CHAPTER 10 Handling Exceptions

Introduction

SCL provides two mechanisms for handling error conditions:

The program halt handler
: Program halt handlers typically allow your application to print a message, save some information, and then either try to continue execution or halt the application. The SCL generic program halt handler is sort of an all-purpose routine for handling program halts that occur for a variety of different reasons at any point in the program.

The CATCH and THROW statements
: The SCLException class and the CATCH and THROW statements enable you to define specific exceptions and recovery routines that are specific to each exception. You can define the exceptions and recovery routines in the locations in your code where the exceptions may be encountered, thus making error recovery code a natural part of the program.

Using the Program Halt Handler

The program halt handler is designed to handle unexpected run-time errors. The programHalt class contains methods that are called when certain run-time exceptions occur. By overriding these methods, you can specify whether an application should halt immediately or continue executing. You can control how exceptions are handled.

In the following example, the _onGeneric method creates a list named MSGS, inserts information about the location where the application failed into the list, and displays the list with the MESSAGEBOX function. You can use this code to create your own program halt handler.

```
class myHalt extends
   sashelp.classes.programHalt.class;
```

```
      _onGeneric:method / (STATE='O');
         dcl list msgs=makelist();
         rc = insertc(msgs, "SCL program failed at ", 1);
         rc = insertc(msgs, "Entry=" || entry, 2);
         rc = insertc(msgs, "Line=" || putn(lineNumber, "3.0"), 3);
         if (keywordType = 'function') then
            rc = insertc(msgs, "Function=" || keyword, 4);
         else
            rc = insertc(msgs, "Method=" || keyword, 4);

         rc = messageBox(msgs);

         /* continue execution */
         stopExecution = 'No';
      endmethod;
   endclass;
```

Note: **Entry**, **lineNumber**, **keyword**, and **keywordType** are all object attributes that are defined in the class sashelp.classes.programHalt.class. △

The _onGeneric method traps any error messages that are generated by SCL and saves them in the MSGS list. Developers can use this list to identify and fix potential problems in their code.

The programHalt handler must be declared at the beginning of your application. For example:

```
dcl myHalt obj = _new_ myHalt();
```

Your program can instantiate multiple programHalt handlers, or your program may instantiate only one handler, but then call a second program that instantiates its own handler. The last programHalt handler that is instantiated is the *current* programHalt handler. Only the current programHalt handler is active at any one time.

SCL uses a stack to keep track of programHalt handlers. Each time a programHalt handler is instantiated, the new instance is pushed onto the stack. The handler on the top of the stack is always the active handler. Before a program terminates it must terminate (using the _term method) its programHalt handler. For example:

```
obj._term();
```

Terminating a programHalt handler pops it from the stack, and makes the next programHalt handler on the stack the active handler.

For example, if your program instantiates the programHalt handler, and then calls another SCL program, the second program may also instantiate a programHalt handler. The second programHalt handler becomes the current programHalt handler. Before the second program ends, it must terminate the second programHalt handler. The first programHalt handler then becomes the current programHalt handler. If the second programHalt handler is not terminated, it will remain active even after the program that instantiated it has terminated.

Handling Exceptions with CATCH and THROW

All exceptions are subclasses of the SCLException class, which is a subclass of the SCLThrowable class. You can use the CLASS statement to define your own exception classes, and then use the THROW and CATCH statements to handle the exception.

Because an exception is a class, you can design the class to contain any information that is relevant to recovering from the specific exception. A simple exception class may contain only an error message. For example, the following class defines a subclass of SCLException called NewException, which defines an error message string named SecondaryMessage:

Example Code 10.1 NewException Class

```
Class NewException extends SCLException
   dcl string SecondaryMessage;
endclass;
```

You can then create a new instance of NewException and raise this exception with the THROW statement,as shown in the ThrowIt class:

Example Code 10.2 ThrowIt Class

```
Class ThrowIt;
  m: method;
  dcl NewException NE = _new_ NewException('Exception in method m');
  NE.SecondaryMessage = "There's no code in m!";
  throw NE;
  endmethod;
endclass;
```

Note: You must always declare a variable to hold the thrown exception. △

The code that processes the exception is enclosed in CATCH blocks. CATCH blocks can contain any code needed to process the exception, including more CATCH and THROW statements.

When an exception is thrown, normal execution of the entry stops, and SCL begins looking for a CATCH block to process the thrown class. The CATCH block can contain any statements needed to process the exception. For example, the following code prints the stack traceback at the point of the throw.

```
do;
dcl NewException NE = new NewException('Exception in method m');
NE.SecondaryMessage = "There's no code in m!";
throw NE;

catch NE;
   put NE.getMessage();         /* Print exception information. */
   call putlist(NE.traceback);
   put NE.SecondaryMessage=;    /* Print secondary message. */
endcatch;
end;
```

Note: CATCH blocks must always be enclosed in DO statements. △

The traceback information that is printed by this example is stored automatically by SCL when an exception is thrown. See "The SCLThrowable and SCLException Classes" on page 163 for more information.

Note: When a CATCH block has finished executing, control transfers to the end of the current DO statement, and the program resumes normal execution. If no exception has been thrown and SCL encounters a CATCH block, control transfers to the end of the current DO statement and execution resumes at that location. Therefore, any SCL statements the occur between CATCH blocks or following the last CATCH block within the same DO group will never be executed. Any SCL statements within the DO group that are not part of a CATCH block but must execute must be entered at the beginning of the DO group. △

After an exception is processed, program execution continues normally.

Example

Suppose you have the following class Y. This class defines a method called update that throws an exception that is an instance of the SCLException class.

```
import sashelp.classes;
class Y;
update: method;
   if (_self_.readOnly) then
      /* Throw an exception.  Set message via constructor. */
      throw _new_ SCLException('Cannot update when in ready-only mode');
endmethod;
endclass;
```

Class X defines method M, which declares a local variable to hold the exception, and then calls the update method, which throws the exception. The exception is then processed by the CATCH block for SCLE.

```
import sashelp.classes;
class X;
M: method;
   do;
   /* Declare the local exception variable. */
   dcl SCLException scle;
   dcl Y y = _new_ y();

   /* Call update method, which throws SCLEception. */
   y.update();

   /* Process the SCLException. */
   catch scle;
      /* Print exception information. */
      put scle.getMessage();
      call putlist(scle.traceback);
   endcatch;
   end;
endmethod;
endclass;
```

How SCL Determines Which CATCH Block To Execute

SCL uses the scope of the DO group that contains the CATCH block and the class of the exception to determine which CATCH block to execute.

- SCL first looks in the scope of the DO group where the exception was initially thrown. If SCL does not find a corresponding CATCH block, it expands its search outward to the next enclosing DO group.

  Note: If you are rethrowing an exception that has been thrown and caught at least once already, then SCL automatically passes the exception outside of the DO group where the exception was rethrown. △

  SCL continues expanding the scope of its search until it finds a corresponding CATCH block or it has searched the current SCL entry. If the current SCL entry does not contain a CATCH block for the thrown class, then the exception is passed up the stack to the calling entry where the process is repeated. If the calling entry contains a CATCH statement for the thrown class, then execution resumes at the location of the CATCH statement. If the calling entry does not contain a CATCH statement for the thrown class, then the exception is passed up the stack until SCL finds a corresponding CATCH statement or until the stack is completely unwound. If SCL does not find a corresponding CATCH statement, then the exception is treated the same as a program halt.

- SCL uses the class hierarchy to determine which CATCH block to execute. Within the scope that it is currently searching, SCL chooses the CATCH block for the class that is most closely related to the class of the thrown exception. For example, if the current scope contains a CATCH block for the thrown class, then SCL will execute that CATCH block. If the current scope does not contain a CATCH block for the thrown class, but does contains a CATCH block for the parent class of the thrown exception, then SCL will execute the CATCH block for the parent class. If none of the CATCH blocks in the current scope are related to the thrown class, then SCL continues its search for an appropriate CATCH block.

Suppose that in addition to the NewException class (see Example Code 10.1 on page 159) you define a subclass of NewException called SubException:

Example Code 10.3 SubException Class

```
Class SubException extends NewException
   ...code to process SubExceptions...
endclass;
```

As with all exceptions, SCL first searches the current DO group for a CATCH block that is related to the thrown class. In this example, because NEsub is an instance of SubException and SubException is a subclass of NewException, SCL will execute the CATCH block for NE because it is in the scope of the current DO group. The CATCH block for NEsub is in a different scope (the outer DO group), so it will not be executed unless the CATCH block for NE is modified to rethrow (see “Catching and Rethrowing Exceptions” on page 162) the exception. If the CATCH block for NE rethrows the exception, then both CATCH blocks will be executed.

Example Code 10.4 Nested DO Statements

```
dcl NewException NE;
dcl SubException NEsub;

do;
   do;
   NEsub = _new_ SubException('Exception in method m');
   NEsub.SecondaryMessage = "There's no code in m!";
   throw NEsub;

   catch NE;
      put NE.getMessage();         /* Print exception information. */
      call putlist(NE.traceback);
      put NE.SecondaryMessage=;    /* Print secondary message. */
      /* Could rethrow the NEsub exception if needed. */
   endcatch;
   end;

/* The following CATCH block will not be executed */
/* unless the CATCH block for NE rethrows the exception. */
```

```
catch NEsub;
   ...code to process NEsub exceptions...
endcatch;
end;
```

Catching and Rethrowing Exceptions

Each entry in the stack can process an exception and then pass it back up the stack by rethrowing it, which allows the calling entry to perform additional processing. Each entry can perform whatever processing is relevant to that entry.

```
do;
catch e1;
   ...process the exception...
   throw e1;  /* Rethrow the exception. */
endcatch;
end;
```

Note: If an exception is rethrown within a CATCH block, no CATCH block within the same scope can recatch the exception. The exception is passed out of the scope where it was thrown. △

If SCL finds a second CATCH block for E1 within the same SCL entry but outside of the scope of the DO group where the exception was thrown, then execution continues with that second CATCH block. If SCL does not find another CATCH block for E1 in that same SCL entry, then the exception is passed up the stack to the calling entry.

Suppose you have defined the NewException class (see Example Code 10.1 on page 159) and the ThrowIt class (see Example Code 10.2 on page 159). The following program section calls method M, which throws the exception NE. The two CATCH blocks catch, rethrow, and recatch the exception.

```
init:
dcl ThrowIt TI = _new_ThrowIt();
dcl NewException NE;
do;
   do;
   TI.m();

   catch NE;
      put 'caught it';
      throw NE;
   endcatch;
   end;

catch NE;
   put 'caught it again';
endcatch;
end;
return;
```

Note: You cannot define multiple CATCH blocks for the same exception within the same scope. △

Nested CATCH Blocks

You can nest CATCH blocks. For example, suppose you define the class W as follows:

```
class w;
m: method n:num;
do;
dcl e1 e1;
dcl e2 e2;
   do;
   if (n < 0) then throw _new_ e2();
      else throw _new_ e1();
   catch e2;
      put 'caught inner e2';
      do;
      dcl e1 e1;
      if (n<0) then throw _new_ e2();
         else throw _new_ e1();
      catch e1;
         put 'caught inner e1';
      endcatch;
      end;
   endcatch;
   end;
   catch e1;
      put 'caught outer e1';
   endcatch;

   catch e2;
      put 'caught outer e2';
   endcatch;
end;
endmethod;
endclass;
```

If you invoke method M with a negative argument as in the following program section:

```
init:
dcl w w = _new_ w();
w.m(-2);
return;
```

then the output would be

```
caught inner e2
caught outer e2
```

The SCLThrowable and SCLException Classes

All exceptions are subclasses of the SCLException class, which is a subclass of the SCLThrowable class. When an exception is thrown, SCL automatically stores the name of the entry that throws the exception, the line number where the throw occurs, and the stack traceback at the point of the throw. You can set the message attribute via the constructor when an instance of the exception is created. You can use the getMessage method to return the message.

Example Code 10.5 SCLThrowable Class

```
class SCLThrowable;
   public string(32767) message;
   public list traceback;  /* stack traceback */
   public string entry;    /* SCL entry name */
   public num line;        /* line number */

   SCLThrowable: public method s:string;
      message = s;
   endmethod;

   getMessage: public method return=string;
      return message;
   endmethod;
endclass;
```

Example Code 10.6 SCLException Class

```
class SCLException extends SCLThrowable;
   SCLException: public method /(state='o');
      _super("SCLException");
   endmethod;

   SCLException: public method s:string /(state='o');
      _super(s);
   endmethod;
endclass;
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Component Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Component Language: Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–495–0

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999