DDS Reader and Writer

A Short Notice on (A)Synchronous Readers

You can process data that you receive in two ways:

- **Asynchronously**: The data is read immediately when it gets received (similar to interrupts, but in another thread instead of interrupting the current control flow). You can get access to this data and e.g. store it in a thread-safe manner (e.g. using mutexes) into your own data structures, which can be processed by your own threads or your main thread.
- **Synchronously**: The data is stored in a buffer. When you call a function (like *take* or *read*), the received data (all data since the last function call / only the most recent one / ... depending on your QoS settings) can be obtained. This can be done e.g. periodically in your main thread.

Reader.hpp

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/cpm_lib/include/cpm/Reader.hpp

If data is read synchronously, the user might be interested in the most recent data only. For most of our own data types, we use a self-created header that contains timing information:

- create_stamp: Time (in nanoseconds) of data creation. The higher, the more recent is the data.
- valid_after_stamp: Time (in ns) when the data can be used. Data is not supposed to be used before this point in time is reached.

The Reader class can be used to create a data reader from which **only the most recent valid data can be requested**, using `get_sample`. Data is valid if its `valid_after_stamp` is lower than the current timestamp `t_now` that is passed to the function with which you can obtain the most recent data (the .hpp should contain all required information). You can use it for data types like trajectories or vehicle data.

> The Reader stores all received data in a ring buffer. The order in which data arrives is not regarded when data gets overwritten. Thus, some slightly newer samples might get lost if `get_sample` is called infrequently - the Reader's buffer is not flushed regularly in between - and if data does not arrive in the correct order.

ReaderAbstract.hpp

https://git.rwth-aachen.de/CPM/Project/Lab/software/-/blob/master/cpm_lib/include/cpm/ReaderAbstract.hpp

If you only need a **basic DDS reader**, then you should preferably use this class. It only **offers information regarding matched publications and a take() function**. It does not require the user to know or understand DDS implementation details - please only used more advanced concepts if you really need to. **Setting QoS parameters** is available through the constructor.

AsyncReader.hpp

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/cpm_lib/include/cpm/AsyncReader.hpp

Data received by a **DDS Reader** can either be read synchronously or **asynchronous**ly. The latter requires the usage of e.g. a StatusCondition, an AsyncWaitSet and a callback function. To save the users the time to look up which StatusCondition is (mostly) required and to write the > 10 lines of code required for every single reader that relies on a callback function, AsyncReader is provided as a sort of **wrapper** that performs the set-up.

The class **must be passed a callback function**. Please be aware that your callback function will be called within the **AsyncReader's own thread**, so you need to use e.g. a mutex to protect your data, and you might have to use lambda captures or std::bind to be able to access e.g. your class members from within the function. It also requires a domain participant (or none, the default then is ParticipantSingleton) and a topic (or a filtered topic) or topic string (for the topic name). Reliable communication can be turned on / off with a boolean parameter.

MultiVehicleReader.hpp

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/cpm_lib/include/cpm/MultiVehicleReader.hpp

Like Reader.hpp, but for multiple vehicles at once: Gather all received messages of type *T* by all vehicles that publish on the (filtered) topic *my_topic*, then return all newest samples when `get_samples` is used. For more information on the implementation, refer to Reader.hpp. An example for a MultiVehicleReader is provided at the central routing example.

> **Important**
>
> The reader in MultiVehicleReader uses a history of up to 2000 samples (it only buffers the most recent 2000 samples). This is due to performance reasons and because take() might not return all received messages immediately (it sometimes needs to be called more than once internally). The value of 2000 samples might not be enough depending on your scenario. In that case, i.e. if messages of some vehicles are sometimes missing because the buffer is not large enough, you might have to increase this value.

Writer.hpp

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/cpm_lib/include/cpm/Writer.hpp

This is a basic **DDS Writer**, which only **offers information regarding matched subscriptions and a write() function**. It does not require the user to know or understand DDS implementation details and is thus recommended whenever a basic writer is required in your application. **Setting QoS parameters** is available through the constructor. You can find examples for a writer-implementation in every provided example as it is essential to transmit the trajectory to the vehicles.