RTI DDS and Matlab

How to use RTI DDS in Matlab

Sample files

The GitLab repository that includes the Middleware Structure also includes some simple Matlab scripts which demonstrate how to use RTI DDS in Matlab. If you have the feeling that some information are missing in this Markdown file, you might find more implementation details in these sample scripts.

There also exists a Matlab init script that you should use to initialize DDS instances and QoS (aside from additional participants that you require). If you use Matlab and want to deploy your software remotely, you should also have a look at the explanation of what to consider in that case here.

Basic Matlab Knowledge

Parameters in Matlab Scripts

```
function []=some_function(some_parameter)
    disp(some_parameter)
    %% Write your script in here
    ...
end
```

It is important that your script and your function have the same name!

Calling a Matlab Script from the Command Line

```
.../Matlab/bin/matlab -nodisplay -nosplash -logfile optional-log.log -nodesktop -sd "path-to-your-matlab-script" -
r "some_function(some_params)"
```

The *-logfile* parameter is optional.

Setting Environment Variables

```
setenv("NDDS_QOS_PROFILES", "file:///home/controller/Documents/QOS_LOCAL_COMMUNICATION.xml;file:///home/controller
/Documents/QOS_READY_TRIGGER.xml");
```

In this example, the environment variable *NDDSQOSPROFILES* is set in order to import two QoS RTI DDS files in Matlab. Other environment variables can be set using *setenv* as well.

Setting Include Paths

```
addpath('./IDL')
```

In this case, additional IDL files for the matlab script are in the subfolder IDL.

Using C Scripts

To use a C function in Matlab, *mex* needs to be used to create a *.mexa64* file from a C script. The script needs to import the mex header file and must contain a *mexFunction* as shown in the following example:

```
#include "mex.h"
#include <time.h>
#include <stdint.h>

long long getTimestampNow() {
    ...
}

//Mex function - entry for matlab
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    //Variable set-up
    plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT64_CLASS, mxREAL);
    uint64_t* data = (uint64_t*) mxGetData(plhs[0]);
    //Data returned to matlab
    data[0] = getTimestampNow();
}
```

In this case, the name of the script is *getTimestampNow.c*. If the file created by mex is on the same path as the matlab script that you are using, you can call the function without importing it:

```
stamp = getTimestampNow();
```

The parameters of *mexFunction* can also be used to pass data from matlab to the C function. Using C functions is useful e.g. if system data need to be obtained. In this case, the current timestamp of the system from *clock_gettime* is passed to Matlab for further use. It is more exact than the Matlab equivalent.

Using RTI DDS in Matlab

Using Message Types Defined in IDL Files

Simply type `DDS.import('YourType.idl','matlab','f')` to import your IDL type. This type can now be used either to create message objects `matlab msg = YourType; msg.some_field = some_value;` or to create RTI DDS participants to send and receive messages. For more information on IDL files, please refer to the RTI Basics.

If you have already imported an IDL file, you can also use the created .m file instead, which is also faster than creating the IDL file every time the Matlab script starts.

Creating a DDS Domain Participant and setting QoS

In DDS, communication takes place within domains. These domains have a unique domain number, and messages can only be sent or received within a domain. To participate in such a DDS domain, a domain participant needs to be created first. If a program is supposed to participate in more than one domain, then a participant for each domain needs to created.

The domain id used for the HLC should be 1. The HLC should only receive messages from and send commands to the Middleware and thus does not need to partake in the communication between the vehicle, the Middleware, the LabControlCenter or other parts of the project (domain id 0). Thus, only one domain participant needs to be created in Matlab:

```
matlabParticipant = DDS.DomainParticipant('MatlabLibrary::LocalCommunicationProfile', domain_id);
```

In this example, a Domain Participant was created that uses the QoS file referenced in this section. In this case, these settings ensure that all messages sent by the script are only sent within *localhost*. Use this setting to communicate with the Middleware. Further information on QoS XML files can be found in RTI Basics. For custom QoS settings, XML files are the easiest ways to set quality of service parameters for Matlab scripts. Alternatively, the built-in QoS profiles can be used (to see those, type `DDS.getProfiles` in Matlab).

If no QoS settings are required, you can alternatively create a default domain participant:

```
matlabParticipant = DDS.DomainParticipant('', domain_id);
```

Creating a Reader

```
some_reader = DDS.DataReader(DDS.Subscriber(matlabParticipant), 'IDLType', some_topic_name,'SomeLibrary::
Something');
```

The data reader *some_reader* can now be used to receive data. It participates in the domain given by the domain participant *matlabParticipant*. It reads messages of the IDL type *IDLType* and the name of the topic over which the data is sent and received is *some_topic_name*. Further information on topics can be found in RTI Basics. The last parameter is optional and refers to a QoS library from an XML file.

Creating a Writer

```
some_writer = DDS.DataWriter(DDS.Publisher(matlabParticipant), 'IDLType', other_topic_name,'SomeLibrary::
Something');
```

The data writer *some_writer* can now be used to write data. It participates in the domain given by the domain participant *matlabParticipant*. It sends messages of the IDL type *IDLType* and the name of the topic over which the data is sent and received is *other_topic_name*. Further information on topics can be found in RTI Basics. The last parameter is optional and refers to a QoS library from an XML file.

Creating and Using a Filter

```
Filter = DDS.contentFilter;
Filter.FilterExpression = 'vehicle_id = 1';

some_reader = DDS.DataReader(DDS.Subscriber(matlabParticipant), 'IDLType', some_topic_name, 'SomeLibrary::
Something', Filter);
```

The filter must be passed to the reader as an additional argument. The *FilterExpression* is used to filter depending on e.g. the value of a field. In this example, only messages where the field *vehicle_id* is 1 are considered by the reader. More complex filter expressions like `a = 10 AND b > 9` can be realized as well. For more information, see https://community.rti.com/static/documentation/connext-dds/5.3.0/doc/api/connext_dds/api_cpp2/group__DDSQueryAndFilterSyntaxModule.html.

Receiving Messages

A reader can be used to receive messages. Received message objects can be used like any other objects: If an object `sample` has been received by the reader and the IDL type contains a field called *some_value*, then the value of that field can by obtained by typing `sample.some_value`.

Polling

```
sample = YourType;
[sample, status, sampleCount, sampleInfo] = some_reader.take(sample);
```

If one or more messages have been received before *take* was called, *sample* now contains one of these messages. Else, sample is empty. Take also removes the message from the internal buffer of the reader.

Waiting

```
some_reader.WaitSet = true;
some_reader.WaitSetTimeout = some_seconds;
```

A *WaitSet* can be used to specify that, whenever a *take* operation is performed, the reader should wait for new data - if no new messages have been received - until the timeout is reached. As soon as new messages arrive or if some message have not yet been processed, the *take* function returns a new sample. Messages can be obtained as shown in polling after the WaitSet was set.

Sending Messages

```
msg = YourType;
...
some_writer.write(msg);
```

How to create and use IDL types (message objects) was shown before. These messages can be sent by using the *write* function of a data writer of the same IDL type. The message object (here: *msg*) is passed as an argument of that function.

Receiving Parameters

The following code is an example of use of RTI DDS in Matlab:

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/high_level_controller/examples/matlab/direct_control/main.m