

Logging Package Status and Plans

Ron Rechenmacher

CCM

3 February 2021

Introduction¹

The CCM is responsible for providing a complete solution for capturing, distributing and archiving logs. Logs in this context are debug statements, information, warning and error messages². While debug statements can be unstructured³, all higher-level⁴ types of messages are structured, carry a well-defined set of information fields and extend from a base class that can be thrown and caught as an exception.

1 Statement from an email from Alex Tapper, in concert with Giovanna and Alessandro.

2 There are 6 severities or streams: fatal, error, warning, info, log, and debug.

3 All messages have at least 2 destinations: memory/fast and stdout/err. (Another destination is “network,” i.e. MTS). For debug statements, a structured object can be used, or a streamer-type message which, when sent to destination(s) other than memory/fast, will be used to create a Message (structured) object. For the memory/fast destination for debug statements, the streamer-type message and/or the message from the structured object is stored in the memory buffer and later retrieved and formatted with other information specified by an environment variable and sent to standard out (or redirected elsewhere).

4 Higher-level types are: fatal, error, warning, and info.

Specifications¹

The logging system shall provide the following functionality:

1. Provide a base class for all structured messages and tools to generate the specific messages in a straightforward way.
2. Provide APIs for reporting debug text and messages.
3. Provide an API to subscribe to messages using a set of criteria.
4. Provide a set of stream implementations for the dispatching of debug text and structured messages to stderr and stdout, as well as network-based streams.
5. Provide a mechanism to receive structured messages over the network, for message subscription.
6. Provide a message archiving system and a UI to search for and analyze messages, live and post-mortem.

¹ Specifications are from an email from Alex Tapper, in concert with Giovanna and Alessandro.

Interfaces¹

The interfaces between the DAQ software and the logging system are defined as follows:

1. Debugging:
 - a. The logging system provides an API to log unstructured messages for debugging purposes.
 - b. The logging system provides tools to dynamically turn on/off or tune the level and type of debug statements which are output.
2. Structured information:
 - a. The logging system provides the base class from which all the messages need to extend as well as helper macros to define specific types.
Note: DUNE chose to use the ERS library developed within ATLAS and the base class is the `ers::Issue`.
 - b. The logging system provides an API in order to be able to output messages according to their severity.
 - c. The logging system provides the tools to configure the implementation of the streams used for each severity.
 - d. The logging system provides an API to subscribe to messages using a set of criteria (e.g. subscribe by message type, application name, hostname, severity, ...).
 - e. The DAQ software consistently uses the logging system and ensures that the verbosity of the messages is appropriate.

¹ Interfaces are from an email from Alex Tapper, in concert with Giovanna and Alessandro.

Rev. 1.0.0 of logging package – basic overview

- ERS has 6 logging methods and 3 logging macros (in addition to Issue declaration macros):
 - `ers::fatal(const Issue &)`
 - `ers::error(const Issue &)`
 - `ers::warning(const Issue &)`
 - `ers::info(const Issue &)`
 - `ers::log(const Issue &)`
 - `ers::debug(const Issue &, int)`
 - `ERS_INFO(message)`
 - `ERS_LOG(message)`
 - `ERS_DEBUG(level, message)`
- The first 4 (blue) are to be used, and the last 5 (red) are not (directly); the macros are #undef'd

Rev. 1.0.0 of logging package – basic overview - continued

- TRACE provides 8 TLOG* macros:
 - TLOG_ERROR, TLOG_WARNING, TLOG_INFO, TLOG_TRACE, TLOG_DEBUG, TLOG_DBG, TLOG, and TLOG_ARB
- 2 survive and 6 are #undef'd.

Rev. 1.0.0 of logging package - summary

1. CMakeLists.txt changes:

- Add: find_package(logging REQUIRED)
- Remove: find_package(ers REQUIRED) and find_package(TRACE <version> REQUIRED)
- To DEPENDENCIES, add: logging:logging

2. #include "logging/Logging.hpp"

3. Use ERS DECLARE_* macros, as usual

- I suspect there may be an "issues package"

4. Use:

```
ers::fatal( ers::Issue & )  
ers::error( ers:: Issue & )  
ers::warning( ers:: Issue & )  
ers::info( ers:: Issue & )  
TLOG() << message _or_Issue;  
TLOG_DEBUG(lvl) << message_or_Issue
```

note: existing uses of TLOG() may need to be adjusted.

5. Remove any uses of ERS_LOG, ERS_INFO, and ERS_DEBUG

Addressing The Specifications

include “logging/Logging.hpp”

Nests ers/ers.h and TRACE/trace.h

1. Provide a base class for all structured messages and tools to generate the specific messages in a straightforward way.

Base class provided by ERS. Macros provided to DECLARE issues with attributes (arguments). C-style casts seem to be required.

2. Provide APIs for reporting debug text and messages.

```
TLOG_DEBUG(lvl) << text << arg;  
TLOG_DEBUG(lvl) << ers::Issue(); // define operator<< for other objects.
```

3. Provide an API to subscribe to messages using a set of criteria.

For all ERS levels/streams, ERS environment variables (e.g. TDAQ_ERS_ERROR) support filter specifications. These variables need to be defined before or very early during program execution. TRACE provides environment variable and command line utilities to specify “name”:”level mask” pairs to select which messages go to the memory buffer and for the “log” and “debug” destinations which messages go to ERS.

Addressing The Specifications - continued

4. Provide a set of stream implementations for the dispatching of debug text and structured messages to stderr and stdout, as well as network-based streams.

`ers::log` and `ers::debug` are used by a “trace user method” and other operator<< methods.

5. Provide a mechanism to receive structured messages over the network, for message subscription.

This is currently not addressed by the logging package (v1.0.0).

6. Provide a message archiving system and a UI to search for and analyze messages, live and post-mortem.

Archiving is currently not addressed by the logging package (v1.0.0).

“tshow -F” from a separate terminal window, with standard unix filtering, can be used to show message from the memory buffer as they are put in by the application.

Addressing the Interfaces

1. Debugging:

- a. The logging system provides an API to log unstructured messages for debugging purposes.

Debug messages to other than memory will be “structured” by use of `ers:debug w/ ers::Message` or other `ers::Issue`. Message from the memory buffer retrieved via `tshow` will be formatted according to the format specification in the env.var. `TRACE_SHOW`.

- b. The logging system provides tools to dynamically turn on/off or tune the level and type of debug statements which are output.

The command line function `tlvs` is used to show the current memory buffer configuration. The command line functions `tonM{,g,G}`, and `toffM{,g,G}` (6 functions) are used to turn on/off various levels for a specific name, wildcard name or all names (current or future).

Addressing the Interfaces - continued

2. Structured information:

- a. The logging system provides the base class from which all the messages need to extend as well as helper macros to define specific types.
Note: DUNE chose to use the ERS library developed within ATLAS and the base class is the `ers::Issue`.

ERS methods and macros, as described above are used.

- b. The logging system provides an API in order to be able to output messages according to their severity.

4 ERS methods (with “erstrace” destination configured) are used directly and 2 TRACE macros with a TRACE_LOG_FUNCTION configured which calls the remaining 2 ERS methods (if enabled by dynamic configuration) are used. All 6 severities are covered.

- c. The logging system provides the tools to configure the implementation of the streams used for each severity.

The system relies on environment variables for configuration. The logging package provides `Logging().setup()` to provide reasonable defaults for the environment variables in case they are not provided by the user or CCM. Note: if `Logging().setup()` is not called, the high level ers methods will not send to the memory buffer, but the TLOG macros will still send, if configured to the appropriate ers streams.

- d. The logging system provides an API to subscribe to messages using a set of criteria (e.g. subscribe by message type, application name, hostname, severity, ...).

Standard ERS filter specifications in the standard TDAQ_ERS environment variables are used.

Addressing the Interfaces - continued

- e. The DAQ software consistently uses the logging system and ensures that the verbosity of the messages is appropriate.

The ERS stream environment variables support a throttle specification. The logging package supplies the TLOG() macro which, currently, can not be dynamically disabled, and the TLOG_DEBUG(lvl) macro which can be dynamically disabled (in addition to throttled). Should TLOG() to ERS (slow path) be disable-able?
No limits or direct guidance on which messages should be INFO, LOG, or DEBUG.

To-do's

1. Pull requests for appfwk, trigemu, readout packages (and listrev).
2. `TLOG*() << message ==> ers::log(ers::Message(msg))` set time to match fast/membuf time.
3. Check for “caused by” in streamer methods. (Low priority). Done in erstrace ERS destination.
4. Logical OR of existing `TRACE_LVL` env.var. in `Logging().setup()`.
5. Real tests – ers output, memory messages, and levels.
6. Investigate 2nd `ers::debug` – always at level 0. Ref. ers debug code – severity is set and set back???
7. README (and other documentation)
8. (Work w/ John when he's back to) clean up any loose ends? cmake, lint, etc.
9. Address issue with year in date (`TRACE_TIME_FMT`)

TRACE message control

- export TRACE_FILE=/tmp/trace_buffer_\$USER # in setup (append \$PARTITION)
OR export TRACE_MSGMAX=0 # if interactive (less typing)
- tlvs shell function shows levels, e.g. minidaq:

```
mode:                                     M=1                                     S=1
TID                                     NAME                                     maskM                                     maskS                                     maskT
-----
20                                     DAQSink 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
101 TriggerDecisionForwarder 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
268                                     TRACE 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
293 TriggerInhibitAgent 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
643 RequestGenerator 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
771 HDF5DataStore 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
842 DataWriter 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
890 FragmentReceiver 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
954 DAQSource 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
1021 _TRACE_ 0x0000000000000001ff 0x0000000000000000ff 0x0000000000000000
```

TRACE message control - continued

- Env. Vars: TRACE_NAMLVLSET, TRACE_LVLLS, TRACE_LVLM
- Shell (cmdline) functions: tlvlsSave, tlvlsRestore
- Shell (cmdline) functions: t{on,off}{M,S}{,g,G}
 - -n name, -N regex
 - Levels
 - Raw Bit Levels: 0-63
 - Debug Levels 0-55 (raw_level – 8)

```
/home/ron/work/DUNEPrj/AD202
1-01-27_v2.2.0/MyTopDir
(dbt-pyvenv) ron@mu2edaq13
:^) tcntl lvlstrs | head
0     FATAL
1     ALERT
2     CRIT
3     ERROR
4     WARNING
5     NOTICE
6     INFO
7     LOG
8     DEBUG
9     DEBUG_1
--2021-02-03_00:24:02_CST--
```

Debugging options

1. Traditional log file
 - export TDAQ_ERS_DEBUG_LEVEL=63
 - export TRACE_LVL=-1
2. tshow
3. tshow -F

Log File vs Mem File (tshow)

```
2021-Feb-03 00:51:12,327 DEBUG_0 [main(...) at /home/ron/work/DUNEPrj/AD2021-01-27_v2.2.0/MyTopDir/sourcecode/logging/test/apps/performance.cxx:96] hello from DEBUG_6 loop 0
```

```
2021-Feb-03 00:51:12,327 DEBUG_6 [main(...) at /home/ron/work/DUNEPrj/AD2021-01-27_v2.2.0/MyTopDir/sourcecode/logging/test/apps/performance.cxx:96] hello from DEBUG_6 loop 1
```

VS

```
02-03 00:51:12.327859          0 246818 246818 25          performance:96 D06 main: hello from DEBUG_6 loop 1
02-03 00:51:12.327559        300 246818 246818 25          performance:96 D06 main: hello from DEBUG_6 loop 0
```

delta_us pid tid