Logging to DDS

Logging.hpp

Logging provides an interface to log important messages and error messages. These messages are printed using cerr, stored in a file locally and sent to the LCC for log collection - the user does not need to think about how the log messages are stored and can simply use the .write function. A Logging Singleton is used so that the object must not be passed around between all classes that implement the feature.

Warning - use `set_id`: The ID of the Logging Singleton must be set at the beginning of the program whenever the cpm lib is used. If no ID would be set, the sender of the logging messages could not be identified by the LCC and the feature would become meaningless. As the cpm lib relies on the Logger, the ID must thus be set as early as possible, after the creation of the Singleton and before any other part of the cpm interface is used.

The write function can be used similarly to the C-style `fprintf` function. This is much easier than any cout-style function, where it is unclear when to flush and send / store / print the data, even if temporary objects are used (the user might store the object).

To uniquely identify the log files, the time of initialization of a log is used as a name for the log file. ID, timestamp and content are stored in this file and sent to the LCC. The cerr output does not include the ID, as it is used to identify the program only. The log file is realized as a csv file, which is easy to write and supported by most spreadsheet programs.

Log Level (WIP)

Logging does not always need to be verbose, and might not be required during demonstrations for performance reasons.

The log level is set for all participants using the LCC's UI, which gives options to set the log level correspondingly (see here for more information about how to set the log level in the UI of the LCC).

The name of the topic is ***logLevel***. It is an unsigned short and has four different values:

Log level	Description
0	Do not log anything (e.g. for performance reasons)
1 (**default**)	Only log critical failures (only those that cause immediate critical system failures)
2	Also log less relevant error messages (e.g. including when a condition or specification could not be met)
3	Verbose (log even simple things, like whether a message was received)

You do not need to read the log level. The cpm library does that internally in combination with the Logging feature.

Example

```
#include "cpm/Logging.hpp"
//Always required if the cpm lib is used
cpm::Logging::Instance().set_id("my_id");
//Log a simple message (you must not use unformated strings, or else you will get a warning)
cpm::Logging::Instance().write(3, "%s", "This is a log message");
//Usage is similar to fprintf
cpm::Logging::Instance().write(2, "The %s is %i, although that should not be the case", "answer", 42);
```