NEXTSTEP

Title: EOF Debugging Tips

Entry Number: 1820
Last Updated: 22 Feb 1995
Document Revision:

Keywords: gdb, debug, EOF, NXReportError, 100003, NSBundle, adaptor, EOAdaptor, SQL, MallocDebug

Questions

Q: Occasionally my EOF application exits and produces "Unknown error 100003 in NXReportError". This message is not very informative and makes debugging the application difficult. Is there a way to get more information than this?

Q: From gdb, how can I get information about private Foundation and EOF objects such as the concrete classes in a class cluster?

Q: I am trying to send messages to my adaptor from gdb while debugging my EOF application but gdb tells me that the adaptor objects do not respond to any selectors. What's the problem?

Q: How do I get a listing of the SQL code that is sent to the server?

Q: I am trying to compile my EOF application to run with MallocDebug and I get the following errors:

```
ld: multiple definitions of symbol __NXMallocCheck
/usr/lib/libMallocDebug.a definition of __NXMallocCheck in section
(__TEXT,__text)
/lib/libsys_s.a(malloc.o) definition of absolute __NXMallocCheck
(value 0x5002432)
ld: multiple definitions of symbol __NXDefaultMallocZone
/usr/lib/libMallocDebug.a definition of __NXDefaultMallocZone in
section (__TEXT,__text)
/lib/libsys_s.a(malloc.o) definition of absolute
```

```
__NXDefaultMallocZone (value 0x5002444)
```

What is happening?

Q: When my application raises an exception, I can't get a backtrace even if I am running it from within gdb. This makes debugging difficult. How can I get a backtrace when it raises?

Answers

Here are some useful EOF debugging tips to address these questions.

A: You can implement a top-level error handler that catches all unknown errors, and figure out what the error was and print a more useful message. Here is an example of some code that does this. First, call this someplace early in the app's execution such as **appDidInit:** or the **awakeFromNib** method for one of your top-level objects:

```
NXSetTopLevelErrorHandler(MyTopLevelErrorHandler);
```

"MyTopLevelErrorHandler" must be a function defined with this prototype:

```
void MyTopLevelErrorHandler(NXHandler *errorState);
```

Here is an example of an implementation that simply logs a message to the console with NSLog(). You can implement your own mechanism for presenting the error message, including raising an alert panel, logging a message with NSLog() or printf(), or any combination of these.

```
void MyTopLevelErrorHandler(NXHandler *errorState)
{
    if (errorState->code >= NSExceptionBase &&
        errorState->code <= NSLastException) {
        NSException *exception = (id) errorState->data1;

        NSLog(@"%@: %@\n", [exception exceptionName], [exception exceptionReason]);
    }
}
```

A: gdb's "print-object" command (abbreviated "po") sends a **description** message to an object and prints the results. Most Foundation and EOF objects implement **description** in a meaningful way and will give you information about their contents. You can implement **description** on your own custom objects to look at their contents from gdb. The syntax for the command is:

```
(gdb) po anObject
```

A: The NSBundle class has a class method called **stripAfterLoading:** that allows you to control whether or not symbols are stripped from bundles then they are loaded. Please see NeXTanswer 1720_Debugging_code_loaded_by_NSBundle for more information.

A: There are two ways to see the SQL code that is sent to the server. One way is to turn on debugging output in the adaptor. If you have an outlet to an EOController instance, you can do this with the following:

```
[[[(EODatabaseDataSource *)[controller dataSource] databaseChannel] adaptorChannel]
setDebugEnabled:YES];
```

The typecast is necessary because the data source for an EOController does not have to be an EODatabaseDataSource instance; if you know that it is, you can typecast it in this manner to get rid of the compiler warning.

Another way to see the SQL code is to become the delegate for the EOAdaptorChannel and respond to the **adaptorChannel:willEvaluateExpression:** method. To become the delegate, you can do the following at awakeFromNib time (again assuming that you have an outlet to an EOController instance):

```
[[[(EODatabaseDataSource *)[mainController dataSource] databaseChannel]
adaptorChannel] setDelegate:self];
```

Then that object can implement the EOAdaptorChannel's delegate method like this. In this example, the SQL code is logged with NSLog(). You can implement your own mechanism for displaying the SQL code including adding it to an on-screen Text objects, logging it with NSLog() or printf(), or any combination of these. Note that it is important to return

EODelegateApproves from this method in order for the code to be sent to the server and work correctly.

```
       - (EODelegateResponse)adaptorChannel:channel willEvaluateExpression:(NSMutableString
*)anExpression
       {
           NSLog(@"%@\n",anExpression);
           return EODelegateApproves;
       }
```

A: To resolve undefined symbols when using MallocDebug, you need to do the following:

1) remove the -all_load flag from OTHER_LDFLAGS in your Makefile.preamble.

```
OTHER_LDFLAGS =
```

2) hardlink your adaptor. For example, this will be the new entry in your Makefile.preamble if you use the Sybase adaptor:

```
# Additional relocatables to be linked in at this level
OTHER_OFILES = /NextLibrary/Adaptors/Sybase.dbadaptor/Sybase
```

3) link with MallocDebug:

```
# Additional libs to link apps against ('app' target)
OTHER_LIBS = -lMallocDebug
```

A: If you break on [NSException raise], you will be able to generate a backtrace, since this routine is called whenever an exception is raised. You can configure gdb so that it will always break on this routine by adding this line:

```
    b [NSException raise]
```

to the .gdbinit file in your project directory. You can't add it to the .gdbinit in your home directory, since that is loaded before the symbols for Foundation are loaded -- you need to put it in your project directory.

Note that after you have broken on this method, it is often useful to print the reason for the

exception. You can do this on Intel and Motorola platforms with the following command from gdb:

```
(gdb) po [*((id *)($fp + 8)) exceptionReason]
```

You can define a new command called "reason" to do this for you like this:

```
(gdb) define reason
po [*((id *)($fp + 8)) exceptionReason]
end
(gdb) document reason
Print the reason for an NSException.
When a program is inside of the method -[NSException raise], this
command prints out a textual reason for the exception.
end
```

You CAN add these commands to the .gdbinit file in your home directory so that they will always be available whenever you run gdb.

Valid for: EOF 1.0, EOF 1.1, NEXTSTEP 3.2 Developer, NEXTSTEP 3.3 Developer

See Also:

NeXTanswer #1720, "Debugging code loaded by NSBundle"