PROGRESS SOFTWARE

DEV-22: Catch Me If You Can – Practical Structured Error Handling

Peter van Dam
Future Proof Software

Exchange 08

Agenda

- **Introduction**
- FINALLY!
- The ProError Object
- CATCHing Errors
- Propagating Errors Up The Call Stack
- Limitations and workarounds

Relevant Exchange Sessions

- DEV-8: Structured Error Handling in the ABL
  Sarah Marshall
  Monday, 9 June, 3:15pm–4:15pm
- DEV-38: OpenEdge ABL Info Exchange
  Wednesday, 11 June, 10:45am–11:45am

What is Structured Error Handling?

- Modern – Object-oriented (but you don't have to know much about OO to use it)
- Can catch errors that could not be caught before
- Can co-exist and interoperate with traditional error handling
- This talk presents practical examples of adding structured error handling to existing applications

Agenda

- Introduction
- **FINALLY!**
- The ProError Object
- CATCHing Errors
- Propagating Errors Up The Call Stack
- Limitations and workarounds

First of All: FINALLY

Introducing the FINALLY block

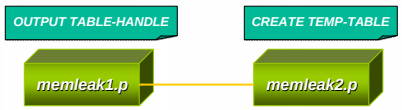
- FINALLY is a new block statement that you can use right away
- FINALLY always executes, whether an error was raised or not
- FINALLY must come after all other executable statements in a block
- Intended for clean-up code
- Examples: delete objects, close streams, write logs etc.

Using FINALLY to Prevent Memory Leaks

Solving the OUTPUT TABLE-HANDLE problem

- TABLE-HANDLE parameters create a TEMP-TABLE in the SESSION Widget Pool
- You are responsible for cleaning it up
- That can be difficult to manage
- Error conditions can also bypass your cleanup code

Creating A Memory Leak



- memleak2.p contains a CREATE TEMP-TABLE statement that creates the TEMP-TABLE in the current WIDGET-POOL
- The temp-table received by memleak1.p is created in the SESSION WIDGET-POOL

Demo: memleak.p

FINALLY in Practice

- FINALLY always executes, on success or failure
- From now on I put all my DELETE OBJECT statements in FINALLY blocks
- Starting with the examples in this presentation!


Agenda

- Introduction
- FINALLY!
- **The ProError Object**
- CATCHing Errors
- Propagating Errors Up The Call Stack
- Limitations and workarounds


The Mother of All Error Classes

Progress.Lang.ProError

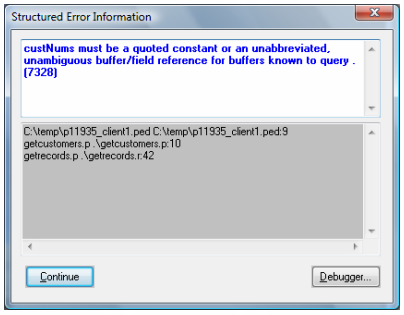
- CallStack
- NumMessages
- Severity

- GetMessage()
- GetMessageNum()


Enabling Stack Tracing

- Adds overhead – not enabled by default
- You must use the **`-errorstack`** startup parameter or **`SESSION:ERROR-STACK-TRACE`** to enable Stack Tracing
- Enable this feature in your startup procedure when **`SESSION:DEBUG-ALERT`** is **`TRUE`**
- This feature enables you to access the stack trace in the **`CallStack`** property of error objects

Enabling Stack Tracing



Agenda

- Introduction
- FINALLY!
- The ProError Object
- **CATCHing Errors**
- Propagating Errors Up The Call Stack
- Limitations and workarounds

Catching Errors

You can now catch everything that was hard to catch before

Catching System Errors

Catching Errors While Printing

```
OUTPUT TO print.txt.
FIND customer WHERE custNum EQ 1000 NO-LOCK.
PUT UNFORMATTED NAME SKIP address.
OUTPUT CLOSE.
```

- The error (customer 1000 does not exist) will end up in your output file and you might not even notice

Catching System Errors

Catching Errors While Printing

```
OUTPUT TO print.txt.
FIND customer WHERE custNum EQ 1000 NO-LOCK.
PUT UNFORMATTED NAME SKIP address.
OUTPUT CLOSE.
CATCH e AS Progress.Lang.ProError :
  MESSAGE "Error occurred during printing:" SKIP
    e:getMessage(1) VIEW-AS ALERT-BOX ERROR.
  DELETE OBJECT e.
END CATCH.
```

- Merely adding a CATCH block will address this situation

Demo: print.p

Methods On Built-in System Handles

The following code does not raise an error condition:

```
DEFINE VARIABLE hBuffer# AS HANDLE NO-UNDO.

CREATE BUFFER hBuffer# FOR TABLE "customer".

hBuffer#:FIND-UNIQUE("WHERE CustNum EQ 1000").
```

- The reason is that errors arising from methods on built-in system handles are treated… 'differently'.

Methods On Built-in System Handles (2)

Errors on built-in system handles are treated 'differently'

- The result is that an error message is displayed on standard output *and processing continues!*
- There is no error condition
- This is pretty bad as it is, and absolutely horrible when it happens on your AppServer
- When ***any*** relevant CATCH block is present, built-in ABL methods raise error.

Demo: built-in.p

Agenda

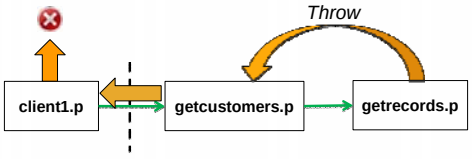
- Introduction
- FINALLY!
- The ProError Object
- CATCHing Errors
- **Propagating Errors Up The Call Stack**
- Limitations and workarounds

ROUTINE-LEVEL ON ERROR UNDO, THROW

- Changes all default ON ERROR phrases to ON ERROR UNDO, THROW in a file
- Must be the first statement in a file (before or after USING)

Propagating Errors Up The Call Stack

Catching AppServer™/ WebSpeed® Errors



Future Proof Software

Exchange 08

25 DEV-22: Catch Me If You Can – Practical Structured Error Handling

Demo: Propagating Errors back to the client

Client1.p

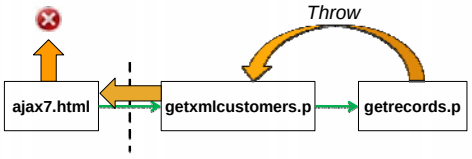
Future Proof Software

Exchange 08

26 DEV-22: Catch Me If You Can – Practical Structured Error Handling

Propagating Errors Up The Call Stack

Ajax example



Future Proof Software

Exchange 08

27 DEV-22: Catch Me If You Can – Practical Structured Error Handling

Propagating Errors Up The Call Stack

Generic Error Handling With Ajax

```
CATCH errorObject AS Progress.Lang.SysError:

  DEFINE VARIABLE iMessage# AS INT NO-UNDO.
  DEFINE VARIABLE hSAXWriter AS HANDLE.
  CREATE SAX-WRITER hSAXWriter.
  hSAXWriter:FORMATTED = TRUE.
  hSAXWriter:SET-OUTPUT-DESTINATION("STREAM", "WEBSTREAM").
  hSAXWriter:START-DOCUMENT().
  hSAXWriter:START-ELEMENT("error").

  DO iMessage# = 1 TO errorObject:numMessages:
    hSAXWriter:WRITE-DATA-ELEMENT("errorMessage", errorObject:getMessage(iMessage#)).
  END.

  hSAXWriter:END-ELEMENT("error").
  hSAXWriter:END-DOCUMENT().

  DELETE OBJECT hSAXWriter.
  DELETE OBJECT hTable NO-ERROR.
  DELETE OBJECT errorObject.

END CATCH.
```


Propagating Errors Up The Call Stack

Return any errors as XML

```
<error>
  <errorMessage>message 1</errorMessage>
  <errorMessage>message 2</errorMessage>
</error>
```

- In this way errors can be processed generically on the Ajax client as well

Propagating Errors back to an Ajax client

Ajax7.p

Agenda

- Introduction
- FINALLY!
- The ProError Object
- CATCHing Errors
- Propagating Errors Up The Call Stack
- **Limitations and workarounds**

Limitations of Structured Error Handling

- Cannot THROW an Error Object across an AppServer boundary
- Cannot catch STOP
- Cannot catch QUIT

Catching the STOP condition

```
DO ON STOP UNDO, RETRY:
  IF RETRY THEN
    UNDO, THROW
NEW Progress.Lang.AppError("Program not found",1).
  RUN nonexistent.p.
END. /* ON STOP... */


CATCH eError AS Progress.Lang.ProError:
  MESSAGE eError:getMessage(1)
          VIEW-AS ALERT-BOX ERROR.
  DELETE OBJECT eError.
END CATCH.
```

Conclusions

First structured error handling steps

- Start using FINALLY blocks wherever that makes sense
- Start adding CATCH blocks to your top-level procedure(s)
- Start adding ROUTINE-LEVEL ON ERROR UNDO, THROW statements to your secondary procedures
- Subsequently add CATCH blocks in your other blocks

Future Proof Software

Exchange 08

Questions ?

Future Proof Software

Exchange 08

Thank You

Future Proof Software

Exchange 08


Future Proof
SOFTWARE
www.futureproofsoftware.com
peter@futureproofsoftware.com
Exchange 08
37 DEV-22: Catch Me If You Can – Practical Structured Error Handling