Table of Contents

EDM Exception Use

Complete:

Goals of page

The SWGuideEdmExceptionAnalysis page explains where this model came from and how it fits into error and message processing. What is covered here is the "Local change of standard program flow".

Where to find things

CMSSW/FWCore/Utilities/interface/Exception.h: The main exception class cms::Exception is contained here. There is documentation at the top of the file that explains its use.

CMSSW/FWCore/Utilities/CodedException.h: A class that allowed the category types to be enumerated (instead of being std::strings).

CMSSW/FWCore/Utilities/test/ExceptionDerived.cpp: An example of how to make an exception class that is derived from cms::Exception.

Exception classes

The exception hierarchy is small:

```
(A --> B means "A is derived from B")

edm::Exception --> cms::Exception --> std::exception
```

cms::Exception

This is the framework's main exception class. The Framework can recognize information contained within this exception and take appropriate action. This class should be used directly or as a base class for new exception types. _Any exception allowed to propagate from a processing module should be a cms::Exception or something derived from it. The action that the framework takes when one of these exceptions is caught is based on a category string given in the constructor.

Example use:

```
if(something wrong with data in the event)
   throw cms::Exception("CorruptData")
      << "It seems as though something is dreadfully wrong.\n"
      << "Unknown ID " << x << " found\n"
else if(too much time)
   throw cms::Exception("Timeout")
      << "Taking too long to process "
      << y << " number of hits\n";
```

The first argument in the constructor is the category. Guidelines for providing good category names will be given later in this document. The category name can be thought of as the general name of the problem. A category should exist for each unique type of action that might be configured.

This exception type (or anything derived from it) allows any object with a stream insertion operator (operator<<) defined to be added to the exception object directly from the constructor call as shown in the code segment.

If you want to establish an exception hierarchy, the base class should be cms::Exception if you allow any of the exceptions to leave your code. The documentation in the header file for this exception explains further how to use this class as a base class. You must propagate a category name to the base class for each unique derived class. One easy way to do this is to use the derived class name as the category name.

edm::Exception

Exceptions that are generated from calls to framework functions (e.g. access to products in the event) are of this type. The edm::Exception is actually a typedef for the class template CodedException. This template allows an enum to be used instead of strings for category names.

Exception handling rules

- Developers in general should not catch exceptions. As described below, the framework itself is responsible for catching exceptions in a configurable way.
- Developers should throw cms::Exceptions whenever they think they will not be able to perform the task they were called to do (eg. produce an object to be put into the event)

What the framework catches

The framework catches a fixed set of exceptions at every important place in its call stack. The exceptions caught are:

- edm::Exception
- cms::Exception
- std::exception

The important places these exceptions are caught include

- code surrounding a call to a processing module
- the schedule executor
- the event loop
- the cmsRun application

The cms::Exception allows for nesting or concatenation of exception information. At each place mentioned above, the framework will throw a new cms::Exception (if the cooresponding action is to do so) with the caught exception contents plus new context information. New context information may include:

- event ID (collision ID and relevant time stamps)
- active module type
- active module label
- current path
- product being operated on
- report on the action taken

depending on where the exception was caught. The final exception printout will contain a trace of all exceptions caught.

Altering framework flow

Currently only a filter module can change the flow of control in an EventProcessor. This is an event pass/no pass return code and is not considered an error condition. The only way to change the framework flow outside this specific case without terminating the job (actually exiting the EventProcessor) is to throw something that

is a cms::Exception. Private or vendor specific exceptions should not be allowed to escape out of a module because the framework will not know what to do with them and valuable context information may not be reported in a useful way. This is very rare, but if you do invoke code that will throw an unrecognized exception it should be caught and rethrown as the known exception, cms::Exception.

What to propagate

The module developers propagate high-level announcements of what has happened. They should not throw any sort of resolution - this action is up to the user configuring the job, not up to the code developer.

How exceptions will be caught

All exceptions will be caught by reference (non-constant).

Currently understood actions

There is currently a fixed set of actions that can be assigned to any of the category names found delivered in a cms::Exception. The framework currently understands the following actions.

- Rethrow: let the caller deal with the exception (This terminates the job with a non-zero return code).
- SkipEvent: stop further processing of this event and continue with the next event
- FailPath: stop processing in the path and mark it as failed, and continue witht he next path
- FailModule: stop the module and mark it as failed, and proceed with the next module
- IgnoreCompletely: pretend the exception never happened (if possible)

These actions apply for exceptions thrown while a module (e.g. an EDProducer, EDFilter, EDAnalyzer, or OutputModule) or input source is processing an event. Exceptions thrown at other times, such as when processing a begin or end Run, always result in a Rethrow action. NOTE: Prior to CMSSW_3_1_0_pre10, exceptions thrown while processing an input source would always be rethrown.

The above actions occur as stated if thrown during module execution on a path (as opposed to an endpath). If thrown during the execution of an input source, there is no path involved, so FailPath or FailModule is treated as SkipEvent. If thrown while executing a module on an endpath, FailPath or SkipEvent is treated as FailModule, so that other modules on the endpath are unaffected.

Parameter set options

Each of the exception categories can be assigned an associated action at runtime. The syntax for making the assignment is as follows.

Warning: The "options" pset below is untracked. This reflects the latest prerelease. It is possible that older prereleases require the word untracked to not be there.

```
# if "options" is present, the framework will use it
options = cms.untracked.PSet(
  # skip the event if processing modules produce
  # category "A", "B", or "C" exceptions
  SkipEvent = cms.vstring( "A", "B", "C" ),
  FailModule = cms.vstring( "Q" )
)
```

The framework will look for string vectors for each of the actions names it understands. The vectors contain the category names that will be assigned to that action.

Framework categories

The edm/framework produces exceptions with the following category names. Next to each is the default action taken by the framework: The default action for any exception not found on this list is 'Rethrow'.

Category Name	Default Action
ProductNotFound	Skip Event
DictionaryNotFound	Rethrow (stops the job)
NoProductSpecified	Rethrow (stops the job)
InsertFailure	Rethrow (stops the job)
Configuration	Rethrow (stops the job)
LogicError	Rethrow (stops the job)
UnimplementedFeature	Rethrow (stops the job)
InvalidReference	Skip Event
NullPointerError	Skip Event
NoProductSpecified	Rethrow (stops the job)
EventTimeout	Skip Event
EventCorruption	Skip Event
FileInPathError	Rethrow (stops the job)
FileOpenError	Rethrow (stops the job)
FileReadError	Rethrow (stops the job)
FatalRootError	Rethrow (stops the job)
MismatchedInputFIles	Rethrow (stops the job)
ProductDoesNotSupportViews	Rethrow (stops the job)
ProductDoesNotSupportPtr	Rethrow (stops the job)
NotFound	Skip Event

Notes

The **FileOpenError**, **FileReadError**, **FatalRootError**, and **MismatchedInputFIles** categories were added for CMSSW_2_1_10. They do not exist in prior releases..

We do not yet support installing user-supplied callback objects for actions. This addition would allow user code to be invoked, most likely with the current event, when an action is about to be taken as a result of an exception being caught.

Guidelines for category names

This section is not complete. There is some information about naming categories in the SWGuideEdmExceptionAnalysis page.

Review Status

Reviewer/Editor and Date (copy from screen)	Comments
Main.jbk - 02 Sep 2006	page author
JennyWilliams - 31 Jan 2007	editing to include in SWGuide
Main.wmtan - 09 Oct 2008	bring up to date
ChrisDJones - 12-Nov-2009	switched to python config language

Responsible: Main.jbk
Last reviewed by: Reviewer

This topic: CMS > SWGuideEdmExceptionUse
Topic revision: r22 - 12-Nov-2009 - 14:56:44 - ChrisDJones