This file documents

ExcHook

- an exception and stack trace logging unit for Delphi[1] applications.

& ParseRIP[2]

- a Delphi IDE expert for inspecting ExcHook log files.

Version 1.03

Overview

Exception handling is a very cool way of getting rid of the endless explicit tests for all the different error conditions that might occur during program execution. Routines that detect an error condition can simply raise an appropriate exception while the rest of the code for the most part can assume that nothing unusual happens. If the default approach doesn't quite cut it, the programmer can check for various specific error conditions, using except statements, and perform special action for those, while other, more generic exception types are passed on to the default handler.

This works quite well for the cases where the meaning of an exception is obvious - even out of context. Very often, however, this is not the case. There are a number of exception types that can occur for any number of reasons in a application.

For exceptions like "list index out of bounds", "file not found", "cannot focus a hidden or disabled window" etc, that can occur for a number of reasons at the higher level, we have the *break on exception* option in the IDE, which - along with the Call Stack window - will usually reveal why the error occured.

Unfortunately, a lot of exceptions do not occur in the development environment but while the application is being tested or even at client sites. The ExcHook unit presented here essentially does what the IDE option *break on exception* does for stand-alone applications. Well, almost: Rather than breaking execution when an exception is raised, ExcHook records a stack trace, and if the exception reaches the default handler (that is, it is not handled by a specific except statement), the exception text and the stack trace is logged to a text file (RIP).

This file can be passed back to the developer, who can locate the program locations from the stack dump using the ParseRIP expert or by utilizing the Search/Find error functionality of the Delphi IDE.

[1] Delphi is a trademark of Borland International
[2] RIP = rest in piece (I chose to use this common acronym though Delphi applications rarely die)

Installation

The "shareable" version of   this product consists of the following files:

ExcHook.dcu - the compiled exception and stack trace logging unit
ExcHook.int - the interface section of ExcHook.pas (not included)
ParseRIP.dll - the .RIP inspection IDE expert
cExcHook.pas - a "component" file for installing ExcHook in complib
ExHook.hlp - this file

In addition, the registered version consists of:

ExcHook.pas - the full source code for the exception and stack trace logging unit
PRIPUser.lic - the user license for the ParseRIP expert

All files can be installed in any directory of your choosing. Only, PRIPUser.lic, if you have it, should be in the same directory as ParseRIP.dll.

Installation of the ParseRIP expert

The ParseRIP expert - like all other Delphi IDE experts - is installed via the delphi.ini file:

In the [Experts] section, add a line like so:

```
ParseRIP=c:\delphi\ExHook\ParseRIP.dll
```

The expert should appear on the Help menu (I wonder why they didn't choose the Tools menu for this) the next time you fire up the Delphi IDE.

Using ExcHook

You include ExcHook in an application or DLL simply by adding it to the uses list of the project. The easiest way of doing this is to add it via the project manager, but editing the project source files works just as well.

I recommend that you mention ExcHook as early as possible in the uses list to ensure that it is active if an exception occurs during initialization of the units that follow. Of course, those units used by ExcHook itself (that's SysUtils, WinTypes, WinProcs, Classes and Forms, btw) will be initialized first.

In case you're wondering why I didn't turn ExcHook into a component, it's because exceptions might happen before such a component would have a chance to be created or after it would have been destroyed.

How it works

During initialization, ExcHook installs itself as kind of a built-in debugger to your application. ExcHook receives notification when an exception is raised just like the IDE debugger or the stand-alone Turbo Debugger would. This means that ExcHook can't do its thing while any one of those debuggers are active. In other words, while your program is running under the Delphi IDE or TD, ExcHook does nothing! This assuming we're talking applications. ExcHook works fine in DLLs that are called from applications running in the IDE. The reason is that the IDE does not support debugging of DLLs, so we're free to use the exception hook for our purposes.

The first time ExcHook gets notified of a raise, it "subclasses" the currently installed default exception handler. That is, it borrows the Application.OnException pointer and installs its own. If an exception reaches the default handler (which is now ExcHook's handler), the exception is logged to a .RIP file and then passed on to the previously installed handler. If you didn't install a custom exception handler in Application.OnException, this will be the default handler which displays the exception text as a message box.

If you do install a custom exception handler, make sure the assignment to Application.OnException takes place before the first exception would normally be raised.

If you want to prevent certain common types of exceptions - user input errors, for instance - from being logged, you can install an exception filter in ExcHook's ExceptionFilter variable. It's of the following type

```
TExceptionFilter = procedure(E: Exception; var PutInLog : Boolean);
```

and it defaults to nil, meaning that all unhandled exceptions are logged by default.

PutInLog defaults to true. You might install an exception log filter in the project file like this:

```
procedure EFilter(E : Exception; var PutInLog : Boolean); far;
begin
  Result := not (E is EDBEditError);
end;

begin {DPR}
  RipInfo := 'Version 1.00';
  ExceptionFilter := EFilter;
..
```

The .RIP file is a plain text file, and it contains the text of the exception that occured, plus a stack dump. The name of the .RIP file is initialized to the name of the application (ChangeFileExt(ParamStr(0),'.RIP')) during initialization and stored in an interfaced variable, RipFileName, so that it can be changed. See ExcHook.int (or ExcHook.pas) for details.

ExcHook appends to an existing RIP file if one exists and creates a new one if it doesn't. ExcHook never deletes files.

A typical RIP file might look like this:

```
Version 1.00

Fault : General protection fault in module EXCPTEST.EXE at 0001:0224
Date/time:11-02-96 20.40.15

Stack dump
----------
0001:0283
0006:2747
0004:0FF5
0004:113D
0006:25BF
0006:4585
0004:0F55
0006:24AB
0006:4662
0007:4A04
0006:25BF
0006:4585
0007:305E
0006:4223
0007:13C8
.
.
```

(next exception)

The "Version 1.00" line in this sample comes from the global RipInfo variable in ExcHook which is user defined information, written once when the RIP file is created.

The accompanying ParseRIP expert will parse files like this and - using information from the MAP file for the application - produce a DOC file which has the source locations for each entry in the stack dump.

Using ParseRIP

ParseRIP is a utility - in the form of a Delphi expert - for converting RIP file stack dumps to a more useful form.

Use of ParseRIP couldn't be simpler: You invoke ParseRIP from the help menu. Then, you "Select RIP file", and if ParseRIP finds a corresponding MAP file, it moves along, producing a DOC file with the same name. If not, you are prompted for the MAP file's name and location. The resulting DOC file can be inspected in the built-in browser, but is also written to disk. A typical DOC file generated with ParseRIP might look like this:

```
Version 1.00

Fault : General protection fault in module EXCPTEST.EXE at 0001:0224
Date/time:11-02-96 20.40.15

Stack dump
----------
0001:0283 (UEXCPTST.PAS line 62)
0006:2747 (CONTROLS.PAS line 1733)
0004:0FF5 (STDCTRLS.PAS line 2302)
0004:113D (STDCTRLS.PAS line 2344)
0006:25BF (CONTROLS.PAS line 1689)
0006:4585 (CONTROLS.PAS line 2567)
0004:0F55 (STDCTRLS.PAS line 2282)
0006:24AB (CONTROLS.PAS line 1639)
0006:4662 (CONTROLS.PAS line 2599)
0007:4A04 (FORMS.PAS line 2566)
0006:25BF (CONTROLS.PAS line 1689)
0006:4585 (CONTROLS.PAS line 2567)
0007:305E (FORMS.PAS line 1859)
0006:4223 (CONTROLS.PAS line 2485)
0007:13C8 (FORMS.PAS line 982)
```

ParseRIP will attempt to synthesize a meaningful symbolic name from any available public symbol, in case line number information isn't available for a particular piece of code. In general, you should include debugging information for all code, if you can.

If you place the cursor on a line with a file name, ParseRIP will let you open that file in the IDE by pressing "Open file at cursor". Unfortunately, the IDE only appears to supports this for files that are in the current directory.

"Delete files" will delete the RIP and DOC you are currently browsing. This can be convenient during testing. In particular for complib.rip.

Note! Make sure that the MAP file you're using is in sync with the RIP file. ParseRIP has no way of knowing whether the MAP file it finds is from a previous or newer version of your application or library.

Using ExcHook in complib

I've already mentioned elsewhere that ExcHook works in DLLs as well normal applications. A special case is complib.dcl, the component library.

Now, since ExcHook itself isn't a component, why, then, would it be interesting to install it in complib? The answer: As an aid for debugging the design-time interface of other components:

Developing components, not to mention component and property editors, can be a frustrating experience: If any kind of exception occurs while you are working with your new component in the IDE, you are left with two alternatives: Either you can stare at the code until you figure out what the problem is, or you can load Delphi.exe in the stand-alone debugger and set breakpoints in complib as it loads. This is clumsy at best.

However, with ExcHook installed in complib, you get a complib.RIP file in your delphi\bin directory when such exceptions occur at design time. All you need to do is load complib.RIP with ParseRIP to get a stack dump telling you where the exception occured and how you got there. Of course, this assumes that you've elected to have a MAP file created for you complib.dcl.

Unfortunately, not all exceptions that occur at design time reach complib's default exception handler, but those that do will end up in the RIP file.

Installing ExcHook in complib.dcl

In the ExHook archive, you'll find a very simple file, cExcHook.pas. In fact, all it contains is an empty Register procedure, and then it mentions ExcHook in its uses list. The Register procedure is there so that the Delphi IDE will accept it as a component file.

Install it via Options/Install components. Of course, there's really no component there, but Delphi will link it as part of complib anyway. When complib has re-compiled, you're all set.

Pricing and availability

This software is shareware. The registration price for ExcHook and ParseRIP is US$49. The package is only available as a site license. With the registered version, you acquire the right to use ParseRIP and to use and distribute the ExcHook unit as part of compiled applications, but not in source or dcu form. The registered version includes a run-time license for ParseRIP, and full (completely undocumented) source code for the ExcHook unit, but not for ParseRIP. Why? I didn't feel like cleaning it up: It pulls a lot of functionality from TurboPower Software's Orpheus package, that I didn't feel like re-inventing. It's hardly rocket science anyway, so you can just write your own ParseRIP code if you need it.

If you're a CompuServe member, you can register ExHook via CompuServe's shareware registration service (GO SWREG, registration ID is 9953).

For credit card orders,   you can order with MC, Visa, Amex, or Discover via my web site http://ourworld.compuserve.com/homepages/PBLarsen. Just follow the links. Or you can order from Public (software) Library directly by calling 800-2424-PsL or 713-524-6394 or by FAX to 713-524-6398 or by CIS Email to 71355,470. You can also mail credit card orders to PsL at P.O.Box 35705, Houston, TX 77235-5705.

THE ABOVE NUMBERS ARE FOR CREDIT CARD ORDERS ONLY.
THE AUTHOR OF THIS PROGRAM CANNOT BE REACHED AT THESE NUMBERS.

Any questions about the status of the shipment of the order, refunds, registration options, product details, technical support, volume discounts,
dealer pricing, site licenses, non-credit card orders, etc, must be directed to me on the address below rather than PsL.

To insure that you get the latest version, PsL will notify me the day of your order and I will ship the license directly to you.

PsL's item code for ExHook is 14.545

You can also register by sending a check to me via ordinary mail. See my address below.

The user license for ParseRIP and the source code for the ExcHook unit amount to approximately 5K bytes compressed.   I will send this to you as a compressed archive via E-Mail as soon as I get confirmation on payment. For an additional handling fee of US$10, I can also send you the complete product on a disk.

How to get the latest version

The latest version of ExHook is always available on http://ourworld.compuserve.com/homepages/PBLarsen on the Internet, and in lib 15 of the Delphi forum on CompuServe.

How to reach me

I usually hang out in the Delphi forum on CompuServe (GO DELPHI), so if you're having trouble using this utility, you might post a question there.

You can write me directly on the following address:

CompuServe 75470,1320

From the Internet, you reach CompuServe mail like so:

75470.1320@compuserve.com

You can also write me by snail mail as

Per Larsen
Sct. Anna Gade 53
DK-8000 Aarhus C
Denmark

Nonsense license agreement.

1. User Obligations.

1.1 User assumes full responsibility that this software meets the specifications, capacity, capabilities, and other requirements of said user, and agrees not to bother the author if the software does not perform as expected, or performs other than expected, or does not perform at all.

1.2 User assumes full responsibility for any deaths or injuries that may result from the normal or abnormal operation of this software. In the event of casualties exceeding 1000 persons or property damage in excess of US$10 million, user agrees that he or she has stolen the software and I didn't even know he or she had it.

1.3 User agrees not to say bad things about the software or the author to anyone claiming to be from "60 minutes".

2. Limited Warranty.

2.1 For a period of 90 minutes, commencing from the time you first thought about getting this software, I warrant that this software may or may not be free of any manufacturing defects. It will be replaced during the warranty period upon payment of an amount equal to the original purchase price plus US$100 for handling. This warranty is void if the software has been examined or run by the user, or if the documentation has been read.

2.2 This software is distributed on an AS WAS basis. The author makes no warranty that this is, in fact, what I say it is in my propaganda, or that it will perform any useful function. I have no obligation whatsoever other than to provide you with this fine disclaimer.

2.3 Some countries do not allow limitations as to how long an implied warranty lasts, so I refuse to imply anything.

2.4 There is an extremely small but non-zero chance that, through a process known as "tunnelling", this software may spontaneously disappear from its present location and reappear at any random place in the universe, including your neighbours computer system. The author will not be responsible for any damages or inconvenience that may result.

3. Limitation of liability.

3.1 I have no liability or responsibility to the user, the user's agents, my creditors, your creditors, or anyone else.

Revision history

Version	Comments
1.00	Initial release
1.01	Fixed a bug in ParseRIP in the algorithm responsible for generating symbolic information from publics. Sometimes, garbage was produced.
	Borrowed some optimized MAP parsing code from MemMonD.
	Fixed a bug with loading user licenses.
1.02	Changed the stack trace logic: The previous algorithm was a bit too simplistic, causing it to miss a few near frames here and there.
	Fixed a bug in the MAP parser in ParseRIP relating to unit names longer than 18 characters.
1.03	ParseRIP is now a non-modal expert.
	ParseRIP adds printing functionality.
	ParseRIP now has a help button for loading this file.
	Fixed a bug in the stack trace code that caused the initial exception address to be omitted from the actual stack trace.
	This version was built with TurboPower Orpheus 2.0 E3.