and log	signed char	Log as BYTE value
HENV	Validate and log	signed int	Log as DWORD value
HFILE	Validate and log	signed long	Log as DWORD value
HGLOBAL	Validate and log	size_t	Log as DWORD value
HHEAP	Validate and log	this	Log
HHOOK	Validate and log	UCHAR	Log as BYTE value
HICON	Validate and log	UINT	Log as DWORD value
HINST	Validate and log	ULONG	Log as DWORD value
HKEY	Log as DWORD value	unsigned char	Log as BYTE value
HKL	Validate and log	unsigned int	Log as DWORD value
HLOCAL	Validate and log	unsigned long	Log as DWORD value
HMENU	Validate and log	unsigned short	Log as DWORD value
HPROCESS	Validate and log	WORD	Log as DWORD value
HRSRC	Validate and log	WPARAM	Log as DWORD value
HSTMT	Validate and log		

The validation module utility can validate the following return types.

- ATOM
- bool
- BOOL
- CHAR
- char
- char *
- COLORREF
- double
- DWORD
- FAPPROC
- FILE *
- HACCEL
- HANDLE
- HBITMAP
- HBRUSH
- HCONV
- HCONVLIST
- HCURSOR
- HDC
- HDDEDATA
- HDWP
- HENHMETAFILE
- HFILE
- HFONT
- HGDIOBJ
- HGLOBAL
- HHOOK
- HICON
- HINSTANCE
- HKL
- HLOCAL
- HMENU
- HMETAFILE
- HMODULE
- HPALETTE
- HPEN
- HRESULT
- HRGN
- HRSRC
- HSZ
- HWINSTA
- HRESULT
- HRGN
- HRSRC
- HSZ
- HWND
- INT
- int
- int *
- LANGID
- LCID
- LONG
- long
- long *
- LPSTR
- LPVOID
- LPWSTR
- LRESULT
- PDWORD
- PSID_IDENTIFIER_AUTHORITY
- PUCHAR
- RETCODE
- SC_HANDLE
- SC_LOCK
- SERVICE_STATUS_HANDLE
- SHORT
- short
- short int *
- signed int
- size_t
- struct lconv *
- struct tm *
- time_t
- tm *
- UINT
- unsigned char
- unsigned char *
- unsigned int
- unsigned int *
- unsigned long
- unsigned long *
- unsigned short int
- unsigned short int *
- void
- void *
- WORD

Index