WCT Logging

Brett Viren

June 20, 2024

Contents

1 Introduction **1**

2 Usage Guidelines **1**

3 WCT specifics **2**
3.1 Building 2
3.2 Running 2
3.3 Executing 3
3.4 Checking 3
3.5 Developing 4

1 Introduction

In WCT code we must "never" use `std::cout` or `std::cerr` and certainly never for "logging". Instead, emit log messages using WCT's support for the spdlog. That package is directly exposed so see its good documentation for details. Here we give a brief usage guidelines and some WCT specifics.

2 Usage Guidelines

- Do not use `std::cout` nor `std::cerr`.
- Use `spdlog::<level>()` function the macros: `SPDLOG_LOGGER_<LEVEL>()` or `SPDLOG_<LEVEL>()` macros.
- Never place any code that is required in general inside these macros.
- Make messages informative of program state and avoid purely static strings.
- Consider log analysis when defining a message format, (eg use consistent `var=<val>` pattern).
- Consider creating a github issue for any message with a level above "info" so users encountering the message may find guidance and understanding.
- Follow the level guidelines listed below.

An important and communal responsibility is to select a proper log "level" for any given message content. The most important considerations in selecting a level are the frequency at which the message would be emitted and the expected response from a human seeing the message. The following table provides guidelines on these two considerations and states the default compilation for use of the CPP macros.

level	verbosity	intention	default
trace	very	Produced and consumed by developers to understand some code aspect.	off
debug	somewhat	Produced by users with a special build to help developers debug an issue.	off
info	low	Notify user of non-problematic progress and state of the program.	on
warn	rare	Notify user of assumption made to correct incorrect user input (data/config).	on
error	rare	Notify user that a context raised an error that is likely corrected by caller.	on
critical	rare	Notify user that a context raised and error that is likely not correctable.	on

3 WCT specifics

3.1 Building

The default SPDLOG "active level" is set at build configuration time to be "info" level. This will deactivate any code inside `SPDLOG_DEBUG()` and `SPDLOG_TRACE()` macros. To change this level, for example to "trace":

```
./wcb configure --with-spdlog-active-level=trace [...]
./wcb
```

3.2 Running

The `wire-cell` command line program allows changing the SPDLOG level on a per logger basis.

```
wire-cell --help
...
-l [ --logsink ] arg  set log sink as <filename> or 'stdout' or 'stderr', a
                      log level for the sink may be given by appending
                      ':<level>'
-L [ --loglevel ] arg set lowest log level for a log in form 'name:level' or
                      just give 'level' value for all (level one of:
                      critical,error,warn,info,debug,trace)
...
```

With no logsink, WCT logging-compliant code is totally "silent". A minimal setting is:

```
wire-cell -l stderr -L debug [...]
```

This sets the default log level to be DEBUG with log output sent to stderr. Many WCT data-flow processing nodes implement `Aux::Logger` which supports setting a per-node instance log level. To set this, one must use the triplet of the log group and the component type and instance names. These can be deduced from existing log lines. For example,

```
[15:59:11.745] D [  root  ] <UbooneBlobSource:live-uvw> loading live blobs of views bits: 7
```

This logger has a full name:

```
root/UbooneBlobSource/live-uvw
```

To set the level for the logger of this one instance one may give, for example:

```
wire-cell -L root/UbooneBlobSource/live-uvw:debug -l stderr [...]
```

This will only show that one instance at DEBUG and all logging will be at INFO.

However, there is a pattern matching mechanism to set logger levels. This job happens to have four instances of the component with type name `UbooneBlobSource`. All four can have their log levels set by giving a common prefix:

```
wire-cell -L root/UbooneBlobSource:debug -l stderr [...]
```

Likewise, all loggers using the logger group `root` (as in the WCT sub-package using ROOT, not the root of some tree), one can give:

```
wire-cell -L root:debug -l stderr [...]
```

3.3 Executing

When WCT is executed as a toolkit the calling code may use SPDLOG directly to control log levels. In addition the user may set the environment variable, eg:

```
SPDLOG_LEVEL=debug
```

The emission of log messages from SPDLOG macros is still subject to the build time configuration.

3.4 Checking

As the logging control is rather flexible, it can be tricky to understand. A simple check program is provided that excerpts the logging functionality of the WCT main. With it, one can quickly try different command line options. Here is an example of its use:

```
This allows checking WCT logging as it would be used via wire-cell.

Print help

$ check-logging -h

Default log stream, level and sink:

$ check-logging

Limit the default stream

$ check-logging -L warn

Setting any log sink (-l/--logsink) will cause the default log stream to be named "wct".
```

```
Set a default log sink

$ check-logging -l stderr

Set a sinks with levels

$ check-logging -l stderr:trace -l info.log:info

Set message level

$ check-logging -l stderr:trace -l info.log:info -l trace

Set message level on per stream basis

$ check-logging -l stderr:trace -l info.log:info -l trace -L wct:critical -L test:warn
```

3.5 Developing

Developers may create a specific SPDLOG logger object or may use the "main" object. Log object construction may be simplified when constructing a WCT component that inherits from `WireCell::Aux::Logger`. See `util/test/doctest_logging.cxx` and existing use in WCT source for some examples.

The emission of log messages from SPDLOG macros is still subject to the build time configuration.