Table of Contents

EDM Paths and Trigger Bits

Complete: ▨▨□

Introduction

The paths in a parameter set script file together form a set of bits that can be used for event filtering. Each bit is true if the path claimed success (passed or accept) in processing the event or false if the path claimed failure (fail or reject). Each path is assigned one bit position according to the paths position in the configuration script. The first encountered path is assigned bit position zero. Endpaths do not participate in bit assignment.

The initial release of a framework scheduler that manages trigger bits is available. The purpose of this page is to describe how one can work with the trigger bit information and with the statistics kept by the framework concerning path and module event passes and failures.

The exception handling configuration affects event and trigger path processing.

Reminder of path syntax

A configuration script for cmsRun can contain any number of *path* and *endpath* statements. The syntax for each of these is identical. Later sections of this document explain the actual processing differences between the two path types.

The general form of the path (endpath) is: `process.pathname =cms.Path(process.moduleName * process.moduleName2.......)`

where

- *pathname* is any valid name you assign to the path
- moduleName is the name of a module or sequence to be run in this path
- The operator between two module names can be "+" or "*". In this context, these operators have the exactly the same meaning and behavior.

Parentheses can appear as python allows them in expressions, but they do not affect anything externally visible in the behavior or output of the process. They only effect the internal evaluation order. They might improve code readability by grouping modules visually in some cases.

The modules in a path are executed in order from left to right. Execution is stopped when a filter is executed that returns false (meaning fail or reject). For any given path, the overall result is the logical "and" of all the filter results.

Note that the boolean result of any *filter* mentioned in a path can be reversed by adding a "~" in front of it. The boolean result of any *filter* mentioned in a path can be ignored (always pass) by passing the module to the function `cms.ignore` . E.g.

```
cms.ignore(foo)+bar
```

`bar` will always be run right after `foo` because the result of `foo` will always be ignored.

Trigger results in the event

There is an EDProduct called TriggerResults which stores the pass or fail result of each path. The Framework automatically writes an object of this type into every event. The TriggerResultInserter is the producer that places this product in the event. It is run with a fixed "module_label" of **TriggerResults**. The TriggerResultInserter is run after all normal paths have completed, but before any end paths have run.

The order the results are written into the TriggerResults object is the same as the order the paths appear in the configuration. The interface of the TriggerResults allows one to

- get out a bit pattern recording the status of each path: pass, failure, ready, or error (the last two only occur when there is some kind of problem)
- get the index of the module in the path whose status is fail
- get the parameter set ID of the parameter set containing the trigger names (most users should not use this, but use the utilities in the TriggerNamesService instead).

The TriggerNamesService and associated TriggerNames class have functions that allow one to

- get the names of all the bits in the order corresponding to TriggerResults
- get the index of a bit by trigger path name
- get the name of a specific bit from its index
- these functions work for the current process or if given a TriggerResults object they work for a current or previous process (as of release 1_6_0_pre1)

Prior to release 1_6_0_pre5, there were configuration parameters named "listOfTriggers" and "makeTriggerResults". These parameters are no longer used by the Framework and have no effect. Although harmless, as a matter of cleanup they should be removed from all configuration files.

Liz's notes on supporting more extensive saving of results

- It is very difficult to exactly reproduce what the HLT does offline. In a running experiment the code used in the trigger evolves at a much slower pace then the offline due to the need for extra stability. Often after an initial commisioning period the only motivation for udpating the code is retiring of the platform (RH7.3) that the code runs on. Another major obsticle to reproducability is the undertermined state of the calibrations used in the HLT. The trigger table is a very complex application O(200) paths. Not many in the collaboration are expert enough to run it reliably with exactly the right configuration (including DB specification)
- The HLT/online selection group should have a standard set of well designed single purpose (wehre possible) filter modules that they control and debug.
- Because of the above it is important to save the values of the quantities that you cut on for each event. A miminal solution would save for each filter a type key and the small number of floats necesary
- If the possible filters are known in advance it is possible to minimize and customize the history information saved for them.
- A small percentage (.5% at CDF) of auto accept data is needed for both L1 and HLT triggers. These triggers should save the full HLT reconstructed event. In this way suspected BUGS in the trigger can be investigated after the fact.
- These triggers are not a substitute for well designed backup triggers, however they are usefull in doing offline studies of optimizing trigger rates of the list as a whole.

Summary report

The framework can produce a summary report of path and module pass/fail information. The report is activated by a bool parameter in the "options" pset:

```
process = cms.Process("PROD")
process.options   = cms.untracked.PSet( wantSummary = cms.untracked.bool(True) )
```

Warning: see green note above on use of untracked in front of the options pset.

The report only include path and module summaries. The column headings for the path report include:

- Bit: the number of the bit being reported
- Passed: the number of events passed by this path
- Failed: the number of events failed (rejected) by this path
- Name: the name of this path

The column headings for the module report include:

- Passed: how many events were passed by this module
- Failed: how many events were failed by this module
- Run: how many times this module was run
- Visited: how many times this module was asked to run by paths (it only runs once per path and returns cached results the rest of the time.
- Error: how many events caused exceptions to to occur in this module
- Name: the module label name of this module

There is an addition event total summary that includes total events processed, total number that passed any trigger path, and total number that were failed by all trigger paths.

Output modules using trigger results

Any standard output module can be configured to select events for output depending on the value of bits in the TriggerResults.

Warning: This section of this document reflects configuration file rules for the latest prerelease. In very early releases of this feature, the "SelectEvents" pset could not have "untracked" in front of it. You may need to remove "untracked" from "SelectEvents" if you are working in an older release.

Warning: This document reflect capabilities of the latest prerelease. In releases earlier than 2.0, wildcards other than `"*"` and `"!*"` are not supported, and the syntaxes "exception@p1" and "exception@*" and "p1&noexception" and the like are not supported.

The way to configure which events are to be accepted by an output module is via the *SelectEvents* vstring (in the PSet *SelectEvents*), which lists one or more path specifiers to look for to accept the event. The answer of whether or not to keep the event is an **"or"** of all the answers for the named paths.

Usual path specifiers look for some trigger bit indicating **Pass**. Negated path specifiers, starting with !, look for **every** trigger bit matching the specification to be **Fail** (see NegatedWildcardPathSpecifiers.) Path specifiers starting with exception@ look for events that have trigger bits indicating **Exception**. The syntax also provides ways to specify that events selected are required to have no trigger bits indicating **Exception**, thus providing an easy way to create "clean" collections of events.

Path Specifiers

The syntax available for specifying selection criteria ("Path Specifiers") to select events that have some bits in the **Pass**, **Fail**, or *Exception state, is described here by examples. For the purposes of these examples, assume that the following path expressions exist in the configuration script:

```
process.p1 = cms.Path (process.A*process.B*process.C)
process.p2 = cms.Path (process.D*process.E*process.F)
process.p3 = cms.Path (process.G*process.H)
process.p4 = cms.Path (process.I*process.J)
```

```
   process.hlt_q1 = cms.Path (process.K*process.l)
   process.hlt_r2 = cms.Path (process.M*process.N)
   process.calib1 = cms.Path (process.P)
   process.calib2 = cms.Path (process.Q)
   process.pt_hi = cms.Path (process.R)
```

The vector of strings named SelectEvents lists the paths that are of interest. The answer of whether or not to keep the event is an **"or"** of all the answers for the named paths.

Selecting Pass for Some Trigger Bit(s)

If we want to select all events that pass p1, then do the following

```
  process.outp1=cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('savep1.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('p1')
                )
        )
```

If we want to select all events that pass p2 or p3, then do the following

```
  process.outp1=cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('savep23.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('p2','p3')
               )
        )
```

If we want to select all events that pass p2 where p2 was a path in a prior process named "HLT", then do the following

```
  process.outp2hlt=cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('savep2hlt.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('p2:HLT')
                )
        )
```

If we want to select all events that pass p* (in the example, p1, p2, p3, p4, or pt_hi), then we can specify each of these separately, or do the following

```
 process.outp = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_pstar.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('p*')
                )
        )
```

If we want to select all events that pass any trigger matching *2 (in the example, hlt_r2 and calib2), then do the following

```
 process.anything2 = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_anything2.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('*2')
                )
        )
```

Both * and ? (for single-character match) are supported. If we want to select all events that pass any trigger matching p? (p with exactly one charactor afterward; in the example, p1, p2, p3, and p4 but not pt_hi), then do the following

```
process.outpn = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('save_pn.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('p?')
               )
       )
```

If we want to select all events that pass any trigger, then do the following

```
 process.outanytrigger = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('save_anytrigger.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('*')
               )
       )
```

Demanding No Exceptions

An addition to the Path Specifier syntax permits selecting events that pass some trigger bits, and contain no trigger bits indicating **Exception**. This is done by adding `&noexception` (which may be shortened to `&noex`) after the path. This ability may be used to create some "clean" streams, plus one or more streams containing events with exceptions to be studied.

If we want to select all events that have no exceptions and pass p1, then do the following

```
process.outp1 = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('savep1.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('p1&noexception')
               )
       )
```

If we want to select all events that have no exceptions and pass p2 or p3, then do the following

```
 process.outp23 = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('savep23.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('p2&noex','p3&noex')
               )
       )
```

If we want to select all events that have no exceptions and pass p2 where p2 was a path in a prior process named "HLT", then do the following

```
 process.outp2hlt = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('savep2hlt.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('p2&noex:HLT')
               )
       )
```

If we want to select all events that have no exceptions and pass p*, then we can do the following

```
process.outp = cms.OutputModule("PoolOutputModule",
       fileName = cms.untracked.string('save_pstar.root'),
       SelectEvents = cms.untracked.PSet(
               SelectEvents = cms.vstring('p*&noexception')
```

```
                )
        )
```

If we want to select all events that have no exceptions and pass any trigger, then do the following

```
  process.outanytrigger = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_anytrigger.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('*&noexception')
                )
        )
```

Negated Path Specifiers

We can specify that we wish to select events in which a certain trigger bit (or bits) is in the **Fail** state, by prepending an exclamation point (`!`) to the path specifier. This may optionally be combinded with the `&noexception` specification.

If we want to select all events that fail p4 (invert p4 decision), then do the following

```
  process.outnot4 = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_not4.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('!p4')
                )
        )
```

If we want to select all events that have no exceptions and pass p3 *or* fail p4 (invert p4 decision), then do the following

```
process.outnot3out4 = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_p3not4.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('p3&noex,'!p4&noex')
                )
        )
```

If we want to select all events that **Fail** *every* trigger matching hlt* (in the example, hlt_q1 and hlt_r2), then do the following

```
 process.outnothlt = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_nothlt.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('!hlt*')
                )
        )
```

If we want to select all events that **Fail** *every* trigger, then do the following

```
 process.outfailall = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_failall.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('!*')
                )
        )
```

If we want to select all events that have no exceptions, then do the following

```
  process.outnormal = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_normal.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('*&noex,'!*&noex')
                )
        )
```

Notes:

1. Placing a "!" in front of any name means that if the bit is **Fail**, then we accept the event. The **Ready** or **Exception** state does not cause the event to be selected.
2. The meaning assigned to `"!*"` or <"!p*"> is that every one of the indicated bits must be **Fail** to cause selectoin

Exception Path Specifiers

We can select events based on the presence of **Exception** states in one or more of the trigger bits.

If we want to select all events that have any trigger bit indicating an exception, then do the following

```
  process.outexceptions = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_exceptions.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('exception@*')
                )
        )
```

If we want to select all events that have bit p1 indicating an exception, then do the following

```
  process.outexceptionp1 = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_exceptionp1.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('exception@p1')
                )
        )
```

If we want to select every event, both with and without exceptions, then either omit the SelectEvents PSet for that output module, or, equivalently, do the following

```
 process.outevery = cms.OutputModule("PoolOutputModule",
        fileName = cms.untracked.string('save_every.root'),
        SelectEvents = cms.untracked.PSet(
                SelectEvents = cms.vstring('*','!','exception@*')
                )
        )
```

The syntax `"!exception@xyz"`, which might mean "select all events that do not have **Exception** state of any bits whose name matches the expression `xyz`," is not supported.

Additional notes about Path Specifiers

1. Placing a "*" into the list is the same as listing all the triggering paths in the list.
2. Placing a wildcard expression into the list is the same as listing all the triggering paths that would match that expression.
3. Placing a "!" in front of any name means that if the bit is **Fail** then we may accept the event.
4. Placing a "!" in front of any wildcard expression means that if all the bits for triggers matching that expression are **Fail**, then we may accept the event.
5. If the *SelectEvents* PSet is left out, then all events are selected for output, regardless if a TriggerResults object is in the event or not.
6. A TriggerResults object does not need to be in the event if the output module is configured to select all events for output.

7. If SelectEvents = { "*", "!*", "exception@*" }, this is equivalent to selecting all events for output (or leaving out the SelectEvents section altogether).
8. If any trigger is needed to make a selection decision, then a TriggerResults object must be present in the event.
9. Paths themselves provide the capability of forming the **"and"** operation of multiple **Pass** results, so selection is only based on the **"or"** of all the required conditions.

Beyond **Pass**, **Fail**, and **Exception**, there is a fourth possible trigger state: **Ready**. The event processor should not reach any output module with any trigger bits in the **Ready** state, unless some paths were skipped because of an exception, in which case one of the other trigger bits will be in the **Exception** state. Details of the behavior of various wildcarded selection path specifications, when some trigger bits are **Ready**, appear in BehaviorOfEventSelector.

Other (static) functions in EventSelector

This is for people setting up creation of streams of events:

In order to assist a later stream in identifying which output module produced an event in a stream (so that mixes of events from different output modules, which might include different products, don't go into the stream being produced), the EventSelector class contains a static function maskTriggerResults.

In order to help ensure (at initialization) that the configuration of Path Specifiers for the various output modules does not lead to ambiguities about which output module wrote an event, the EventSelector class contains a static function testSelectionOverlap.

Details of the behavior of these functions appear in BehaviorOfEventSelector.

Generic prescale module

FWCore, after CMSSW_0_5_0_pre3, provides a generic prescale filter module called **Prescaler**. Below is an simple example of how to configure it.

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("TEST")

import FWCore.Framework.test.cmsExceptionsFatalOption_cff
process.options = cms.untracked.PSet(
  wantSummary = cms.untracked.bool(True),
  Rethrow = FWCore.Framework.test.cmsExceptionsFatalOption_cff.Rethrow
)

process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(20)
)

process.source = cms.Source("EmptySource")

process.pre1 = cms.EDFilter("Prescaler",
    prescaleFactor = cms.int32(5)
)

process.pre2 = cms.EDFilter("Prescaler",
    prescaleFactor = cms.int32(2)
)

process.print1 = cms.OutputModule("AsciiOutputModule")

process.print2 = cms.OutputModule("AsciiOutputModule",
    SelectEvents = cms.untracked.PSet(
        SelectEvents = cms.vstring('p2')
    )
)

process.p1 = cms.Path(process.pre1)
process.p2 = cms.Path(process.pre2)

process.e1 = cms.EndPath(process.print1)
process.e2 = cms.EndPath(process.print2)
```

Notes concerning path specifications

The scheduler only runs modules at most once per event. Modules are run the first time they are encountered. If a module occurs in more than one path, then a cached result is returned. If the module threw an exception, then the module will rethrow the same exception if it is encountered in a later path.

The decision of any module in any path can be reversed by adding a bang "!" to the front of the name. The reversal of the decision only applies to the current path and is strictly a property of the path instance (it is not a property of the module). Similarly, the decision of any module in any path can be ignored (always pass) by adding a "-" in front of the name (first implemented in 1_3_0_pre1).

```
  process.a = cms.Path (process.F*process.Y*process.Z)
  process.b = cms.Path (~process.F*process.U*process.V)
```

Here path 'b' proceeds if 'F' is false and path 'a' proceeds if 'F' is true.

Review Status

Reviewer/Editor and Date (copy from screen)	Comments
Main.jbk - 30 Jan 2006	last added page content
JennyWilliams - 31 Jan 2007	editing to include in SWGuide
MarkFischler - 18 Feb 2008	enhancement of Output Modules using Trigger Results
MarkFischler - 5 Mar 2008	Wildcard and exception@ in Output Modules using Trigger Results,

Responsible: Main.jbk
Last reviewed by: Sudhir Malik- 4 April 2009

This topic: CMSPublic > SWGuideEDMPathsAndTriggerBits
Topic revision: r39 - 2017-07-07 - DavidDagenhart