# Table of Contents

# CMS MessageLogger: Guide To Issuing Messages

Complete:

## Goal of this page

This page outlines the use of the CMS MessageLogger service.

## Introduction

The CMS MessageLogger Service is meant to allow code in modules, other services, and other framework "scaffolding" to log messages to a unified message logging and statistics facility.

The MessageLogger Service captures and coordinates messages originating in multiple modules (which, under the CMS framework, will in general be running in multiple threads) into a specified set of destinations. The management of these destinations is based on the ZOOM Error Logger package developed at Fermilab.

All users of the MessageLogger service should read the section on Issuing Messages.

The behavior of the MessageLogger can be adjusted via lines in the job's configuration file (_cfg.py). Users wishing to customize the behavior of the MessageLogger should read the section on MessageLogger Parameters.

**Note** that the configuration syntax was revised and simplified in CMSSW_11_2_0_pre10, see also FWCore/MessageService/Readme.md.

- MessageLogger Concepts
- Issuing Messages
- MessageLogger Parameters - Configuring Destinations and their filtering behaviors
- Obtaining Message Statistics
- An OpenIssuesDiscussion wiki is provided.
- Feature List -- and where to find documentation of how to use each feature.
- CMS Guidelines for Messages and Categories
- Frequently Asked Questions
    - Dealing with the MessageLogger from non-cmsRun applications
    - Quieting information from specific verbose modules
    - Producing debug information without impacting production running
    - Preventing the SWGuideMessageLogger from tinkering with the message format

## Issuing Messages

In order to issue messages, the module must include the MessageLogger service header:

```
#include "FWCore/MessageLogger/interface/MessageLogger.h"
```

In addition, it is strongly recommended (for consistency with the way all services are used ) that the `_cfg.py` file contain at least the line

```
process.load("FWCore.MessageService.MessageLogger_cfi")
```

Parameters of the MessageLogger can be explicitly set after the load to modify the behavior (see MessageLogger Parameters); if no parameters are supplied, a sensible CMS default behavior is provided.

If the _cfg.py file does not specify the MessageLogger service, or if a message is issued in code executed before any services are initiated, then the response to issuing a message will be that the content will be sent to `cerr`.

## Functions to issue messages

Having included the necessary MessageLogger header, when code wishes to issue a message, one of these constructors can be used:

```
edm::LogError   ("category") << a << b << ... << z;          edm::LogProblem    ("category") << a
edm::LogWarning  ("category") << a << b << ... << z;         edm::LogPrint    ("category") << a <<
edm::LogInfo    ("category") << a << b << ... << z;          edm::LogVerbatim ("category") << a <
```

(The constructors on the left will create an object that will format the message, adding a header with category and context information. The constructors on right are for non-formatting output, at the corresponding severity levels.)

When issuing messages:

- LogInfo, LogWarning, and LogError represent three levels of "severity" of the message. It is possible (see MessageLogger Parameters ) to configure the job to ignore all LogInfo messages, or all messages of severity less than LogError. See CMS Guidelines for Messages and Categories for advice on which severity is appropriate, and on choosing category names.
- LogVerbatim is identical in all ways to LogInfo, except that absolutely no message formatting is performed, and no context, module, or other information is appended to the message. This is appropriate, for example, if the program has prepared a nicely-formatted table for output. Similarly, LogPrint is identical in all ways to LogWarning, except that absolutely no message formatting is performed, and likewise for LogProblem and LogImportant relative to LogError.
- The category should specify what this message is about. The category will generally appear as the first part of the message, but it also plays two other roles:
    1. It is possible to set limits on how many times some specific type of message will be sent to the outputs of the logger. By "type" we mean any messages with some specific category. See MessageLogger Parameters for details.
    2. When message statistics are provided, the counts of how many times messages of a given type were issued again keyed to category.

  Normally a category name should be up to 20 characters long, without special characters other than underscore.
  (See CMS Guidelines for Messages and Categories.)
- It is unnecessary to explicitly specify the module label or the run/event number; these are automatically provided by the logger.
- An arbitrary number of additional objects << a << b << ... << z can be appended to the message. These can be strings, ints, doubles, or any object that has an **operator<<** to an ostream. (Or the message can be issued with no additional objects at all.)
- There is no need to add spaces at the beginning or end of text items sent to the message, or to add text separators between numberical items. This spacing is taken care of by the logger. However, if any item appended to a message ends in a space, then it is assumed that the user is handling spacing explicitly, and no further automatic spaces are inserted for that message.
- There is no need to affix any sort of **endl**; when the statement ends the message will be dispatched.
- Newline characters can be included in the objects appended to the message. These will be used in formatting the message. But they are generally not necessary: Line breaks are automatically inserted if the next appended object would cause the line to exceed 80 characters.

### Issuing complex or computational costly messages

Beginning with the CMSSW_10_2 release, the messaging classes, e.g. edm::LogError, gained a new function **log** which can be used to issue complex messages or messages which require computational costly algorithms to prepare the message. The **log** function guarantees that the work needed to create the message will ony be done if the messaging level of the system is set high enough for the message to be sent. The **log** function works by taking a C++ lambda as an argument where the lambda takes a messaging object as its argument.

Example

```
edm::LogInfo("yourCategory").log( [&](auto & li) {
   li << a << b;
   li << thisIsATimeConsumingFunction();
   for(auto const& thing: stuff) {
     li << thing << "\n";
   }
});
```

In the example, all the code within the lambda including the **thisIsAtimeConsumingFunction()** and the for loop will only be run if the MessageLogger is configured such that info messages will be kept.

# LogDebug

There is an additional form for issuing a message:

```
   LogDebug    ("category") << a << b << ... << z;          LogTrace    ("category") << a << b
```

This is identical to the others, except:

- LogDebug affixes the **__FILE__** and **__LINE__** number to the message.
- LogDebug messages are considered to be lower in severity than LogInfo messages.
- By default, LogDebug messages will be rapidly discarded with a minimum of overhead. The user must specify in the .cfg file LogDebug messages from various modules are to be enabled; see Controlling LogDebug Behavior: Enabling LogDebug Messages Controlling LogDebug Behavior: Enabling LogDebug Messages for how that is done.
- Because it must get **__FILE__** and **__LINE__** from the spot issuing the message, LogDebug is implemented as a macro rather than a free function.
- Because LogDebug is a macro, it is not prepended with the edm:: namespace designation.

LogTrace is identical in every way to LogDebug, except that absolutely no formatting or information appending will be done. This is appropriate, for example, for trace output coming from a package which assumes it has full control of formatting.

*Note that in order for* LogDebug *or* LogTrace *to have any effect, the symbol* EDM_ML_DEBUG *must be defined.*
This is ordinarily done by placing the option -DEDM_ML_DEBUG into the compiilation flags, as in

scram b -j8 USER_CXXFLAGS="-DEDM_ML_DEBUG"

*Note that* in the LogDebug and LogTrace messages need to be turned on in the configuration as well as described in SWGuideMessageLoggerDebug#enabling. Below is a quick recipe for configuration files using the `FWCore.MessageLogger.MessageLogger_cfi`

```
process.load("FWCore.MessageService.MessageLogger_cfi")
process.MessageLogger.cerr.threshold = "DEBUG"
# enable LogDebug messages only for specific modules
```

```
process.MessageLogger.debugModules = ["moduleLabel", "anotherModuleLabel"]
# or enable LogDebug messages for all modules
process.MessageLogger.debugModules = ["*"]
```

(note that this is the same for both "old" and "new" syntax)

All configuration of what is done when LogTrace messages are issued is taken from the control issued for LogDebug. Similarly, the configuration of what is done when LogVerbatim messages are issued is taken from the control issued for LogInfo, and likewise for LogPrint related to LogWarning and for LogProblem related to LogError .

# Some other wrinkles about issuing messages:

## Compound categories

The category of a message can instead be a "compound" category, with individiual categories separated by a vertical bar (|), as in

```
  LogWarning("ReadoutError|TimeBudgetExceeded") <<"Processed "  <<nitems
                                                <<"items and ran out of time";
```

Compound messges will be reported if any of the individual categories would be reported.

## Subroutine name

The first item streamed to a message can optionally be of the form `"@SUB=methodName"` (note no spaces). This will indicate that the message came from that method; two messages from different "subroutines" are considered, for statistics and limits purposes, to be two different types of messages even if they are in the same category and from the same module.

# Features List

## Issuing messages

- Header file to include in code issuing messages -- Issuing Messages
- Issuing LogError, LogWarning, LogInfo and LogDebug Messages -- Issuing Messages
- Issuing non-formatting LogProblem, LogPrint, LogVerbatim and LogTrace messages -- LogVerbatim, LogPrint, LogProblem -- LogTrace
- Specifying that the MessageLogger service be configured -- Issuing Messages or MessageLogger Parameters
- Guidelines for assigning categories and shoosing which type of message to issue -- CMS Guidelines for Messages and Categories
- Messages can be assigned multiple categories -- Issuing Messages
- The module label will and run/event context will automatically be included in the reported message.
- There is an optional mechanism for specifying the method or subroutine from which the message has been issued -- Issuing Messages
- It is permissible for messages to be issued from code which may execute before the MessageLogger service has been configured -- Issuing Messages
- Spaces are automatically inserted between items streamed to a message, but this can be turned off by ending an item with an explicit space -- Message Formatting: Spaces Between Items

## Configuring destinations

- Specifying a destination for messages, which will write the messages to a file -- SWGuideMessageLogger Parameters: Example .cfg file
- Specifying a destination for messages, which will write the messages to `cerr` or `cout` -- MessageLogger Parameters: Example .cfg file
- Normally, the destination file will be assigned the extension **`.log`**, but a different extension can be specified -- File Destinations: Explicit Extensions
- A different filename (even in a different directory) for a destination file can be specified -- File Destinations: Explicit File Name
- Specifying a destination which will forward messages to `log4cplus` -- MessageLogger Parameters: Routing Messages to log4cplus
- Specifying a threshold for a destination (for example, directing the destination to ignore `LogInfo` and `LogDebug` messages) -- MessageLoggerThreshold
- Specifying limits on how many times a destination will respond to a given message category -- MessageLogger Parameters: Examples of Limits and Timespan Parameters
- Specifying that a given message category should be reported only every n-th time- - MessageLogger Parameters: Example of reportEvery Parameter
- Dictating the line-break policy followed by a destination -- MessageLogger Parameters: Adjusting Linebreak Policy

## Suppression of Messages From Specific Modules

- Suppressing all LogInfo (or LogDebug or LogWarning) messages from specific modules -- MessageLogger Parameters: Suppressing Information from Specific Modules
- Enabling LogDebug messages issued by a specific module -- Controlling LogDebug Behavior: Enabling LogDebug Messages

## Debug messages

- Issuing LogDebug Messages -- Issuing Messages or Controlling LogDebug Behavior
- LogTrace behaves identically to LogDebug, but produces output without headers or formatting -- Issuing Messages
- By default, LogDebug messages will efficiently be ignored -- Controlling LogDebug Behavior: Enabling LogDebug Messages
- Enabling all LogDebug messages -- Controlling LogDebug Behavior: Enabling all LogDebug Messages
- Enabling LogDebug messages issued by a specific module -- Controlling LogDebug Behavior: Enabling LogDebug Messages
- Lowering the threshold for a destination, so it will respond to enabled LogDebug messages -- Controlling LogDebug Behavior
- `#define NDEBUG` will cause compile-time elimination of LogDebug message overhead -- Controlling LogDebug Behavior: Compile-time Elimination of LogDebug Messages
- `#define ML_NDEBUG` to cause compile-time elimination of LogDebug message overhead when `NDEBUG` is not defined -- Controlling LogDebug Behavior: Compile-time Elimination of LogDebug Messages
- `#define ML_DEBUG` will force LogDebug compilation, even if `NDEBUG` is defined -- Controlling LogDebug Behavior: Compile-time Elimination of LogDebug Messages

## Message Statistics

- Counts of various types of messages issued can be obtained --
- Statistics summaries are written automatically at end of job -- Obtaining Message Statistics

- Statistics destinations can be configured to use a specified severity threshold -- Obtaining Message Statistics
- Statistics destinations can write to their own file or share a file with an ordinary output destination -- Obtaining Message Statistics
- User code can trigger the statistics destinations to write out the statistics accumulated thus far -- Obtaining Message Statistics: The edm::LogStatistics() Function
- Each statistics destination can be configured to reset after each time the statistics are written out, or to continue accumulating further statistics -- Obtaining Message Statistics: The edm::LogStatistics() Function
- The program can set statistics for specific categories issued by various modules to be grouped in the summary (rather than each module listed separately) -- Obtaining Message Statistics: The edm::GroupLogStatistics() Function

### Framework Job Reports

- Destinations can be declared which will write an xml version of the logged messages -- Framework Job Reports
- A `fwkJobReport` destination can be configured for limits and thresholds in the same manner as for other output destinations -- Framework Job Reports , and MessageLogger Parameters
- Ordinarily, configured Framework Job Reports will **not** be produced -- Framework Job Reports:Command-line Control
- Production of the Framework Job Reports can be enabled by -e option on the cmsRun command line. -- Framework Job Reports:Command-line Control
- A name can be assigned for a Framework Job Report by a -j option on the cmsRun command line. -- Framework Job Reports:Command-line Control

## Frequently Asked Questions

The MessageLogger Frequently Asked Questions page addresses the following topics:

- Dealing with the MessageLogger from non-cmsRun applications
- Quieting information from specific verbose modules
- Producing debug information without impacting production running
- Preventing the SWGuideMessageLogger from tinkering with the message format
- What if two destinations are configured to write to the same file?

### MessageLogger Behavior

Descriptions of various aspects of the MessageLoggerBehavior, including:

- What happens if the configuration specifies two destinations using the same file name?

## CMSSW Guidelines for Messages and Categories

Messages sent to the logger are identified by two basic dimensions: a severity and a category. The severity is embedded in the names of four basic functions, one for each level. Each also has a partner function which outputs the message contents with absolutely no formatting. Here are some guidelines for the use of the functions. The use of the category will be covered afterwards.

### Guidlines for which severity to use

The non-formatting function is given in parentheses.

1. LogDebug (or LogTrace): These are designed as macros, to make inactive LogDebug statements as inexpensive as possible. There will be no message formatting overhead, and the compiler may be able to remove the statement completely at compile-time, if the proper "#defines" are in place to inform the compiler that this is production mode. Also, the configuration can disable LogDebug on a per-module basis.
    - Use this anywhere you want to report state information about your algorithm that is useful for figuring out the behavior of the algorithm or reporting positions that the program has reached. Examples:
        - LogDebug("SegmentFinder") << "Matching segment " << i << " in tracking volume " << vol << "\n";
        - LogDebug("SegmentFinder") << "Starting to find candidates from " << numhits << " hits\n";
    - Use this for messages that would not naturally be included when running on a farm in production.

2. edm::LogInfo (or edm::LogVerbatim):
    - Use this for low frequency or course-grained status reporting, such as
        - edm::LogInfo("ElectronFinder") << "Saw " << x << " electrons in " << y << " events\n";
    - Messages of this severity will not routinely be included in reports while running on a grid in production.

3. edm:LogWarning (or edm::LogPrint):
    - Use this when an algorithm encounters a perverse situation which does not cause an exception to be thrown, but which person running the program should be made aware of.
        - edm::LogWarning("Convergence") << "Could not satisfy convergence criteria in " << iters << " iterations, continuing\n";
        - edm::LogWarning("TooManyHits") << "Found too many hits: " << x << ", Pattern recognition skipped in " << reg << "region\n";
    - Caretakers of production jobs run on the grid will see LogWarning messages. Please do not inundate them with messages that express situations which are not intended to eventually be corrected, or with other information which they should not be concerned about.

4. edm::LogError (or edm::LogProblem): This is used by the framework to report errors that result from exception throws.
    - Do not use edm::LogError this if you throw an exception - the system will make this call for you when the exception is caught. Put all the information that you want reported into the exception. For more information about error handling see: https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideEdmExceptionUse.
    - If you do have a situation where you encounter an error, but cannot throw an exception (i.e. can only return a bad return code), then you can report this via edm::LogError.

## Guidelines for choosing category names:

One use of the category is for filtering messages or observing messages types across all algorithms. The category concept matches that of the exception processing, where different actions can be taken based on exception category. When exceptions are caught by the framework, the category of the exception is used as the LogError category. Choosing general names for conditions will facilitate filtering and output log browsing.

For errors and warnings, category name examples include:

- DataNotFound

- TooLittleData
- TooMuchData
- ReadoutError
- TimeBudgetExceeded

For Info and Debug, category names can include the function that is being performed.

By convention, category names should be 20 characters or shorter. Category names of up to 200 characters will work fine, but only the first 20 characters will be displayed in message statistics tables. Category names longer than 200 characters should not be used.

Category names can contain any combination of letters, numbers and underscores. Due to the need to be parsed as PSet names in the configuration file, and to be used and possibly combined when issuing messages, category names:

- Should not contain blank spaces or tabs.
- Should not contain other special characters. In particular
  / , \ , . , [ , " , { , } , = , - , |
  are known to be problematic if used in category names.
- The category name is not allowed to contain a slash / or a backslash \ . (Yes, we know this is a redundant warning.)

A log message can be a member of more than one category using the syntax:

- LogWarning("TooLittleData|Tracking") << "...";

Note the "|" character that is used as a separator for the two category names.

You may also consider adding the class name from which the message is issued as a category, however if the class is an EDProducer or other type of module this will be redundant information. As with other framework services, the message logger sets a context for itself between module invocations so that it can record an appropriate "context" for each issued message. Currently the context contains the module type and label as well as the current run/event counter. However if the message originates in a low level algorithm class, then the class name::member function information would be a useful category in helping to track down the origin of the message.

## Composing the Message

The content of the message depends, of course, on the information you need to convey. However, because these messages will appear in a stream together with messages issued by others, the following guidelines are provided:

- Information which repeats information present in the message header (including date/time and module label) is discouraged.
- Please prefer to use the formatted logging functions rather than to create your own format which includes the information which would be in the standard header.
- "ASCII art", included in a message to help it "stand out" from other messages, is **strongly discouraged**. This includes multiple line feeds, dashes across the page, rows of plus signs and so forth. Such techniques inevitably lead to escalating "ad-wars" and in the long run nobody is helped. Choosing category names which will assist in automated searching for particular sorts of messages is a good alternative.
- One last note. Beware of copy-paste errors, this can lead to mis-leading information which is almost worse then no information at all.

# MessageLogger Glossary

The wiki page SWGuideMessageLoggerGlossary contains explanations of the key concepts and jargon that the MessageLogger examples and documentation use.

# Examples of Configuration Files

## Tutorial Examples

SWGuideSimpleMessageLoggerUsage

SWGuideEnablingLogDebugForModules

SWGuideTypicalCMSusage

## Reference Examples

The following examples are intended to cover all or most of the available options. Since some of these options are esoteric, these examples are not as easy to follow as the "tutorial" examples.

# Message Logger SandBox

Area in free format for items that the groups wish to add: meetings, to do lists, links etc

## Review Status

The Message Logger twikis created are in the twiki below:

SWGuideMessageLoggerListOfTwikis

This is still an ongoing work. Please help improve it or in case have suggestions.

| Reviewer/Editor and Date (copy from screen) | Comments |
|---|---|
| Main.fischler - 25 Jul 2006 | page author, latest modification (Mark Fischler) |
| Main.fischler - 31 Jan 2008 | Guidelines for Messages and Categories brought up to date (Mark Fischler) |
| Main.fischler - 9 Apr 2008 | Link to GroupLogStatistics feature (Mark Fischler) |
| NitishDhingra - 22-Apr-2012 | Complete Review |

Responsible: SudhirMalik
Last reviewed by: SudhirMalik

---

This topic: CMSPublic > SWGuideMessageLogger
Topic revision: r61 - 2020-12-18 - MattiKortelainen