Table of Contents

CMS MessageLogger Setup Examples

Goal of this page

This page gives examples of CMS MessageLogger service setup examples.

Introduction

The CMS MessageLogger Service is meant to allow code to log messages to a unified message logging and statistics facility. While reasonable default behaviors are provided, the behavior of the MessageLogger can be adjusted via lines in the job's configuration file (*_cfg.py).

Message Logging Concepts

- **Destination** - The MessageLogger is capable of routing messages to multiple **Destinations**. The behavior -- the nature of which messages would appear and which would be ignored -- can be controlled for each destination. At present, two types of **destinations** are supported:
    - Output files, the names of which can be specified in the job's .cfg file. (If the file specified already exists, the messages will be appended to the file.)
    - Streams `cerr` and `cout`. (Because our parameter set syntax treats these names specially, creation of an output file named cerr or cout is not supported.)

- **Message ID** - Each message is given a category when issued. This is intended to represent what this message is about. For example, several messages with different detailed content could all be of type **tracking**. Limits can be imposed on the reporting of messages in a given category, and statistics can be obtained by category.

- **Severity** - `LogDebug`, `edm::LogWarning`, `edm::LogInfo`, and `edm::LogError` produce messages with severity `DEBUG`, `INFO`, `WARNING`, and `ERROR` respectively. These are ordered with `DEBUG` as the lowest severity and `ERROR` as the highest.

- **Threshold** - For any given destination, the .cfg file can specify that every message with a severity lower than a given `threshold` shall be ignored. By default, a destination will have a threshold of `INFO`; responses to messages issued via LogDebug can be enabled by setting a threshold to `DEBUG`.

- **Limit** - For each message category, or for messages in general, or for a specified severity level, the .cfg file can instruct the logger to ignore messages after some number (the `limit`) have been encountered. (Actually, beyond the applicable limit, an occasional further message will appear, based on an exponential backoff. Thus, if the a limit is set at 5, and 100 messages of that id are issued, then that destination will react to the messages number 1, 2, 3, 4, 5, 10, 15, 25, 45, and 85.) A limit of zero will disable reporting that category of messages.

- **Timespan** - When a limit is set, it is possible to specify that if no occurrences of that type of message are seen in some number of seconds (the `timespan`), then the count toward that limit is to be reset. Thus if you wish to suppress most of the thousands of warnings of some type expected at startup, but would like to know if another one happens after a gap of ten minutes, this can be specified as a timespan of 600.

Routine MessageLogger Parameters

The following is a portion of a .cfg file requesting the MessageLogger Service, and setting up setting a

destination which will write messages to the file **messages.log**. The default threshold and limits will apply: No LogDebug messages will be reported, and only the first five messages of each category will be reported.

```
process.MessageLogger = cms.Service("MessageLogger",
                            estinations   = cms.untracked.vstring('messages')
)
```

The parameters controlling the MessageLogger are to be **untracked**.

Example _cfg.py File Exercising Some Options

The following is an example .cfg file, requesting the MessageLogger Service, setting up some destinations, and specifying some thresholds for how "severe" a message must be in order for it to appear in each destination.

```
import FWCore.ParameterSet.Config as cms
process.load("FWCore.MessageLogger.MessageLogger_cfi")
process.MessageLogger = cms.Service("MessageLogger",
       destinations   = cms.untracked.vstring(
                                             'detailedInfo'
                                               ,'critical'
                                               ,'cerr'
                    ),
       ,critical       = cms.untracked.PSet(
                       , threshold = cms.untracked.string('ERROR')
        ),
       detailedInfo   = cms.untracked.PSet(
                       threshold = cms.untracked.string('INFO')
       ),
       cerr           = cms.untracked.PSet(
                        threshold = cms.untracked.string('WARNING')
        )
)
 }

# set the number of events
process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(200)
)

process.myAnalysisModule = cms.EDAnalyzer('ModuleThatIssuesMessages')
process.p = cms.Path(process.myAnalysisModule)
```

In the above example, the files produced would be detailedInfo.log and critical.log, along with messages routed to cerr. Since the use of the dot character in a PSet name is discouraged, the MessageLogger assigns a default extension .log. The extension used can, however, be specified to be something else, as in:

Enabling LogDebug Messages

The following is a portion of a .cfg file appropriate for a job that will run code instrumented with many LogDebug messages. This hypothetical user cares only about those LogDebug messages in the category `interestingToMe` and, in this file, prefers not to see any other LogDebug or LogInfo messages.

```
process.MessageLogger = cms.Service("MessageLogger",
                     destinations       =  cms.untracked.vstring('debugmessages'),
                     categories         = cms.untracked.vstring('interestingToMe'),
                     debugModules  = cms.untracked.vstring('*'),

                     debugmessages          = cms.untracked.PSet(
```

```
                                                    threshold =  cms.untracked.vstring('DEBUG'),
                                                    INFO       =  cms.untracked.int32(0),
                                                    DEBUG   = cms.untracked.int32(0),
                                                    interestingToMe = cms.untracked.int32(10000000)
                                                        )
)
```

By default, all LogDebug messages issued from any modules would be rapidly discarded. This user chooses to enable the LogDebug messages issued by all modules, via `debugModules  = cms.untracked.vstring('*')`. (Instead, LogDebugs from a set of specific modules could be enabled; see Controlling LogDebug Behavior: Enabling LogDebug Messages for details.

Even if LogDebug messages are not rapidly discarded, destination have a default threshold of INFO, so no LogDebug messages would be reported. Here, for the debugmessages.txt destination, the user causes LogDebug messages to be reported by `threshold =  cms.untracked.vstring('DEBUG')`. If the PSet for `debugmessages` were to end there, all LogDebug messages would be reported by this destination.

This user, however, desires more selectivity. The next two lines establish that for any category of messages without its own limit specification, if the message has severity INFO or DEBUG, use a limit of zero (which means don't report those messages). Finally, the *_cfg.py file overrides this for messages in the category interestingToMe, permitting messages of that category to be reported.

Suppressing Messages From Specific Modules

Occasionally, there will be a job that requires LogInfo level output to some destination, but which relies on one or two modules which are overly verbose when the threshold is LogInfo. Since neither limits nor thresholds are module-specific, we have provided a way to suppress all messages from a given module, at a given severity level.

The messages subject to this suppression can be LogWarning, LogInfo, or LogDebug. (As always, the behavior with respect to LogVerbatim and LogTrace is controlled by the control of LogInfo and LogDebug, respectively). LogError should not be used to convey ignorable information; we do not provide for suppressing LogError messages.

The following is a portion of a .cfg file appropriate for a job that will run code instrumented with useful messages, but which involves overly verbose modules which this hypothetical user wishes not to see:

```
process.MessageLogger = cms.Service("MessageLogger",
                     destinations       =  cms.untracked.vstring('messages.txt'),
                     debugModules   = cms.untracked.vstring('*'),
                     messages           = cms.untracked.PSet(
                                                threshold =  cms.untracked.vstring('DEBUG')
                                                    ),
                    suppressDebug  = cms.untracked.vstring('wordyModule'),
                    suppressInfo       = cms.untracked.vstring('verboseModule', 'otherModule'),
                    suppressWarning= ms.untracked.vstring('cryWolfModule')
)
```

This user has chosen to enable the LogDebug messages issued by all modules, via vstring debugModules = { "*" }. However, certain messages from four modules will be totally ignored at the point of origin:

- LogWarning, LogInfo or LogVerbatim, and LogDebug or LogTrace messages issued when the module label is "cryWolfModule".
- LogInfo or LogVerbatim, and LogDebug or LogTrace messages issued when the module label is "verboseModule" or "otherModule".

- LogDebug or LogTrace messages issued when the module label is 'wordyModule'.

It is worth noting that such suppressed messages will not even be put on the queue of messages to be handled by the Message Service. In consequence:

* Items streamed to these messages will not be needed in any way. In optimized builds, their computation may be eliminated by the compiler unless that computation has other side effects. * Suppressed messages will not appear in any message statistics accounting. * Suppressed messages are not counted toward any limits or "reportEvery" intervals, and do not affect any timespans.

And the suppress directives belong in the MessageLogger PSet, rather than in individual desintation PSets.

Also, note that the suppression is meant to apply to specific modules. For example, the syntax suppressInfo = { "*" } (intended to mean suppress all Info messages from every module) is not supported. Use the threshold mechanism to turn off all messages of various severities issued by every module.

Routing Messages to log4cplus

The MessageLogger will route messages to log4cplus if the .cfg file requests the MLlog4cplus service. The following is an example .cfg file, requesting the MLlog4cplus service, and specifying a threshold for how "severe" a message must be in order for it to be routed to log4cplus.

```
process.MessageLogger = cms.Service('MLlog4cplus')
process.MessageLogger = cms.Service("MessageLogger",
                     destinations =  cms.untracked.vstring('detailedInfo' ),
                     log4cplus     =  cms.untracked.PSet(
                                              threshold =  cms.untracked.string('ERROR')
                                            ),
)

process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(10))
process.myAnalysisModule = cms.EDAnalyzer('ModuleThatIssuesMessages')
process.p = cms.Path(process.myAnalysisModule)
```

The four severities of MessageLogger messages correspond to the four directives for issuing to log4cplus. For instance, LogWarning("category") leads to a LOG4CPLUS_WARN(...) call.

log4cplus dispatches to "appenders". For example, a fileAppender may be established to write the text to a file, or a consoleAppender to write to the job console. It is up to the user code or configuration file to establish which appenders are wanted. In the current implementation, the MLlog4cplus service automatically establishes a fileAppender writing log4cplus.ouput.

Examples of Limits and Timespan Parameters

```
import FWCore.ParameterSet.Config as cms
process.load("FWCore.MessageLogger.MessageLogger_cfi")
process.MessageLogger = cms.Service("MessageLogger",
       destinations   = cms.untracked.vstring(
                                            'detailedInfo'
                                              ,'critical'
                                              ,'jobdebug'
                                              ,'anotherfile'
                                              ,'cerr'
        ),
      categories = cms.untracked.vstring(
                                         'unimportant'
                                        ,'trkwarning'
                                        ,'serious_matter'
```

```
      ),
      critical       = cms.untracked.PSet(
                        threshold = cms.untracked.string('ERROR'),
                        default     = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(10),
                                                          timespan = cms.untracked.int32(180)
                                                         ),
                        serious_matter = cms.untracked.PSet(
                                                         limit=cms.untracked.int32(100000)
                                                         )
       ),
      detailedInfo   = cms.untracked.PSet(
                        threshold = cms.untracked.string('INFO'),
                        default     = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(10),
                                                          timespan = cms.untracked.int32(60)
                                                         ),
                        WARNING  =      cms.untracked.PSet (
                                                          limit = cms.untracked.int32(100),
                                                          timespan = cms.untracked.int32(60)
                                                         ),
                        ERROR       = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(100),
                                                          timespan = cms.untracked.int32(60)
                                                         ),
                      trkwarning  = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(20),
                                                          timespan = cms.untracked.int32(1200)
                                                         ),
                        unimportant  = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(5)
                                                         ),
                        serious_matter  = cms.untracked.PSet (
                                                          limit = cms.untracked.int32(1000000)
                                                         )
       ),
       cerr           = cms.untracked.PSet(
                         threshold = cms.untracked.string('WARNING')
       ),
       jobdebug           = cms.untracked.PSet(
                                  default = cms.untracked.PSet(
                                                       limit = cms.untracked.int32(1000000)
                                 )
        ),
        anotherfile       = cms.untracked.PSet(
                                  serious_matter = cms.untracked.PSet(
                                                       limit = cms.untracked.int32(1000)
                                  )
        ),
        default                   =  cms.untracked.PSet(
                                                        limit = cms.untracked.int32(10),
                                                        timespan = cms.untracked.int32(60)

       )
)

process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(10))
process.myAnalysisModule = cms.EDAnalyzer('ModuleThatIssuesMessages')
process.p = cms.Path(process.myAnalysisModule)
```

Example of Parameter

Consider a job which runs through many thousands of events, and where the input module issues, for each event, some message saying that event thus-and-such has been read. The user may not wish to suppress all of these messages, or even set a limit after which the messages are exponentially throttled back. Instead, it is reasonable to want to see every 100-th (or 10000th or whatever) message in that category.

The way to set that up is to use the reportEvery parameter for a category. This behaves like limits in the sense that reportEvery can be established for a specific category in a specific destination, for a specific category by default in every destination, or (although this is probably not sensible) for every category.

```
process.MessageLogger = cms.Service("MessageLogger",
                                                destinations   = cms.untracked.vstring('detailedInf
                                                categories      = cms.untracked.vstring('eventNumbe
                                                detailedInfo    = cms.untracked.PSet(
                                                                          eventNumber = cms.untrack

                                                                                                    )

)

process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(20000))
process.myAnalysisModule = cms.EDAnalyzer('MyAnalysis')
process.p = cms.Path(process.doSomeAnalysis)
```

The throttling of messages with a reportEvery for their category never throttles the first such message; that is, if the reportEvery parameter is 100, then messages 1, 101, 201, and so forth will be seen.

The reportEvery parameter only prevents some fraction of messages from being output; it does not replace limits or thresholds so as to allow a message to get through. If a category of message would not go through the log because of a limit or threshold, it will still not be output if it is the 100-th or 1000-th such message as requested by reportEvery.

Adjusting Linebreak Policy

By default, output destinations format the message, breaking the text (at boundaries where there was a new item of information or an operator<<) to avoid lines of greater than 80 columns.

If this is not the desired behavior (for example, if the output file will be processed by an automated parsing tool which prefers each message to be on a single line) then this can be controled in the _cfg.py file:

```
process.MessageLogger = cms.Service("MessageLogger",
                                            destinations   = cms.untracked.vstring( 'detailedInfo'
                                            critical             = cms.untracked.PSet(
                                                                        noLineBreaks = cms.untracked
                                                          ),
                                            detailedInfo   = cms.untracked.PSet(
                                                                       lineLength = cms.untracked.in
                                                     )
)

process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(10))
process.myAnalysisModule = cms.EDAnalyzer('ModuleThatIssuesMessages')
process.p = cms.Path(process.myAnalysisModule)
```

////////////////////

```
process TEST = {

  service = MessageLogger {
    untracked vstring destinations = { "detailedInfo"
                                     , "critical"
                                     , "cerr"

                                     }
    untracked PSet critical     = { untracked string threshold = "ERROR"   }
```

```
    untracked PSet detailedInfo = { untracked string threshold = "INFO"    }
    untracked PSet cerr             = { untracked string threshold = "WARNING" }
  }
  path p = { myAnalysisModule }
  module myAnalysisModule = ModuleThatIssuesMessages { }
  source = EmptySource {untracked int32 maxEvents = 5}
}
```

-- SudhirMalik - 09-Nov-2009

This topic: Sandbox > MessageLoggerSetupExamples
Topic revision: r5 - 2011-09-18 - unknown