Logging

Table of contents

Overview

The Holoscan SDK uses the Logger module to convey messages to the user. These messages are categorized into different severity levels (see below) to inform users of the severity of a message and as a way to control the number and verbosity of messages that are printed to the terminal. There are two settings which can be used for this purpose:

- Logger level
- Logger format

Logger Level

Messages that are logged using the Logger module have a severity level, e.g., messages can be categorized as INFO, WARN, ERROR, etc.

The default logging level for an application is to print out messages with severity INFO or above, i.e., messages that are categorized as INFO, WARN, ERROR, and CRITICAL. You can modify this default by calling `set_log_level()` (`C++`/`Python`) in the application code to override the SDK default logging level and give it one of the following log levels.

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- CRITICAL
- OFF

Ingested Tab Module

Additionally, at runtime, the user can set the `HOLOSCAN_LOG_LEVEL` environment variable to one of the values listed above. This provides users with the flexibility to enable printing of diagnostic information for debugging purposes when an issue occurs.

```
export HOLOSCAN_LOG_LEVEL=TRACE
```

Note

Under the hood, Holoscan SDK uses GXF to execute the computation graph. By default, this GXF layer uses the same logging level as Holoscan SDK. If it is desired to override the logging level of this executor independently of the Holoscan SDK logging level, environment variable `HOLOSCAN_EXECUTOR_LOG_LEVEL` can be used. It supports the same levels as `HOLOSCAN_LOG_LEVEL`.

Note

For distributed applications, it can sometimes be useful to also enable additional logging for the UCX library used to transmit data between fragments. This can be done by setting the UCX environment variable `UCX_LOG_LEVEL` to one of: fatal, error, warn, info, debug, trace, req, data, async, func, poll. These have the behavior as described here: UCX log levels.

Logger Format

When a message is printed out, the default message format shows the message severity level, filename:linenumber, and the message to be printed.

For example:

```
[info] [ping_multi_port.cpp:114] Rx message value1: 51 [info]
[ping_multi_port.cpp:115] Rx message value2: 54
```

You can modify this default by calling `set_log_pattern()` (`C++` / `Python`) in the application code to override the SDK default logging format.

The pattern string can be one of the following pre-defined values

- **SHORT** : prints message severity level, and message
- **DEFAULT** : prints message severity level, filename:linenumber, and message
- **LONG** : prints timestamp, application, message severity level, filename:linenumber, and message
- **FULL** : prints timestamp, thread id, application, message severity level, filename:linenumber, and message

Ingested Tab Module

With this logger format, the above application would display messages with the following format:

```
[info] Rx message value1: 51 [info] Rx message value2: 54
```

Alternatively, the pattern string can be a custom pattern to customize the logger format. Using this string pattern

```
"[%Y-%m-%d %H:%M:%S.%e] [%n] [%^%l%$] [%s:%#] %v";
```

would display messages with the following format:

```
[2023-06-27 14:22:36.073] [holoscan] [info] [ping_multi_port.cpp:114] Rx message value1: 51 [2023-06-27 14:22:36.073] [holoscan] [info] [ping_multi_port.cpp:115] Rx message value2: 54
```

For more details on custom formatting and details of each flag, please see the spdlog wiki page.

Additionally, at runtime, the user can also set the `HOLOSCAN_LOG_FORMAT` environment variable to modify the logger format. The accepted string pattern is the same as the string pattern for the `set_log_pattern()` api mentioned above.

Precedence of Logger Level and Logger Format

The `HOLOSCAN_LOG_LEVEL` environment variable takes precedence and overrides the application settings, such as `Logger::set_log_level()` (`C++` / `Python`).

When `HOLOSCAN_LOG_LEVEL` is set, it determines the logging level. If this environment variable is unset, the application settings are used if they are available. Otherwise, the SDK's default logging level of INFO is applied.

Similarly, the `HOLOSCAN_LOG_FORMAT` environment variable takes precedence and overrides the application settings, such as `Logger::set_log_pattern()` (`C++` / `Python`).

When `HOLOSCAN_LOG_FORMAT` is set, it determines the logging format. If this environment variable is unset, the application settings are used if they are available. Otherwise, the SDK's default logging format depending on the current log level (`FULL` format for `DEBUG` and `TRACE` log levels. `DEFAULT` format for other log levels) is applied.

Calling the Logger in Your Application

The **C++ API** uses the HOLOSCAN_LOG_XXX() macros to log messages in the application. These macros use the fmtlib format string syntax for their format strings.

Note

Holoscan automatically checks `HOLOSCAN_LOG_LEVEL` environment variable and sets the log level when the Application class instance is created. However, those log level settings are for Holoscan core or C++ operator (C++)'s logging message (such as `HOLOSCAN_LOG_INFO` macro), not for Python's logging. Users of the **Python API** should use the built-in `<a href="https://docs.python.org/3/howto/logging.html">logging</a>` module to log messages. The user needs to configure the logger before use (`logging.basicConfig(level=logging.INFO)`):

```
>>> import logging >>> logger = logging.getLogger("main") >>>
logger.info('hello') >>> logging.basicConfig(level=logging.INFO)
>>> logger.info('hello') INFO:main:hello
```