All Topics, MFC / C++ >> Free Tools >> Debugging
http://www.codeproject.com/tools/blackbox.asp

C++ (VC6)
Windows (WinXP, Win2K, NT4)
Win32, VS (VS6)
Dev
Posted 2 Aug 2002
Updated 26 May 2003
73,336 views

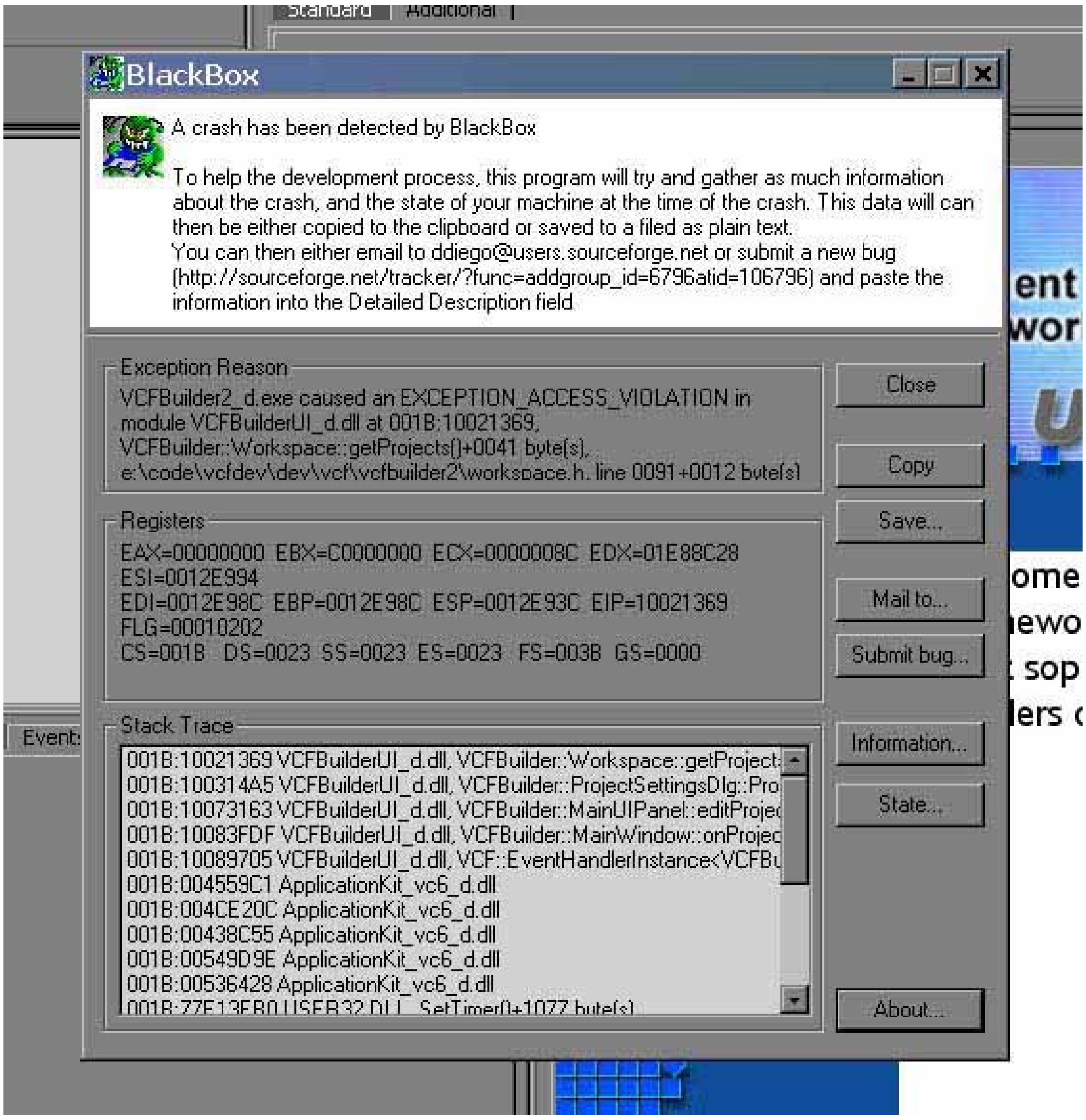
Trapping Bugs with BlackBox

By Jim Crafton

A brief article explaining how to use BlackBox in your programs to trap errors

25 votes for this article.
Popularity: 6.31. Rating: **4.51** out of 5.

- Download source files - 178 Kb



A crash has been detected by BlackBox

To help the development process, this program will try and gather as much information about the crash, and the state of your machine at the time of the crash. This data will can then be either copied to the clipboard or saved to a filed as plain text.
You can then either email to ddiego@users.sourceforge.net or submit a new bug (http://sourceforge.net/tracker/?func=addgroup_id=6796atid=106796) and paste the information into the Detailed Description field

Exception Reason
```
VCFBuilder2_d.exe caused an EXCEPTION_ACCESS_VIOLATION in
module VCFBuilderUI_d.dll at 001B:10021369,
VCFBuilder::Workspace::getProjects()+0041 byte(s),
e:\code\vcfdev\dev\vcf\vcfbuilder2\workspace.h, line 0091+0012 byte(s)
```

Registers
```
EAX=00000000  EBX=C0000000  ECX=0000008C  EDX=01E88C28
ESI=0012E994
EDI=0012E98C  EBP=0012E98C  ESP=0012E93C  EIP=10021369
FLG=00010202
CS=001B   DS=0023  SS=0023  ES=0023   FS=003B   GS=0000
```

Stack Trace
```
001B:10021369 VCFBuilderUI_d.dll, VCFBuilder::Workspace::getProject
001B:100314A5 VCFBuilderUI_d.dll, VCFBuilder::ProjectSettingsDlg::Pro
001B:10073163 VCFBuilderUI_d.dll, VCFBuilder::MainUIPanel::editProjec
001B:10083FDF VCFBuilderUI_d.dll, VCFBuilder::MainWindow::onProjec
001B:10089705 VCFBuilderUI_d.dll, VCF::EventHandlerInstance<VCFBu
001B:004559C1 ApplicationKit_vc6_d.dll
001B:004CE20C ApplicationKit_vc6_d.dll
001B:00438C55 ApplicationKit_vc6_d.dll
001B:00549D9E ApplicationKit_vc6_d.dll
001B:00536428 ApplicationKit_vc6_d.dll
001B:77E13EB0 USER32.DLL, SetTimer()+1077 byte(s)
```

Introduction

Have you ever had an application crash on a customer's machine and have only, "it just crashes and displays a funny message box", to go on in attempting to track down what the error was? Ever wish you could put some kind of "watchdog" that would monitor your application on the customers computer and then if it crashed, automatically scoop up where the crash happened, and allow the customer or mail this information to you? If so then BlackBox may be just what you're looking for!

BlackBox (which I have since discovered is the name of a window manager for linux, so if anyone has any ideas about a better name, please feel free to let me know :)) is a dll that you simply load up and then forget about. It will set a crash handler as soon as the library is loaded (in `DLLMain()`), and then just waits patiently for your application to crash. If and/or when your application crashes, instead of just displaying an error message like "Error writing to memory address 0XDEADBEEF", a nice little dialog comes up that displays the information in a fashion similar to Netscape's TalkBack, or the new error reporting dialog that pops up with the newer version of Microsoft's Internet Explorer. This dialog allows the user to then copy or save the crash data, and then email or be taken to an appropriate website where they can submit the information. This information can then be used by a programmer to have a much better idea of exactly where the problem occurred, as well as information about the machine that it crashed on.

It is extremely lighweight as a DLL at only 32 Kb in size, so there is no worry about bloat if you decide to use this!

Customization

You can customize the text that is displayed in the main label, by opening BlackBox.cpp and editing the following code:

```
const char* g_errorLable =
                "This is my version of BlackBox - An error has occured!
                 You are screwed";
```

By changing the `g_errorLable` variable, this will change what is displayed in the label in the top area of the dialog.

In addition, you can customize the email address to use when the "Mail to..." button is clicked, and the URL that is used when the "Submit bug..." button is clicked. Again open BlackBox.cpp and editing the following code:

```
const char* g_mailToAddress=
                //change this to something useful
                "mailto:me@wherever.com"
const char* g_submitBugURL=
                //change this to something useful
                "http://www.mybugtrackingpage.html"
```

Usage

BlackBox displays a variety of information:

- The Exception Reason
- The Registers
- The Stack Trace, a complete stack trace from where the error occured
- The machine information, such as CPU, OS version, physical memory, etc. Clicking on the "Information..." button will display this in a dialog.
- The machine state - a list of ALL the processes running at the time of the crash, and any DLLs loaded by the processes. Clicking on the "State..." button will display this in a dialog.

As a user, you can copy the contents to the clipboard by clicking on the "Copy" button. Clicking the "Save..." button will prompt you to save the contents to a file, with the default name being of the form: "errorlog-<GMT time, dd-mm-yyyy>(<time, hh:mm:ss>).log".

Click the "Mail To..." button and an attempt will be made to execute the mailto: command on your system. Assuming a properly installed email program, it will be brought up with the appropriate email address. See above for configuring these two features.

The basics of the code is "stolen" from John Robbins excellent book Debugging Applications that show how to do all sorts of gory low level things, among them performing a stack trace. To get the stack trace to provide something useful, however, you may have to make some modifications to your build settings for release builds. The stack tracing relies on certain system calls, as well as information in the executable itself. If the executable is a release build, then there will be almost no debug information, and as a result, your stack trace will show almost nothing, which for me, at least, is one of the critical reasons why I wrote this tool. To allow the symbol engine to successfully display information, you need to enable the generation of debug information (/debug) in the linker settings (you do NOT need to do this in your code generation or C++ settings), and you need to tell the linker to generate a mapfile (/map ["filename"]). Then when you distribute your application make sure to also put the `BlackBox.dll` and the map file(typically the map file is the same name as the executable, but with a "`.map`" extension) in the same directory as your exe. Obviously this does mean you are "exposing" a bit more information to people (i.e. your customers), but the tradeoff is readable stack trace information that can be invaluable in locating where a crash happened. BlackBox will still work with an executable that does not supply a map file or linker debug information, it will just not display as much information.

To actually use this in your application, you only need to do the following:

```
HINSTANCE hLib = LoadLibrary( "BlackBox.dll" );
```

That's it! To be a good Win32 citizen, you should unload the library when your app exits:

```
if ( NULL != hLib ) {
    FreeLibrary( hLib );
}
```

Everything else is handled internally in the BlackBox library.

One final note: To build this code you will need the latest Platform SDK (at least from the last year or so). In addition, when you put this out in the wild, so to speak, you will need to distribute the `psapi.dll` and the `dbghelp.dll` dll's for this to all work, they are part of what allows the symbol engine to work.

To Do:

In the near future I would like to add support for encrypting the data, right now it is just plain text, and this may not be appropriate in certain cases. I would also like to write a server program that would run on a machine, and to which the BlackBox client program could simply send the encrypted data via sockets.

Further Customization:

Should you wish to do more radical customizations, the core UI code is all contained in the BlackBoxUI.cpp. This is what allocates all the memory, which is done using `GlobalXXX` functions, as we may not be able to use the local heap. The starting point for all of this is the `ShowBlackBoxUI()` function, which sets up all the information and then displays the main dialog.

Credits:

John Robbins, Debugging Applications, Microsoft Press; ISBN: 0735608865, 2000

See this for more information about John Robbins at Wintellect.

The display code for the various processes was based on work that Emmanuel Kartmann did in his article Display Loaded Modules

Updates:

The BlackBox dll is part of the toolset created for the Visual Component Framework. For further updates please see the VCF project page and either check the CVS status or look at the files section for the newest BlackBox file release.

About Jim Crafton

Currently working on the Visual Component Framework, a really cool C++ framework. Currently the VCF has millions upon millions upon billions of Users. If I make anymore money from it I'll have to buy my own country.

Click here to view Jim Crafton's online profile.

Discussions and Feedback

61 comments have been posted for this article. Visit **http://www.codeproject.com/tools/blackbox.asp** to post and view comments on this article.

All Topics, MFC / C++ >> Free Tools >> Debugging
Updated: 26 May 2003