DDS Blockset Pilot Support Package (PSP)
User Guide

ISSUE DATE: 8 March 2018

MathWorks

Pilot Engineering Group

Contents

1 Product Description

The DDS Blockset Pilot Support Package (PSP) feature allows Simulink® and MATLAB® models to interact with other simulation components via the OMG Data Distribution Service (DDS) publish/subscribe interface. **DDS** is the first open international middleware standard directly addressing *publish-subscribe* communications for *real-time and embedded systems*.

The DDS Simulink blocks and MATLAB classes use RTI Connext DDS, the market leading implementation of DDS. RTI provides the messaging backbone for the world's most demanding real-time systems. RTI Connext™ enables applications – running on the smallest devices and the largest enterprise servers – to seamlessly share information and work together as one.

Blocks can be added to a Simulink model that will allow the model to interact with other DDS participants during a simulation (via RTI Connext DDS). C/C++ code that is generated from a Simulink model will conform to the RTI Connext DDS API. The generated code can then be compiled and executed on any platform supported by RTI Connext DDS or RTI Connext Micro DDS.

Similarly, instances of MATLAB RTI DDS classes can be created in MATLAB to interact with other DDS participants during a simulation (via RTI Connext DDS). C code generation is currently not supported for the MATLAB RTI DDS classes.

1.1 Acronyms

API – Application Programming Interface

DDS – Data Distribution System

PSP – Pilot Support Package. Customized updates to MATLAB and Simulink software that is not yet available in the officially released version of MATLAB and Simulink.

TLC – Target Language Compiler

1.2 Definitions

1.3 References

1.4 Contact Information

- Mark McBroom – MathWorks. mark.mcbroom@mathworks.com

2 Document History

Date	Version	Author	Description
20 Sept 2012	1.0	MDM	Initial version
18 Nov 2012	1.1	RL	RTI edits
30 Nov 2012	1.2	MDM	Add MATLAB Toolbox content
11 Mar 2013	1.3	MDM	Add description for block return codes
15 Apr 2013	1.4	MDM	Add Linux® install directions
19 Nov 2013	1.5	MDM	Micro DDS support. IDL import, export. IDL 'module' keyword support
9 Dec 2013	1.6	MDM	Add IDL sequence support
15 Mar 2014	2.2	MDM	Improved sequence support for DDS Toolbox. Add sequence support for DDS Blockset simulation. Add DDS.import() support for MATLAB classes. Add support for IDL Inheritance
4 Nov 2014	2.3	MDM	Add optional rtiddsgen switches to DDS.import() Add support for Accelerator and Rapid Accelerator modes Add support for topic content filtering
	2.5.0	MDM	Add support for RTI Connext DDS 5.2.0 and microDDS 2.4.4
21 Oct 2015	2.6.0	MDM	Remove ability to define Topic Type for Toolbox functions using ML Struct or Simulink.Bus
21 Mar 2016	2.8.0	MDM	Add Raspberry Pi™ support
14 Apr 2016	2.8.0	MDM	Add Timestamp support for Data Reader
20 Apr 2016	2.8.0	MDM	Add appendix with instructions for using MATLAB Compiler™
17 May 2016	2.8.0	MDM	Add support for Simulink Data Dictionary
22 Jun 2016	2.8.0	MDM	Add DDS.discoveryMonitor() feature
21 Jul 2016	2.9.0	MDM	Migrate to DDS Connext 5.2.3
17 Nov 2016	3.1.0	MDM	Add MacOS support
30 Jun 2017	3.5.0	MDM	Simulink Data Dictionary support
29 Jan 2018	3.6.0	MDM	Two new blocks that support the XML Application Creation APIs.

3 System Requirements

3.1.1 Required

- MATLAB version R2015b or later
- Simulink
- RTI Connext DDS version 5.1.0, 5.2.0, 5.2.3, or 5.3.0

3.1.2 Optional

To generate code from a Simulink model, the following products are needed:

- Simulink Coder™
- Embedded Coder®

To use Simulink with long or unsigned long data types:

- Fixed-Point Designer™

To compile and link code for RTI Connext Micro DDS:

- RTI Connext DDS Micro 2.2.3 or newer

4 Installation and Setup

4.1 *Installation*

4.1.1 Install RTI Connext DDS

Before installing the DDS Blockset on your computer, you should first install the RTI Connext DDS Blockset. For details or to obtain a license, contact www.rti.com.

Libraries for RTI Connext DDS are provided for various compiler versions. This table lists the RTI Connext DDS libraries for which this support package was developed with. It is highly recommended that you use these versions of RTI Connext DDS libraries when using the blockset. This is done by configuring environment variables as described later in this chapter.

Table 1. Version Support Matrix.

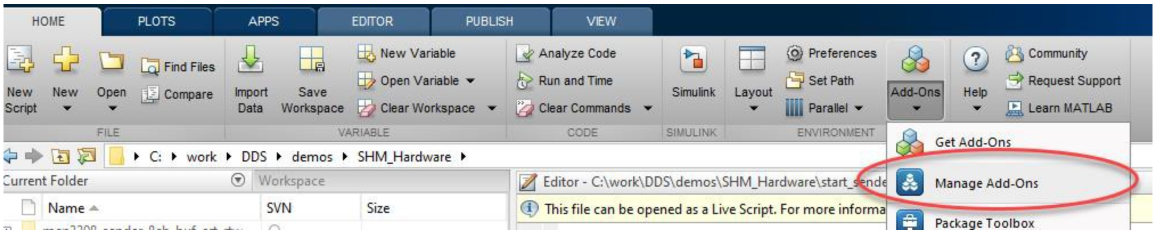
MATLAB Version(s)	OS Version(s)	RTI DDS Library
R2016a, 16b, 17a, 17b	Windows® 64 7, 10	x64Win64VS2013
R2018a	Windows 64 7, 10	x64Win64VS2015
R2016a, 16b, 17a	Linux 64 Debian 7.x	x64Linux2.6gcc4.4.5
R2017b	Linux 64 Debian 8.x	x64Linux2.6gcc4.4.5
R2016b, 17a, 17b	MacOS 10.11	x64Darwin15clang7.0
R2018a	MacOS	X64Darwin16clang8.0

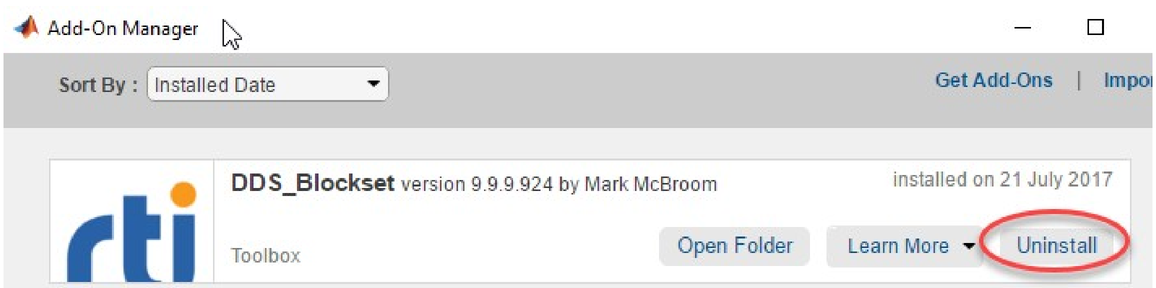
4.1.2 Install MATLAB

4.1.2.1 MATLAB R2017a and Later

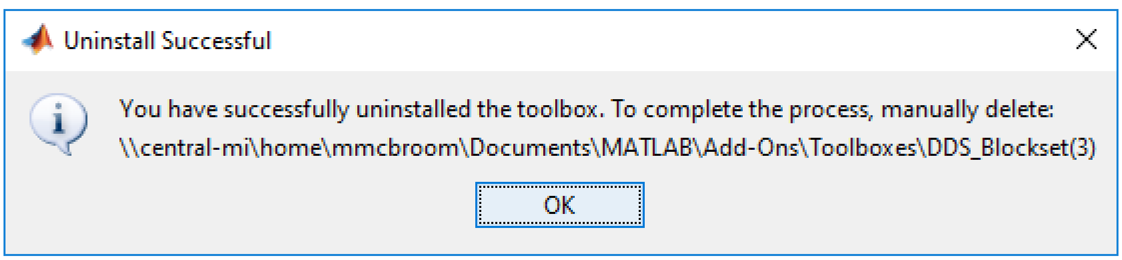
This feature supports all platforms that are supported by MATLAB. The installation program is in the form of a MATLAB Add-On. The same installation package can be used for Windows, Linux, and MacOS.

1. Start MATLAB.
2. If a prior version of the DDS Blockset has been installed, uninstall it.





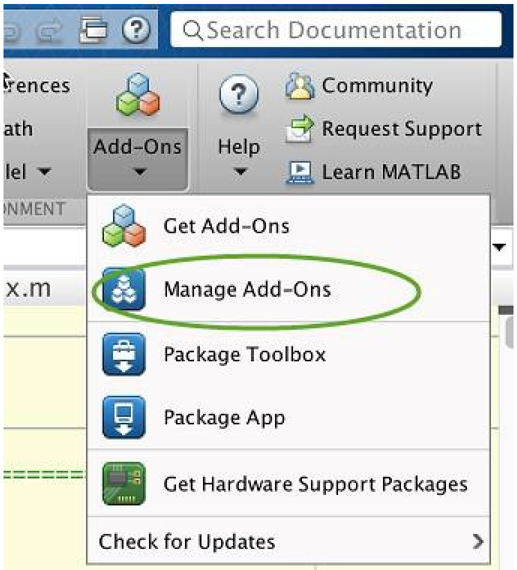
If you see a screen like this, follow the instructions to delete remaining files:

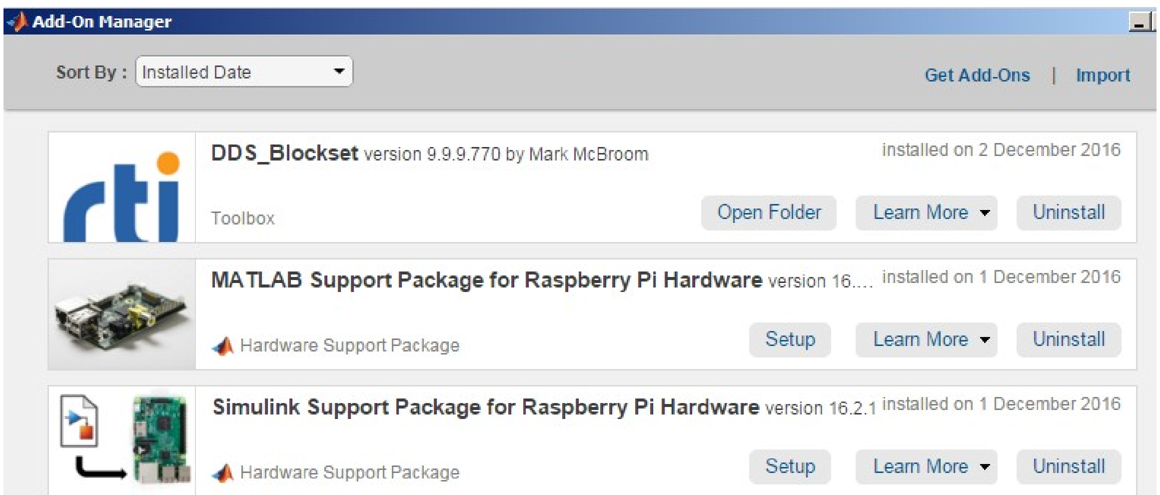


3. Navigate to the location of the .mltbx file that you downloaded to your computer from mathworks.com.
4. Right mouse click and select.

5. Read and accept the licensing agreement.

Once the installation is complete, the DDS Blockset will appear in the list of Add-Ons:





4.1.2.2 MATLAB R2016b and Earlier

The installer for these versions of the blockset is built using InstallJammer. Different installation programs must be selected based on your operating system.

Platform	**Installation Program**
Windows 32/64	DDSBlockset_v3.x.y_win64-Install
Linux 64	DDSBlockset_v3.x.y_glnxa64-Install

4.1.2.2.1 Ubuntu Linux

There is a known issue with Ubuntu® version 14.04 and greater and InstallJammer, which is the software used to build the installer. See section 11 for details of the error message. The workaround is to run the installer from a command shell using the sudo command as follows:

```
sudo ./DDSBlockset-R2016a_v2.x.y_glnxa64-Install --mode console
```

4.2 Environment Variables

For **all** platforms, the environment variables NDDSHOME and RTI_LICENSE_FILE **must** be defined. NDDSHOME points to the location where RTI Connext DDS is installed, while RTI_LICENSE_FILE is the full path to the license file. Note that the **strings should not be terminated with a semicolon.**

The user must also add the location of the DDS shared libraries to the appropriate environment variable so that the shared libraries can be located by the operating system when invoked from MATLAB and Simulink.

4.2.1 Windows

Table 2. Example for DDS Connext 5.2.0 and later.

Env Variable	**Example value**
NDDSHOME	C:\Program Files\rti_connext_dds-5.2.0
RTI_LICENSE_FILE	C:\Program Files\rti_connext_dds-5.2.0\rti_license.dat
PATH	C:\Program Files\rti_connext_dds-5.2.3\lib\x64Win64VS2013

If the user will be generating C code from a Simulink model that is targeted for RTI Micro DDS, then the following two environment variables must also be defined:

Env Variable	**Example value**
RTIMEHOME	C:\Program Files\rti_connext_micro.2.4.4
RTIMEARCH	i86Win32VS2010

4.2.2 Linux

Table 3. Example for DDS Connext 5.2.0 and later.

Env Variable	Example value
NDDSHOME	/usr/rti_connext_dds-5.2.0
RTI_LICENSE_FILE	/usr/rti_connext_dds-5.2.0/rti_license.dat
LD_LIBRARY_PATH	/usr/ rti_connext_dds-5.2.0/lib/x64Linux2.6gcc4.4.5

If the user will be generating C code from a Simulink model that is targeted for RTI Micro DDS, then the following two environment variables must also be defined:

Env Variable	Example value
RTIMEHOME	/usr/rti_connext_micro.2.4.4
RTIMEARCH	i86Linux2.6gcc4.4.5

4.2.3 MacOS

Table 4. Example for DDS Connext 5.2.0 and later.

Env Variable	Example value
NDDSHOME	/Applications/rti_connext_dds-5.2.3
RTI_LICENSE_FILE	/Applications/rti_connext_dds-5.2.3/rti_license.dat
RTI_LD_LIBRARY_PATH	/Applications/ rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0
DYLD_LIBRARY_PATH	/Applications/ rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0
DEVELOPER_DIR	/Applications/Xcode.app/Contents/Developer
JREHOME	See Note 2

NOTE 1. Beginning with MacOS v10.11 (El Capitan), the use of the environment variable DYLD_LIBRARY_PATH is restricted due to a security concern. Refer to section 12 for details.

NOTE 2. You may need to also define JREHOME environment variable. If JREHOME is not defined, then you will see an error message similar to this.

```
>> DDS.import('dSequence.idl')
Warning: OS Darwin may not be supported. Be sure JREHOME is set.
/sandbox/mmcbroom/rti_connext_dds-5.3.0/bin/../resource/app/jre/darwin
```

You will need to locate a Java Runtime Environment™ (JRE) on your computer, or install, and then point JREHOME to the location where you installed. MATLAB also includes a JRE. You can

also point JREHOME to the MATLAB JRE, which is located at:
<matlabroot>/sys/java/jre/maci64/jre

You can set the JREHOME environment variable at the shell prompt prior to starting MATLAB, your you can set in MATLAB with the following command
>setenv('JREHOME', fullfile(matlabroot,'sys','java','jre','maci64','jre'))

4.3 Alternate Approach

This section describes an alternate approach to setting environment variables. This approach can be used if you do not have sufficient privilege to define environment variables.

1. Set NDDSHOME and PATH environment variables from MATLAB. Type the following lines at the MATLAB prompt to temporarily create two environment variables. Update as necessary for the location in which RTI Connext DDS is installed. Note that you will need to type these two commands each time you start MATLAB. You can place these into a script that runs each time MATLAB starts.

```
> setenv('NDDSHOME','C:\Program Files\rti_connext_dds-5.2.0');
> CurrentPath = getenv('PATH');
> setenv('PATH',[CurrentPath, ';C:\Program Files\rti_connext_dds-
5.2.0\lib\x64Win64VS2012']);
```

2. Use alternate approach to specify license file. Following are instructions from **RTI_ConnextDDS_CoreLibraries_GettingStarted.pdf** section 2.4.1. to specify the location of the RTI License File in the QoS XML file. The following approach has been successfully tested.

2.4 License Management

Most package types (Professional, Basic, and Evaluation) require a license file in order to run.

If your Connext DDS distribution requires a license file, you will receive one from RTI via email.

If you have more than one license file from RTI, you can concatenate them into one file.

A single license file can be used to run on any architecture and is not node-locked. You are not required to run a license server.

2.4.1 Installing the License File

Save the license file in any location of your choice; the locations checked by the middleware are listed below.

You can also specify the location of your license file in *RTI Launcher's* **Configuration** tab. Then *Launcher* can copy the license file to the installation directory or to the user workspace.

Each time your Connext DDS application starts, it will look for the license file in the following locations until it finds a valid license:

2. In the PropertyQosPolicy of the *DomainParticipant*, there may be a property called **dds.license.license_file**. The value for this property can be set to the location (full path and filename) of a license file. (This may be necessary if a default license location is not feasible and environment variables are not supported.) You can set the property either in source code or in an XML file.

   Example XML to set **dds.license.license_file**:

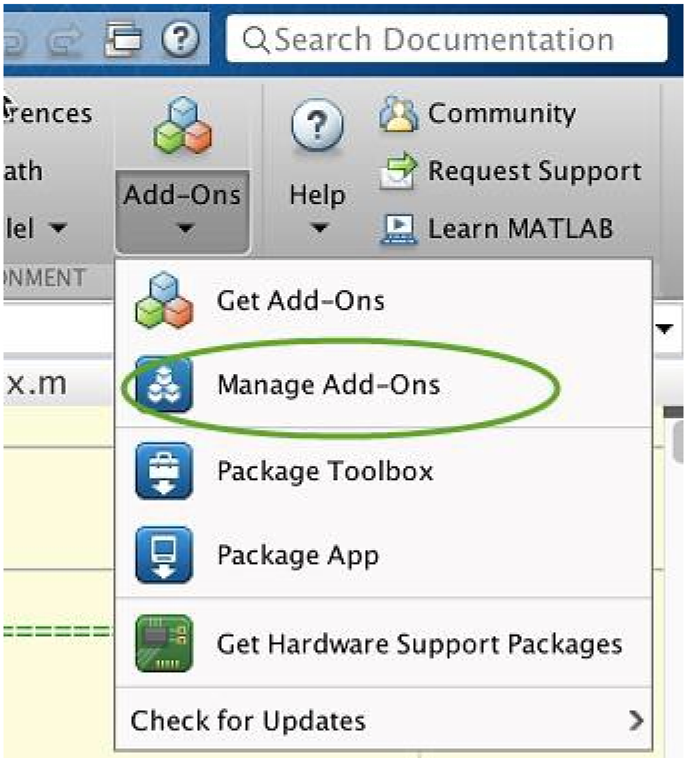
```
<participant_qos>
    <property>
        <value>
            <element>
                <name>dds.license.license_file</name>
                <value>path to license file</value>
            </element>
        </value>
    </property>
</participant_qos>
```

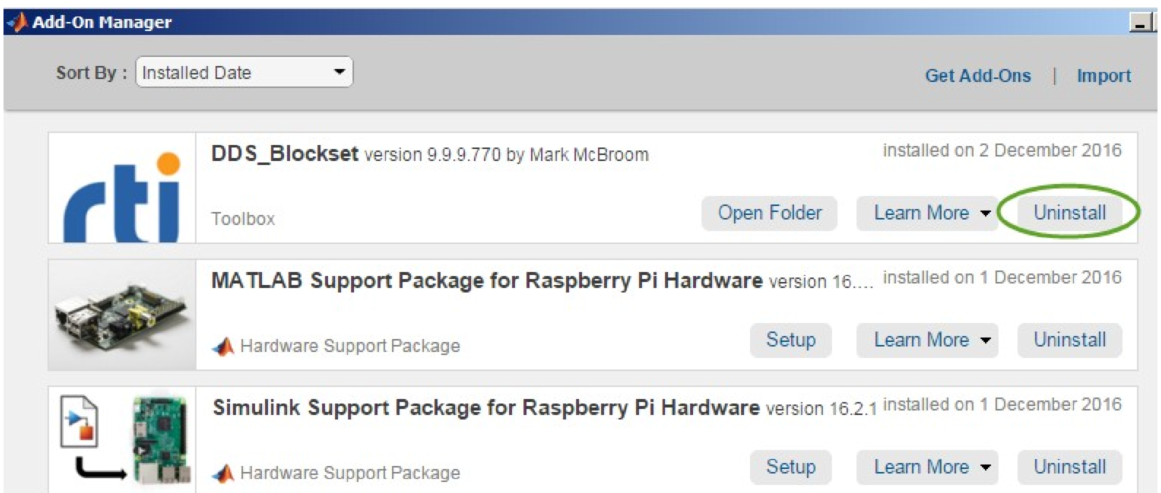
Figure 1. RTI License File Management.

4.4 *Uninstall*

4.4.1 MATLAB R2017a and Later

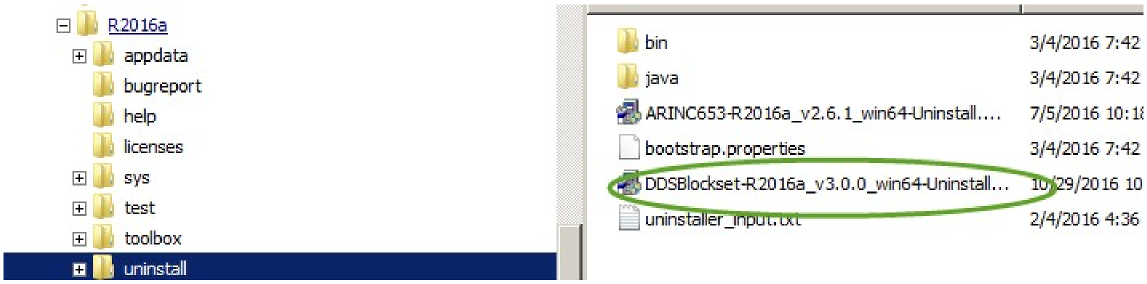
For versions R2017a and later, the DDS Blockset can be unistalled from the MATLAB Add-On manager as follows:





4.4.2 MATLAB R2016b and Earlier

For MATLAB R2016b and earlier, there will be an unistaller located in the <matlabroot>/uninstall folder. Launch this from your file explorer.


R2016a
appdata
bugreport
help
licenses
sys
test
toolbox
uninstall
bin
3/4/2016 7:42
java
3/4/2016 7:42
ARINC653-R2016a_v2.6.1_win64-Uninstall....
7/5/2016 10:1
bootstrap.properties
3/4/2016 7:42
DDSBlockset-R2016a_v3.0.0_win64-Uninstall...
10/29/2016 10
uninstaller_input.txt
2/4/2016 4:36

5 Getting Started

The DDS Blockset provides Simulink blocks for the five key DDS entities:

- Domain Participants
- Publishers
- Subscribers
- Data Writers
- Data Readers

The blockset also provides two blocks that use the XML Application Creation capability of DDS to create Data Readers and Data Writers in conjunction with an XML configuration file.

The blocks appear in the Simulink Library Browser as shown below. Type “simulink” at the MATLAB prompt to display the Simulink Library Browser.

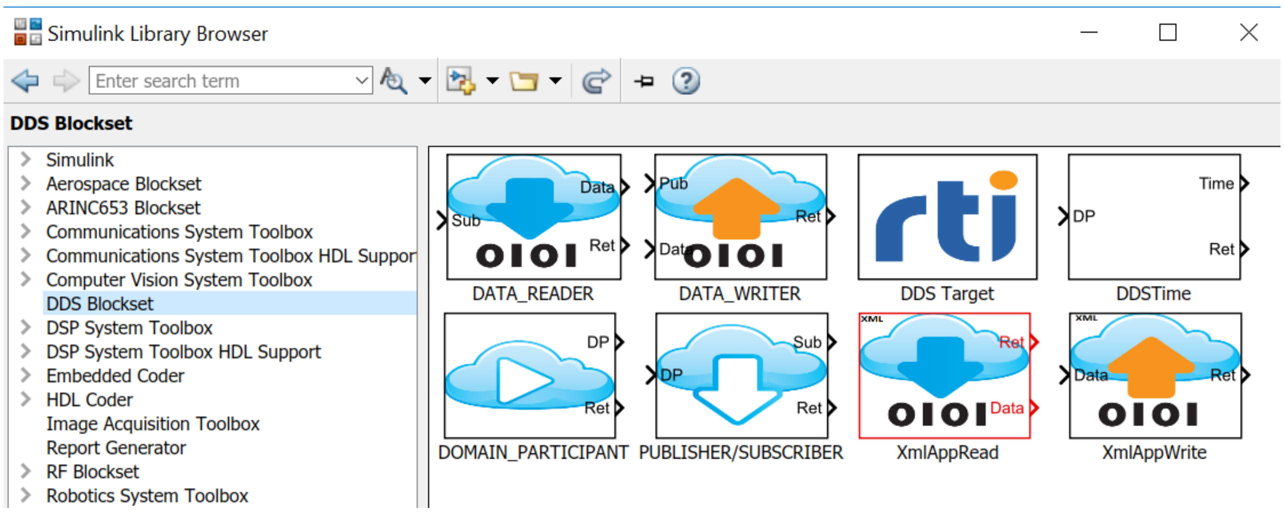


Figure 2. DDS Blockset in the Simulink Library Browser.

5.1 Basic Model

Follow these steps to create a simple Simulink model that contains the Domain Participant block:

1. Create a new Simulink model by selecting “File -> New -> Model” in the menu of the Simulink Library Browser window.
2. Drag a Domain_Participant block into the new Simulink model.
3. Drag Terminator block and Display blocks (in the Simulink-Sinks library) into the model. Connect as shown below.
4. Save the model file by clicking File->Save.
5. Push the Run button.

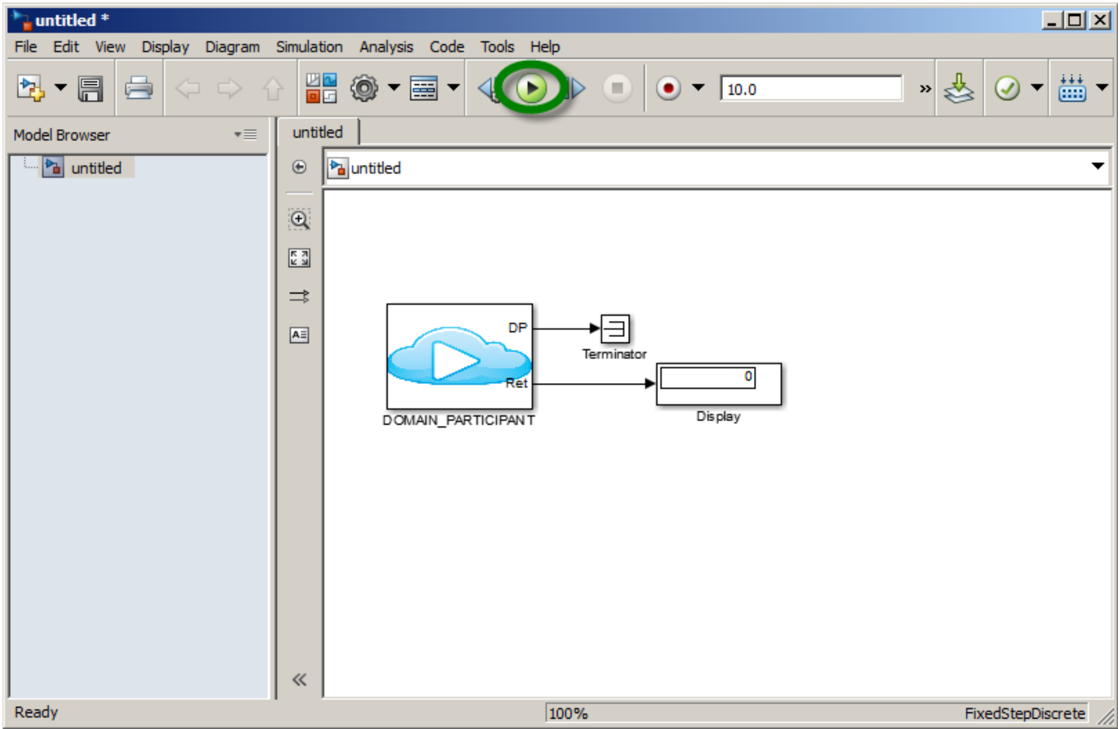


Figure 3. Simulink model with DDS Block.

If you are using the evaluation version of RTI Connext, the DOS output window should show status like the following:

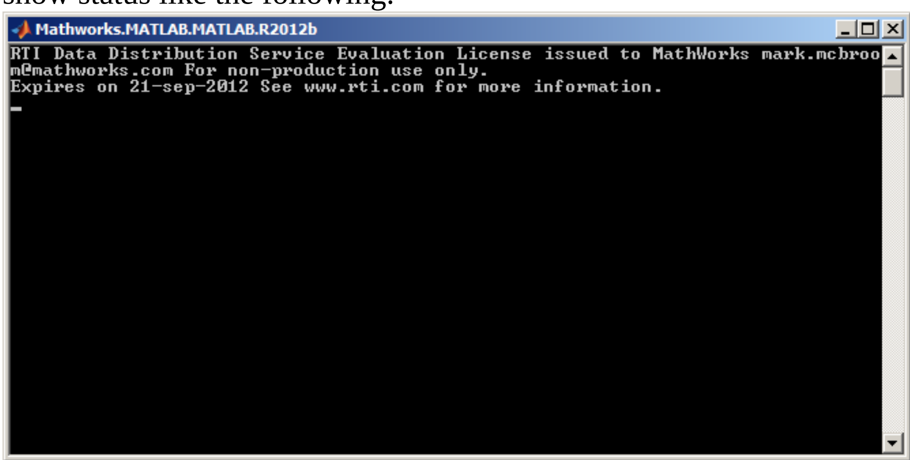


Figure 4. RTI DDS Output Window.

5.2 *Complete Model*

Follow these steps to create a Simulink model that publishes data to DDS and then subscribes to the same data.

If you would rather not create the following model, it is provided as part of the PSP. Open the model by typing: rtwdemo_DDSBasic.

5.2.1 Create a Simulink Bus

DDS Topic Types are represented in Simulink with a bus. Most DDS workflows define Topic Types in IDL files. The first step in this workflow is to create a Simulink Bus object from an IDL file.

DDS.import('BusObject.idl');

5.2.2 Create a Simulink Model

We will now create a Simulink model that sends and receives data for the Simulink Bus/DDS Topic type "BusObject".
Drag and drop blocks from the DDS Blockset into a new model to look like this:

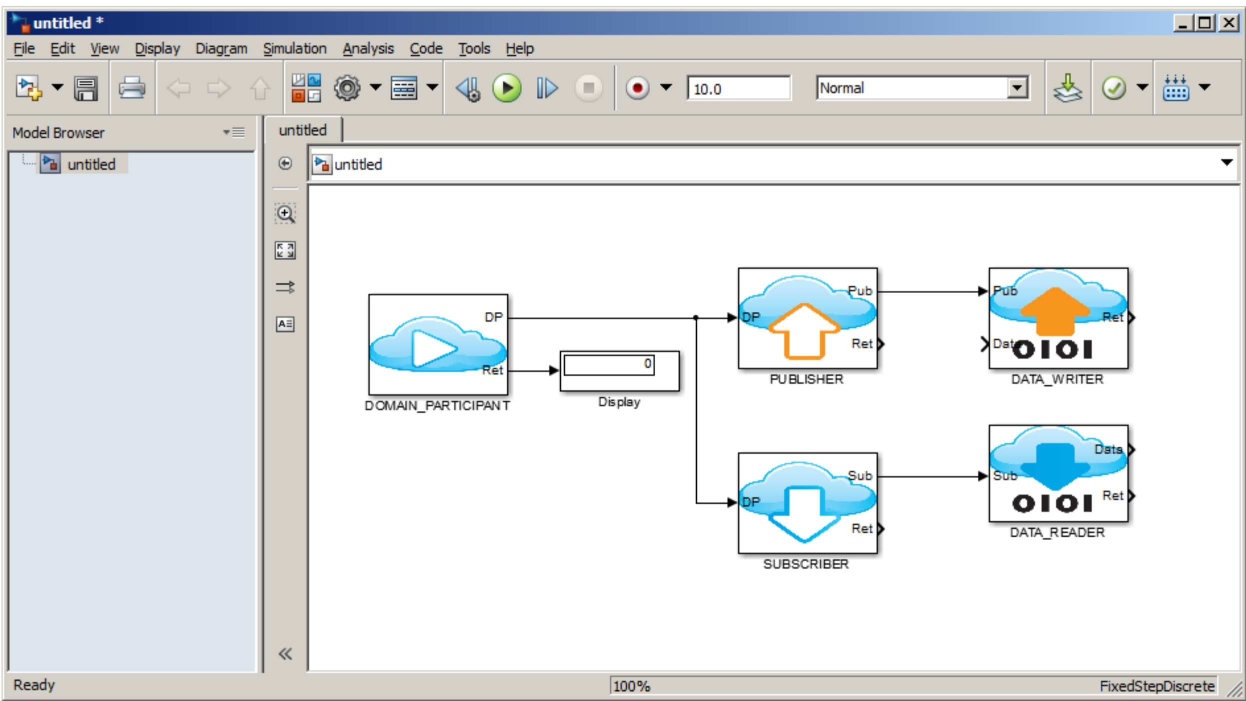


Figure 5. Example Model.

Note that for the Publisher_Subscriber block connected to the Data Writer, you will need to double click on the block and configure it as a Publisher.

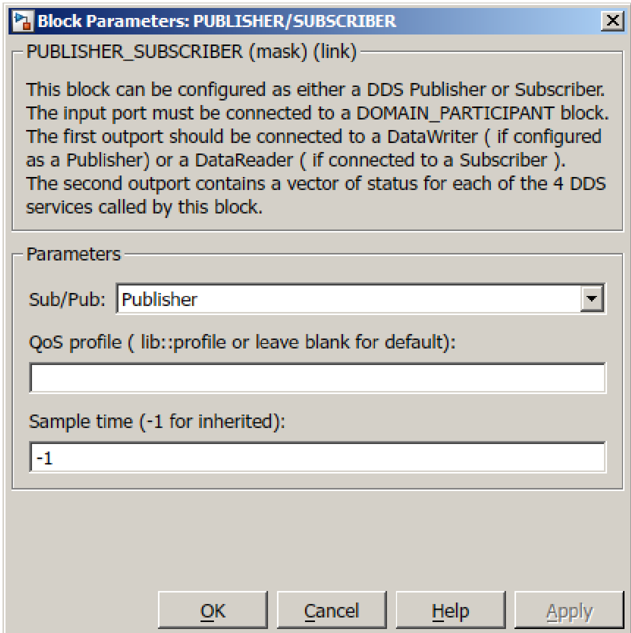


Figure 6. Publisher Block Dialog.

Next, configure the data writer block for the “BusObject” Topic Type. Double click on each block and change the Topic Type as shown below:

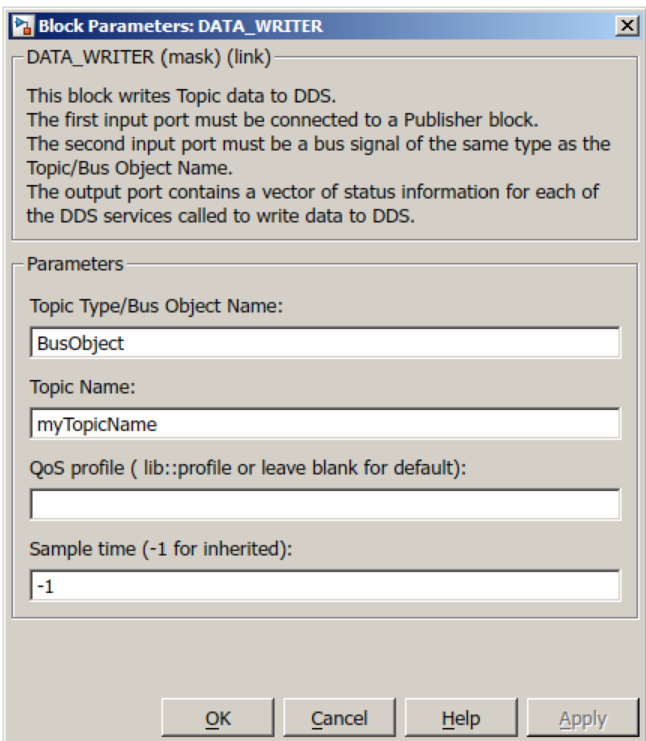


Figure 7. Data Writer Block Dialog.

Now, configure the Data Reader block to read the same Topic Type and Topic Name. These two fields must be identical to the Data Writer block.

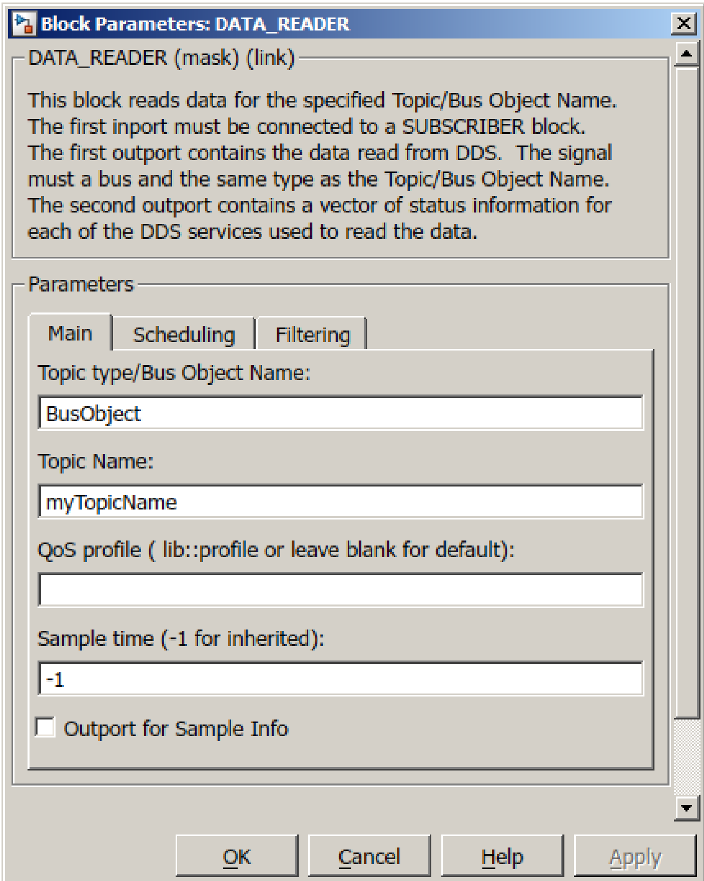


Figure 8. DataReader Block Dialog.

Click on the “Scheduling Tab” and make the following changes:

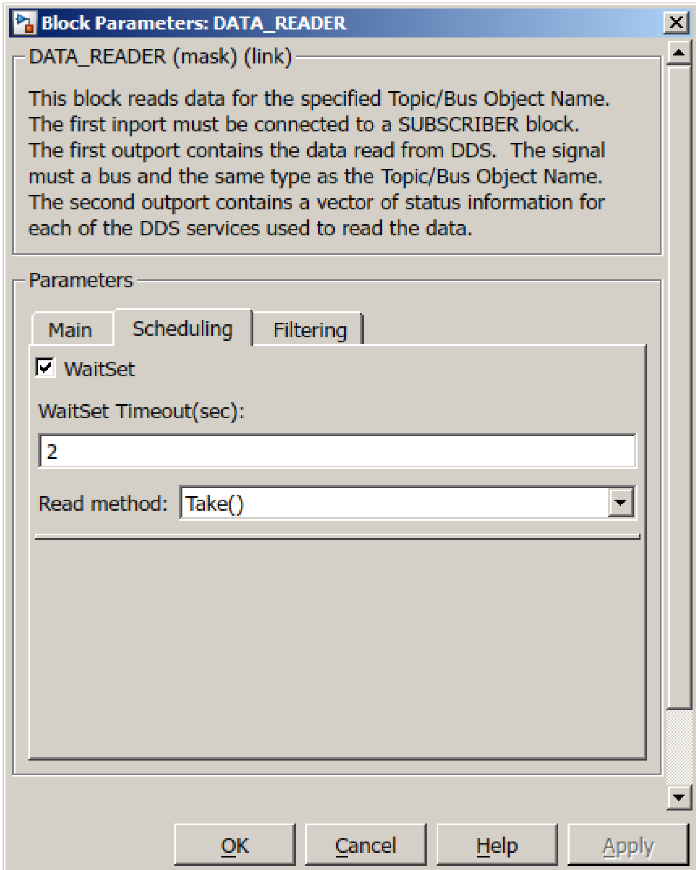


Figure 9. Data Reader Block Dialog.

Although the model will now simulate and interact with DDS, the last step involves writing non-zero data and then viewing the result.
Add Sine, Bus Creator, Bus Selector, and Scope blocks as shown below.

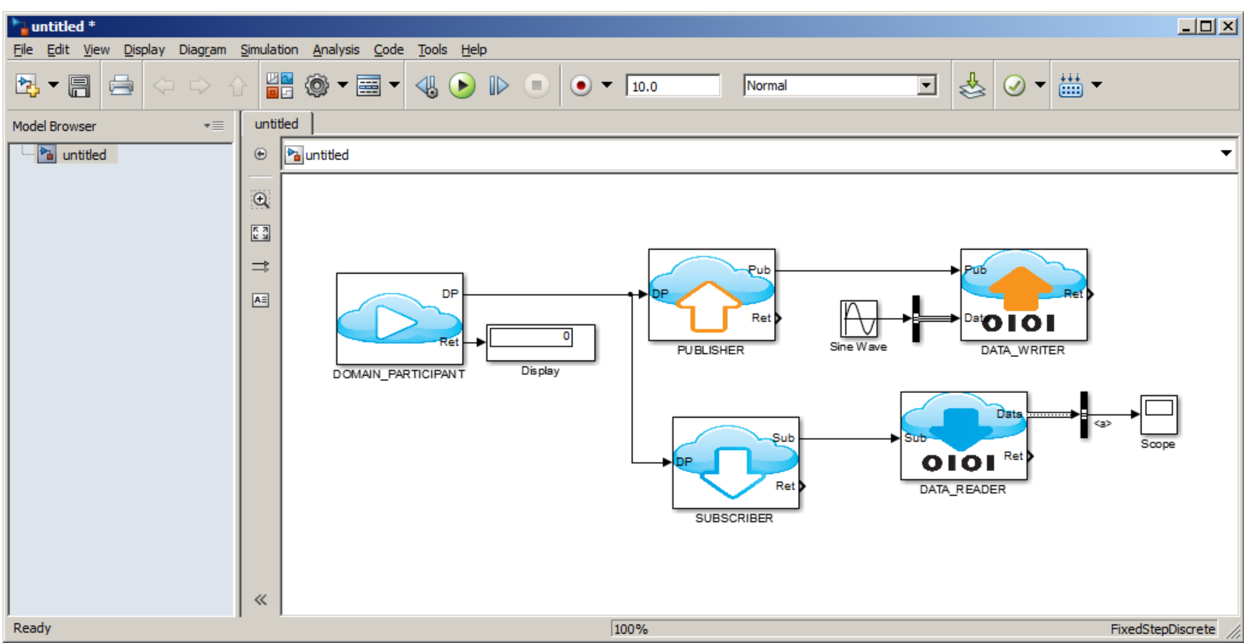


Figure 10. Example Model.

Double click on the Bus Creator block and change to the following:

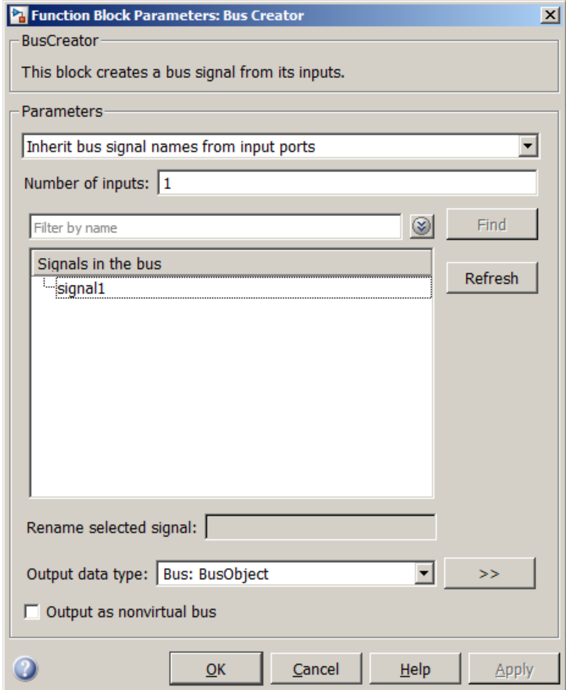


Figure 11. Bus Creator Block Dialog.

Double click on the Bus Selector block and change to the following:

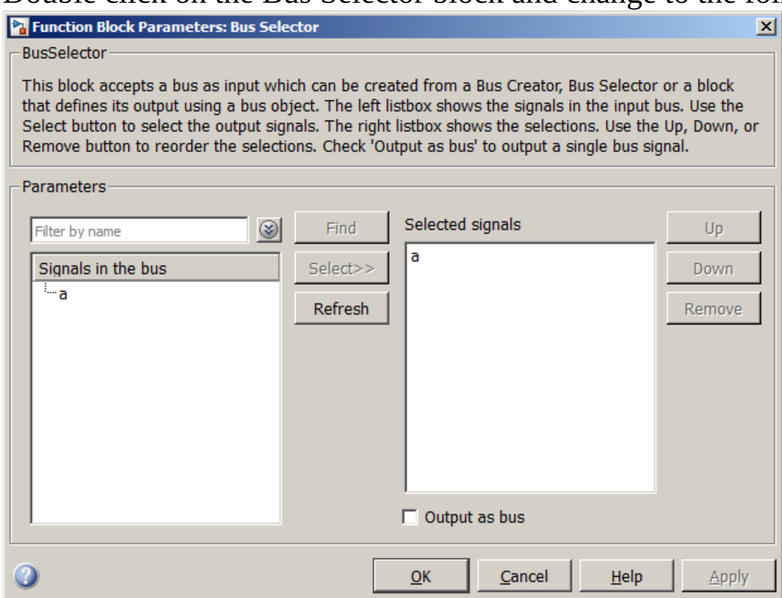


Figure 12. Bus Selector Block Dialog.

The model is now complete. Push the Simulate button.

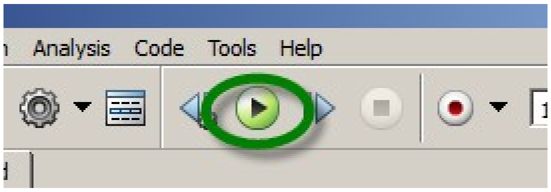


Figure 13. Simulink Run Button.

When the simulation is complete, double click on the Scope block. You should see the following:

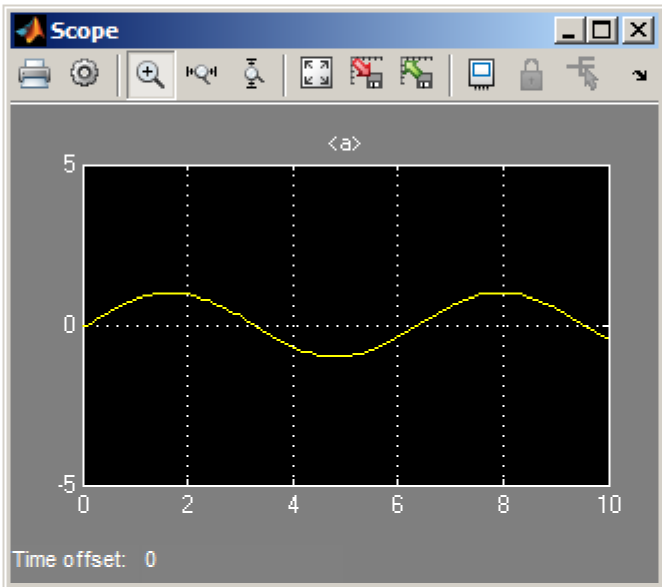


Figure 14. Simulink Scope Block.

5.3 Code Generation

With Simulink Coder and Embedded Coder licenses, code can be generated from a Simulink model that contains DDS Blocks. Follow instructions in the MATLAB documentation for [configuring]() and [generating code]() from a Simulink model. The generated code will have calls to RTI DDS functions for each of the DDS Blocks in the Simulink model. For example, the following code fragment is generated for a Domain Participant block:

```
    untitled_B.DOMAIN_PARTICIPANT_o1 =
      DDS_DomainParticipantFactory_create_participant_with_profile
      (DDS_TheParticipantFactory, 0, "UserQosProfilesLibrary", "MonitorDefault",
       NULL, DDS_STATUS_MASK_NONE);
```

For a subscriber:

```
      untitled_B.SUBSCRIBER_o1 =
        DDS_DomainParticipant_create_subscriber_with_profile
        (untitled_B.DOMAIN_PARTICIPANT_o1, "UserQosProfilesLibrary",
         "MonitorDefault", NULL, DDS_STATUS_MASK_ALL);

```

And for a data reader:

```
      struct DDS_DataReaderQos dataReaderQos = DDS_DataReaderQos_INITIALIZER;
      DDS_DomainParticipant* domainParticipant = DDS_Subscriber_get_participant
        (untitled_B.SUBSCRIBER_o1);

      /* register this data type and Topic with DDS.  */
      RegisterTypeTopic(domainParticipant, &untitled_DWork.DATA_READER_MsgTopic,
                        untitled_DWork.DATA_READER_CStructSize, "BusObject",
                        "myTopicName", "UserQosProfilesLibrary", "MonitorDefault",
                        &Simulink_BusObject, (DDS_ReturnCode_t*)
                        untitled_B.DATA_READER_o2 );

      /* get QoS from user specified library/profile */
      untitled_B.DATA_READER_o2[5] =
        DDS_DomainParticipantFactory_get_datareader_qos_from_profile
        (DDS_TheParticipantFactory, &dataReaderQos, "UserQosProfilesLibrary",
         "MonitorDefault");

      /* create data reader using this QoS. */
      untitled_DWork.DATA_READER_MsgDataReader = DDS_Subscriber_create_datareader
        (untitled_B.SUBSCRIBER_o1, DDS_Topic_as_topicdescription((DDS_Topic*)
          untitled_DWork.DATA_READER_MsgTopic), &dataReaderQos, NULL,
         DDS_STATUS_MASK_ALL);
```

5.3.1 RTI DDS Target Block

By default, code generated from a Simulink model will be compatible with RTI DDS Connext using static typing. The code for defining, registering, and accessing topic samples will be generated by the rtiddsgen utility provided by RTI. However, the user can override these settings by adding an RTI DDS Target block to the model.

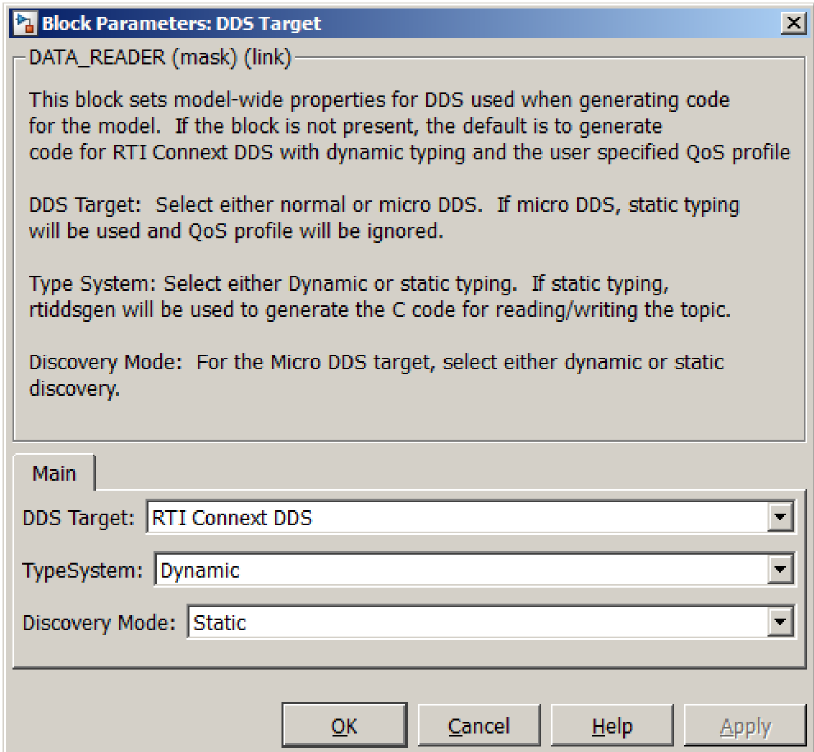


Figure 15. DDS Target Block Dialog.

5.4 RTI DDS Connext Toolbox

The DDS Blockset PSP also includes a set of MATLAB classes that can be used to access RTI DDS Connext from MATLAB. This section explains how to create instances of the MATLAB DDS Connext classes.

Type the following at the MATLAB Prompt.

- DDS.import(‘ShapeType.idl’,’matlab’);
- myTopic = ShapeType;
- myTopic.x = int32(23);
- myTopic.y = int32(35);
- dp = DDS.DomainParticipant
- dp.addWriter(‘ShapeType’, ‘Square’);
- dp.write(myTopic);
- dp.addReader(‘ShapeType’, ‘Square’);
- readTopic = dp.take();

The workspace variable readTopic should be an object of type ShapeType. The “x” and “y” properties of the class should have values of 23 and 35, respectively.

6 Examples

6.1 Simulink/RTI Shapes

The DDS Blockset includes a Simulink model that will interact with the RTI Shapes Demo. This demo is installed as part of RTI Connext DDS.

6.1.1 Start the RTI DDS Connext Shapes Demo

Open the Connext Launcher:

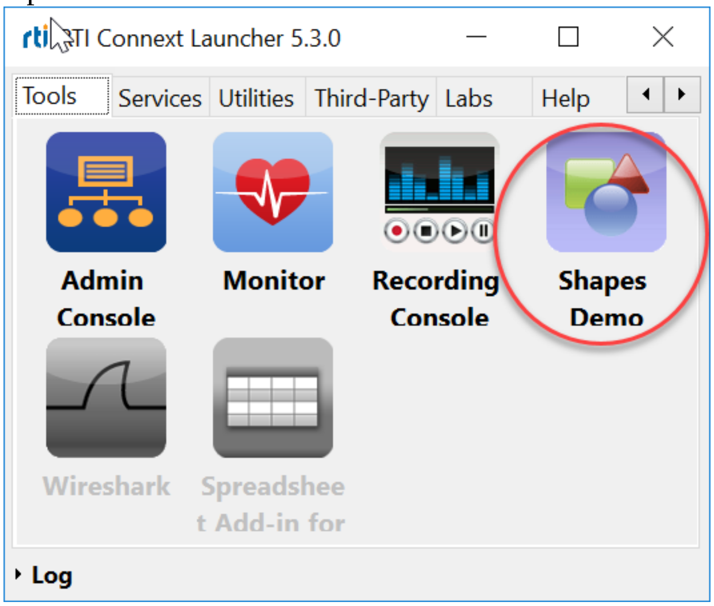


Figure 16. RTI Connext Launcher.

Select the Shapes Demo.

Make the following selections:

1. Select Publish – Square
    a. Set "initial size" = 15
    b. OK
2. Select Subscribe – Circle
    a. OK
3. Select Subscribe – Triangle
    a. OK

When complete, your display should look like this:

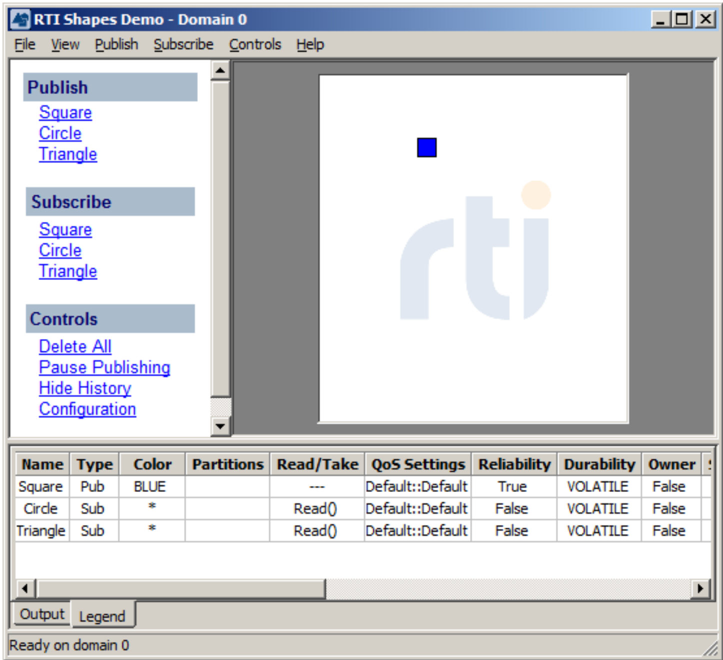


Figure 17. RTI Connext Shapes Demo.

6.1.2 Simulink rtwdemo_RTIShapes Model

Start MATLAB. Before you run the RTI Shapes Simulink model, verify you have a C++ compiler configured. At the MATLAB prompt, type: mex –setup.

You should be prompted for a list of available compilers. If you are using Visual Studio®, you may need to download Microsoft® Windows SDK7.1 before Visual Studio will show up in the list of selectable compilers. See this article link for more information.

At the MATLAB prompt, type: rtwdemo_RTIShapes

Once the model is loaded, press the run button to start the simulation.
You will now see eight triangles and eight circles appear in the RTI Shapes Demo display. The Simulink model is computing the position, velocity, and acceleration of the triangle and circle shapes. The positions of each of these shapes is sent to the RTI Shapes Demo via a DDS Writer. Simulink is reading the Square that is published by the RTI Shapes Demo and using the information from this shape to compute collisions with the other 16 shapes being published by Simulink.

To make the demo more interesting, right-mouse click on the square shape and change its direction and speed.

Note that the Simulink model will stop simulation after about 30 seconds. Push the run button on the Simulink model to resume the example.

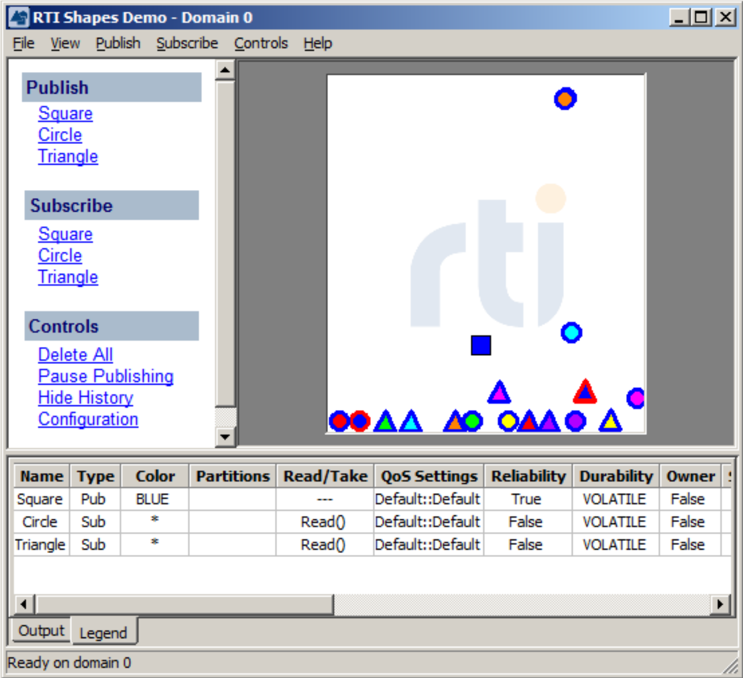


Figure 18. RTI Connext Shapes Demo.

6.2 MATLAB/RTI Shapes

1. Follow steps in 6.1.1 to start the RTI Shapes Demo.
2. Type the following at the MATLAB prompt to create a data reader:

```
% Create a MATLAB class from the ShapeType IDL definition
DDS.import('ShapeType.idl','matlab', 'f');

%% create a DDS Domain participant and data reader
dp = DDS.DomainParticipant;
dp.addReader('ShapeType', 'Square');
```

3. Type the following at the MATLAB prompt. You will see output similar to the following, which is the current position of the blue square in the RTI Shapes Demo.

```
dp.take()

ans =

        color: 'BLUE'
            x: 143
            y: 167
    shapesize: 15
```

4. Type the following at the MATLAB prompt to send a purple circle to the RTI Shapes Demo.

```
dp.addWriter('ShapeType', 'Circle');
myData = ShapeType;
myData.x = int32(20);
myData.y = int32(40);
```

```
myData.shapesize = int32(20);
myData.color = 'PURPLE';
dp.write(shapeData);
```

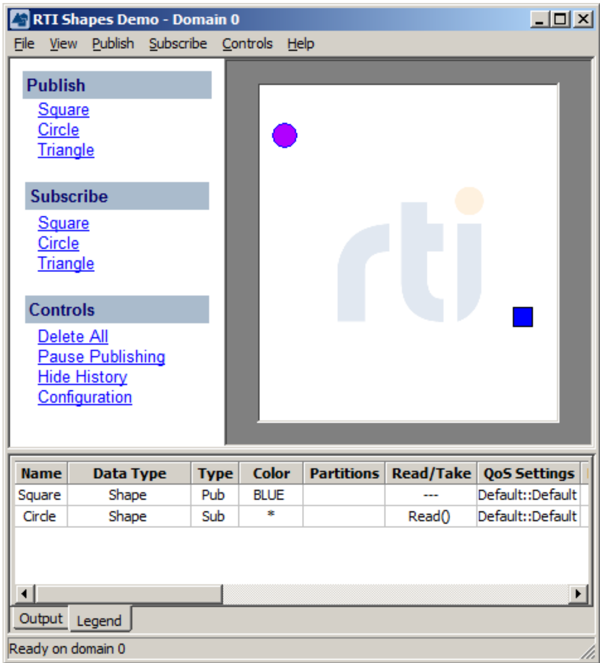


Figure 19. RTI Shapes Demo.

7 Blockset Reference

7.1 DDS Types

Most DDS implementations, including RTI, support definition of data types via a language independent description language. OMG IDL is the most commonly used, but XML and XSD are also often supported. Simulink has a set of built-in data types along with the ability to define structured data types via Simulink Buses and enumerated data types via Simulink enumerated data type. Similarly, MATLAB has built-in data types along with the ability to define structured data types via MATLAB classes/structures and enumerated data via MATLAB enumerated data types. The following table defines the IDL, DDS and MATLAB and Simulink data type mapping and support.

Table 1. DDS Type Support in MATLAB and Simulink.

IDL	***DDS***	***MATLAB Type***	***Simulink Type***
short	DDS_TK_SHORT	int16	int16
long	DDS_TK_LONG	int32	int32
unsigned short	DDS_TK_USHORT	uint16	uint16
unsigned long	DDS_TK_ULONG	uint32	uint32
float	DDS_TK_FLOAT	single	single
double	DDS_TK_DOUBLE	double	double
boolean	DDS_TK_BOOLEAN	logical	boolean
char	DDS_TK_CHAR	int8	int8
octect	DDS_TK_OCTECT	uint8	uint8
struct	DDS_TK_STRUCT	MATLAB Struct/Class	Simulink.Bus
union	DDS_TK_UNION	**Not Supported**	**Not Supported**
enum	DDS_TK_ENUM	MATLAB Enumeration	Simulink.Enum
String<maxlen>	DDS_TK_STRING	char	*DDS_CharArray
sequence	DDS_TK_SEQUENCE	Supported	**Note 1**
“[]” notation	DDS_TK_ARRAY	Supported	Supported
	DDS_TK_ALIAS	**Not Supported**	**Not Supported**
long long	DDS_TK_LONGLONG	Supported	Supported
unsigned long long	DDS_TK_ULONGLONG	Supported	Supported
long double	DDS_TK_LONGDOUBLE	**Not Supported**	**Not Supported**
Wchar	DDS_TK_WCHAR	**Not Supported**	**Not Supported**
Wstring	DDS_TK_WSTRING	**Not Supported**	**Not Supported**

?	DDS_TK_VALUE	**Not Supported**	**Not Supported**
?	DDS_TK_SPARSE	**Not Supported**	**Not Supported**
?	DDS_TK_RAW_BYTES	**Not Supported**	**Not Supported**
?	DDS_TK_RAW_BYTES_KEYED	**Not Supported**	**Not Supported**

Note 1: For code generation, only Dynamic Data type mode is supported. For static code generation, errors will occur during code generation and/or execution of the generated code.
Note 2: For long long and unsigned long long support in Simulink, the Fixed-Point Designer is required.

7.1.1 Representing DDS Types in Simulink

The DDS Simulink blocks assume that the Topic Type will always be in the form of an IDL struct which is modeled in Simulink as a Simulink Bus. The user must create a Simulink Bus with the desired fields/types/sizes for the Topic data to be sent and received. If the Topic Type is defined in IDL, the user must make sure to define the Simulink Bus with the same attributes.

It is strongly recommended that the user create Simulink Buses using the DDS.import() utility. See section 7.1.2. This utility creates buses from struct definitions in IDL files, including all metadata data (i.e. @key, @optional, sequence, etc). required for proper DDS operation.

Figure 20 shows how a Topic Type with nested structures can be represented in Simulink with a corresponding set of nested Simulink Buses.

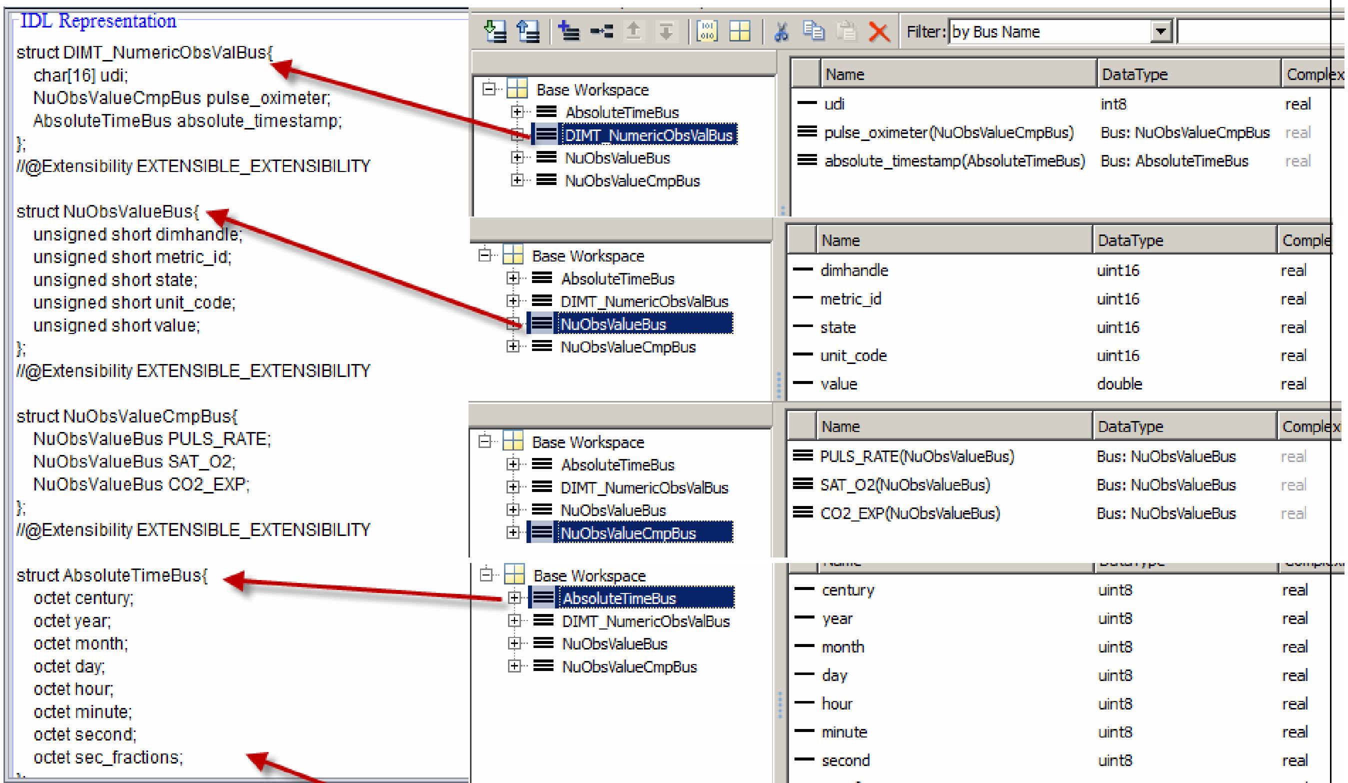


Figure 20. Topic Type Defined by Simulink Bus.

If the Topic Type contains enumerated data types, a Simulink enumerated data type must be created with the same name and enumerators. This link explains how to create and use an enumerated data type.

MATLAB and Simulink do not have built-in support for strings. Strings are therefore treated in Simulink as an array of unsigned bytes. In order for the Simulink Blocks to differentiate between a vector of bytes and a string, an alias data type, DDS_CharArray, was created. This alias type is used by the DDS Blockset infrastructure to differentiate between strings and byte vectors.

7.1.2 Importing IDL into Simulink

The recommended technique for creating buses in Simulink is to import the bus definitions from a DDS IDL or XML file using the following function. By default, the buses will be created in the MATLAB base workspace. The 'sldd' option will import the buses into the user specified Simulink Data Dictionary.

- DDS.import('IDL/XML file name')
- DDS.import('IDL/XML filename', 'f') – 'f' overwrites any existing objects in the workspace.
- DDS.import('IDL/XML filename', 'sldd', 'myData.sldd')
- DDS.import('IDL/XML filename', '-ppDisable') : –ppDisable command line switch to disable IDL preprocessor for rtiddsgen.

Table 2 defines IDL keywords/XML tags that are supported by the DDS.import command and the resulting MATLAB and Simulink entity created. Note that the classes DDS.Bus, DDS.Parameter, and DDS.AliasType are derived from built-in classes Simulink.Bus, Simulink.Parameter, and Simulink.AliasType. These derived classes hold additional meta data needed for proper interaction with DDS, such as key field, sequence, and IDL module.

Table 2. IDL Import/Export Keyword Support for Simulink.

IDL Keyword	XML tag	Simulink	Comment
Struct	<struct>	DDS.Bus	
n/a	<member>	DDS.BusElement	
Enum	<enum>	Simulink.Enum	A dynamic enum will be created.
Const	<const>	DDS.Parameter	
Typedef	<typedef>	DDS.AliasType	

7.1.3 Exporting Buses to IDL

DDS topics defined with buses can be exported to DDS IDL with the function DDS.export(<busName>). Table 2 describes the Simulink entities that will be exported.

7.1.4 Representing DDS Types in MATLAB

Topic Types in MATLAB are represented by MATLAB classes. The MATLAB class for a Topic Type is required when creating a data reader or data writer using the addReader() or addWriter() method.

A MATLAB class must be created for each IDL structure in the Topic Type. The classes must contain a “properties” section to define the type and size of each element. Optional methods can be provided to indicate key fields (see section 7.1.6.2), sequences, and IDL Module keyword information. Note each IDL structures, including the top structure for the Topic Type, must each be represented with a MATLAB class.

Although the classes can be created with a text editor, **it is strongly recommended that the user create MATLAB classes using the DDS.import() utility**. See section 7.1.5 for details. This utility creates MATLAB classes from struct definitions in IDL files, including all metadata data (i.e. @key, @optional, sequence, etc). required for proper DDS operation.

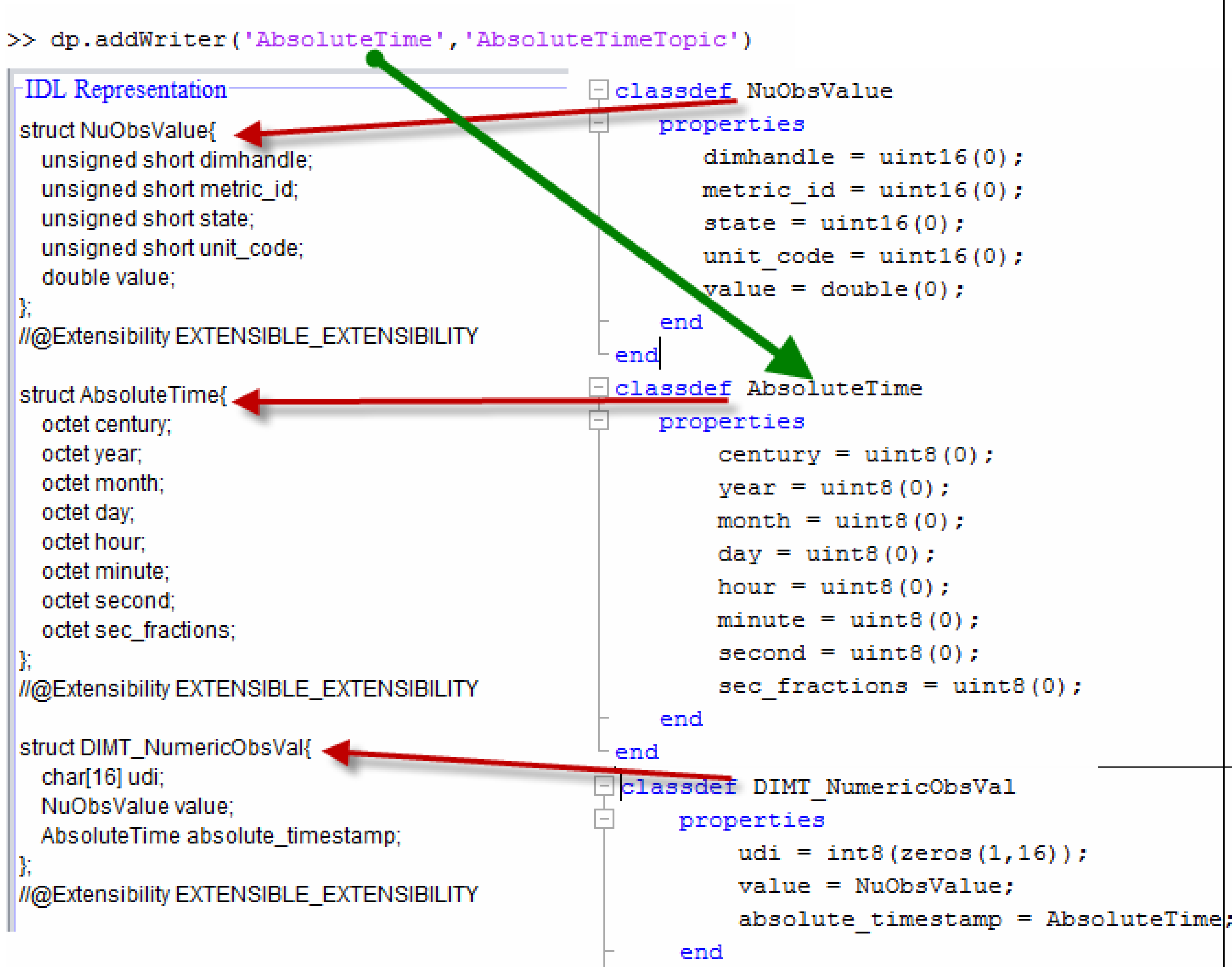


Figure 21. Topic Type Defined with MATLAB Class.

7.1.5 Importing IDL into MATLAB

The DDS.import() utility can be used to automatically create MATLAB classes from an IDL or XML file.

DDS.import('HelloWorld.idl','matlab')
DDS.import('HelloWorld.idl','matlab', 'f')
where:

- 'HelloWorld.idl' – IDL file containing Topic Type(s)
- 'f' – Force over-write if file already exists
- 'matlab' – create MATLAB classes for each Topic Type.

Table 3. IDL Import/Export Keyword Support for MATLAB.

IDL Keyword	XML tag	MATLAB	Comment
struct	<struct>	MATLAB class	Filename will be <module>_<struct>.m Struct elements become properties of the class.
enum	<enum>	MATLAB class	Filename will be <module>_<enum>.m
const	<const>	N/A	If used to define size of a structure, value will be hard coded in the property dimensions.
typedef	<typedef>	N/A	All typedefs in the IDL file will be resolved to build-in MATLAB types, nested classes or enums. These resolved types will be used when defining class properties.
Module	<module>	Class method getIDModule().	Optional

7.1.6 Key Fields

7.1.6.1 Simulink

The DDS Blockset has support for key fields. DDS Topic Types are modeled in Simulink using the data class DDS.Bus and DDS.BusElement, where are derived from the built-in classes Simulink.Bus and Simulink.BusElement. The DDS.BusElement class has an additional property for holding key field information:

If a Topic has a key, DDS can use that information to determine which data object is being affected by your write operation. This allows DDS to implement QoS policies that properly manage the information maintained by the system. The DDS Blocks will use this key information when registering the DDS Topic Type. As a result, readers of data being published by a write block can implement code to register instances for each key. However, the DDS read block currently does not register separate instances for each key field value.

```
>> ShapeType.Elements(1)

ans =

  BusElement with properties:

             Key: 1
       Inherited: 0
        Optional: 0
            Name: 'color'
      Complexity: 'real'
      Dimensions: 128
        DataType: 'DDS_CharArray'
             Min: []
             Max: []
  DimensionsMode: 'Fixed'
      SampleTime: -1
            Unit: ''
     Description: ''
```

Figure 22. Key Field representation in a Simulink Bus.

7.1.6.2 MATLAB

MATLAB classes define key fields via a static method "getKeyFields" must be added to the class definition to identify key fields. Following is an example class for the RTI Shapes Demo. This file is in the <matlabroot>/toolbox/psp/examples/DDSBlockset directory and is used for the demo mldemo_RTIShapes.m The first field ("color") is a key field.

Editor - C:\work\DDS\toolbox\psp\examples\DDSBlockset\ShapeType.m

```
classdef ShapeType
    properties
        color     = 'PURPLE';    % default color
        x         = int32(0);
        y         = int32(0);
        shapesize = int32(0);
    end
    methods (Static = true)
        % return a vector of booleans indicating which fields are key
        % fields.
        function keyFields = getKeyFields()
            keyFields = [true, false, false, false];
        end
    end
end
```

Figure 23. Topic Type and Key Fields defined with MATLAB Class.

7.1.7 IDL “Module” Keyword

The IDL that is used to define DDS Topic Types can include the “module” keyword. This keyword is analogous to the C++ namespace keyword and it allows user to scope an IDL identifier. When a structure is defined within an IDL module, the resulting DDS fully qualified Topic Type name is: module::struct. For example, in the following IDL example, the DDS Topic Type name would be: top::middle::inner::Image

```
module top {
    module middle {
        module inner {
            struct Image {
            long width;
            long height;
            };
        };
    };
};
```

Figure 24. IDL Module Keyword.

The following sections describe how the IDL modules and the double colon operator are mapped to Simulink and MATLAB.

7.1.7.1 Simulink

Simulink does not support double colon operator, and also does not support package directories. Bus, aliases, parameters, and enumerations that are created in the base workspace will have the IDL module as a prefix. For the example, IDL file in the previous section, the DDS.Bus would be named ‘top_middle_inner_Image’.

The module information needs to be retained and used when registering the Topic Type with DDS. The module information is captured in the ‘Module” property of the data object as shown below.

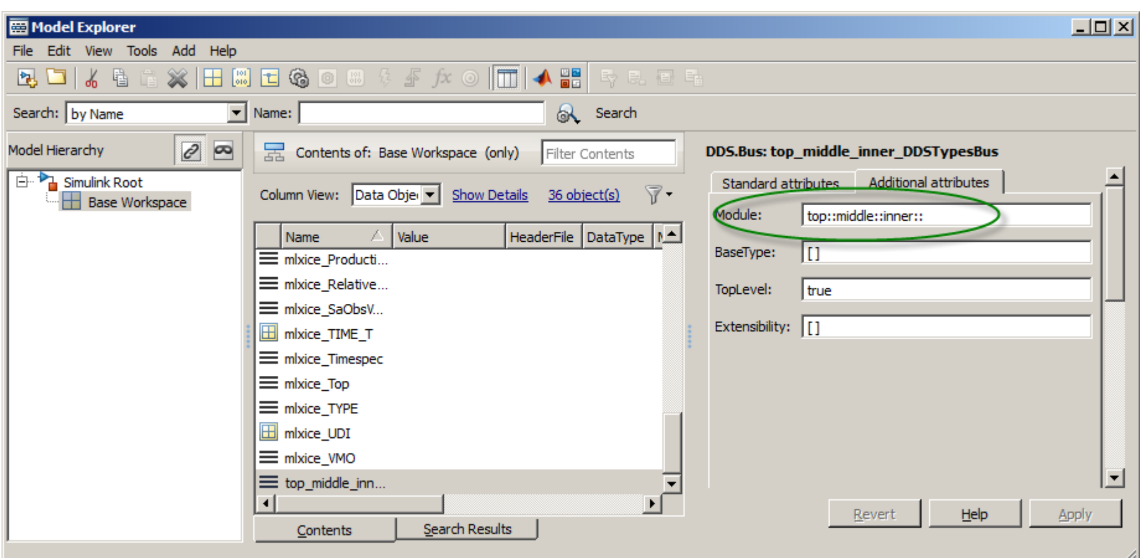


Figure 25. IDL Module Keyword in Simulink Bus.

7.1.7.2 MATLAB

MATLAB does not support double colon operator. MATLAB classes created for IDL structs and enumerations will have the IDL module as a prefix. For the example, IDL file in the previous section, the MATLAB class would be named "top_middle_inner_Image.m". The module information needs to be retained and used when registering the Topic Type with DDS. The information is returned by the "getIDLModule()" method in the MATLAB class. For example, an IDL struct named "ice::AbsoluteTime" would be imported as a MATLAB class named "ice_AbsoluteTime.m" as shown below.

7.1.7.3 Disabling Module Prefix

Some IDL files may have many nested <module> keywords. In these situations, it is possible that the resulting object name or class name exceeds the MATLAB limit of 63 characters due to the large number of module names prefixed to the identifier. If the identifiers are unique, this problem can be resolved by removing the prefixes from object names or class names.

The following MATLAB preference can be used to prevent <module> names from being used as prefixes.

```
setpref('DDSBlockset','ModulePrefix', ‘false’);
```

To restore default behavior in which the module names are used as prefixes:

```
setpref('DDSBlockset','ModulePrefix', ‘true’);
```

```
classdef ice_AbsoluteTime
    properties
        century = uint8(0);
        year = uint8(0);
        month = uint8(0);
        day = uint8(0);
        hour = uint8(0);
        minute = uint8(0);
        second = uint8(0);
        sec_fractions = uint8(0);
    end

    methods (Static = true)
        function module = getIDLModule()
            module = 'ice::';
        end
    end
end
```

Figure 26. IDL Module Keyword in MATLAB Class.

7.1.8 IDL Sequences

IDL supports the concept of a variable length vector, called a sequence. This section describes how to interact with Topic Types that utilize sequences.

7.1.8.1 Simulink

Sequence information for Simulink is captured as the "Dimensions Mode" property of a Simulink Bus Element as shown in Figure 28. Note that the Dimension field must be set to the maximum size of the sequence. Support for sequences is limited to Simulink support for variable sized signals. Specifically:

- Simulink does not allow a nested bus to be a sequence. It must have fixed length.
- Simulink does not allow an element in a nested array of buses to be a sequence.

Editor - C:\work\DDS\test\dSequence.idl

```
module ice {

  enum ConnectionMode {
    Serial,
    Simulated,
    Network
  };

  const short LENGTH_SIGNED = 32;
  const short LENGTH_UNSIGNED = 8;
  const short LENGTH_FLOAT    = 3;

  struct SequenceStruct {
  sequence<char,           LENGTH_SIGNED> s8;
  sequence<octet,          LENGTH_UNSIGNED> u8;
  sequence<short,          LENGTH_SIGNED> s16;
  sequence<unsigned short, LENGTH_UNSIGNED> u16;
  sequence<long,           LENGTH_SIGNED> s32;
```

Figure 27. IDL Sequence Example.

	Name	Value	DataType	Min	M
	ice_LENGTH_FLOAT	3	int16	[ ]	[
	ice_LENGTH_SIGNED	32	int16	[ ]	[
	ice_LENGTH_UNSIGNED	8	int16	[ ]	[
	ice_SequenceStruct				
	ice_SequenceStructNoEnum				
	ans		int8	[ ]	[

Name	Data/Bus Type	Complexity	Dimension	Minimum	Maximum	Dimensions Mode	Sample Time	Unit
s8	int8	real	32			Variable	-1	
u8	uint8	real	8			Variable	-1	
s16	int16	real	32			Variable	-1	
u16	uint16	real	8			Variable	-1	
s32	int32	real	32			Variable	-1	
u32	uint32	real	8			Variable	-1	
b8	boolean	real	4			Variable	-1	

Figure 28. IDL Sequence in a Simulink Bus.

7.1.8.2 MATLAB

An additional method getSequenceFields() must return true for each field that is a sequence. When specifying the size of the property in the MATLAB class, you must specify the maximum sequence length from the IDL file.

```
struct Latency {
    long sequence_number;
    sequence<octet, 8192> data;
};

classdef LatencyClass
    properties
        sequence_number = int32(0);
        data = uint8(zeros(8192,1));
    end
    methods (Static = true)
        % return a vector of booleans indicating which fields are sequences
        % fields.
        function sequenceFields = getSequenceFields()
            sequenceFields = boolean([false true]);  % second field is a sequence
        end
```

For the MATLAB class syntax, sequence lengths less than the maximum sequence length **are** supported.

To send a sequence length less than the maximum length using classes, follow these steps:

- Create an instance of the class
- Set the values for the sequence element.
- Write the class instance using the write() method.

Following is an example using the Latency class in Figure 34.

```
% create instance of the class
myData = Latency;
myData.sequenceNumber = int32(23);

% for this sample, length of sequence will be 4 rather than 8192
myData.data = uint8([ 1 2 3 4]);

% create publisher with Latency class/topic type
dp = DDS.DomainParticipant;
dp.addWriter('Latency','LatencyTopic');

% write the data
dp.Publishers(1).Writers(1).write(myData)
```

Figure 29. Example data writer using sequences.

Note that sequences of structures are also supported. Following is an example MATLAB class that has an element (struct_seq) which is itself a sequence of structures.

```
classdef ParentSeqClass
    % this class is for testing sequence support.  It has botha sequence of
    % base type ( uint8) and a sequence of structures ( ChildClass ).
    properties
        uint32_ID =  uint32(0); %@key
        struct_vec =  repmat(ChildSeqClass,1,8);
        struct_seq =  repmat(ChildSeqClass,1,6);
    end
    methods (Static = true)
        % return a vector of booleans indicating which fields are key
        % fields.
        function keyFields = getKeyFields()
            keyFields = [true, false, false];
        end

        function sequenceFields = getSequenceFields()
            sequenceFields = [false, false, true];
        end

    end
end
```

Max length of sequence of struct ChildSeqClass

element "struct_seq" is a sequence

Figure 30. MATLAB classes used to define sequences of IDL structures.

7.1.9 IDL Structure Inheritance

IDL has the concept inheritance in which a structure definition inherits structure elements from a base structure definition. For example, the IDL code in Figure 31 defines a struct StructA2 that inherits from StructB2.

Editor - C:\work\DDS\test\dA.idl*

```
struct StructB2
{
         unsigned short vIntA;
};
struct StructA2 : StructB2
{
        unsigned long  vMonthA2;
};

```

Figure 31. IDL Structure Inheritance.

This section describes how inheritance can be represented in Buses and MATLAB classes.

7.1.9.1 Simulink

A property in the DDS.Bus is used to indicate fields inherited from a base structure definition and the name of the base structure.

For the previous example, the DDS Bus definition in Figure 28 would be created. Note that:

1. The field vIntA from struct StructB2 have been merged into Bus StructA2
2. The metadata field “BaseType” for bus SructA2 references the base struct/bus StructB2
3. The field vIntA has the meta data “Inherited” set to true

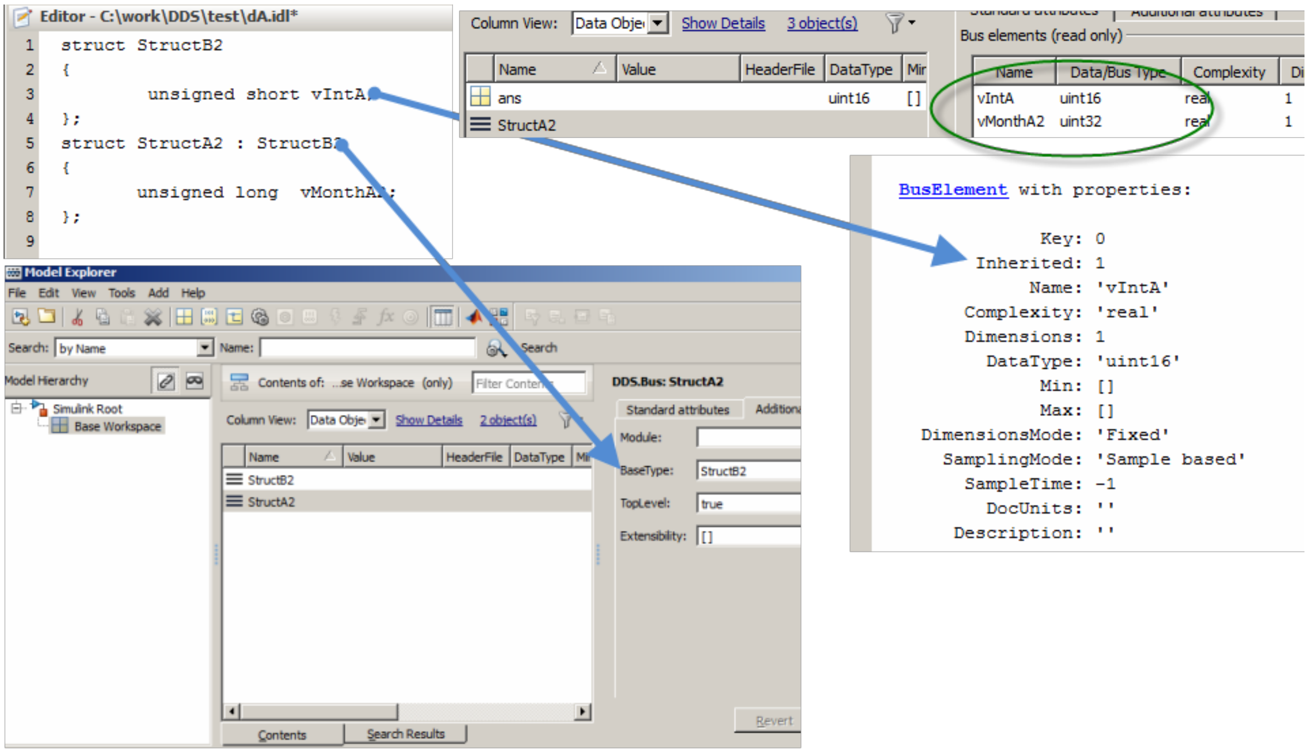


Figure 32. IDL Inheritance in Simulink.

7.1.9.2 MATLAB

It is highly recommended that the DDS.import() utility be used to create MATLAB classes from IDL to ensure the MATLAB classes are constructed properly. A MATLAB class will be created for each IDL structure. MATLAB class inheritance is used to represent IDL inheritance.

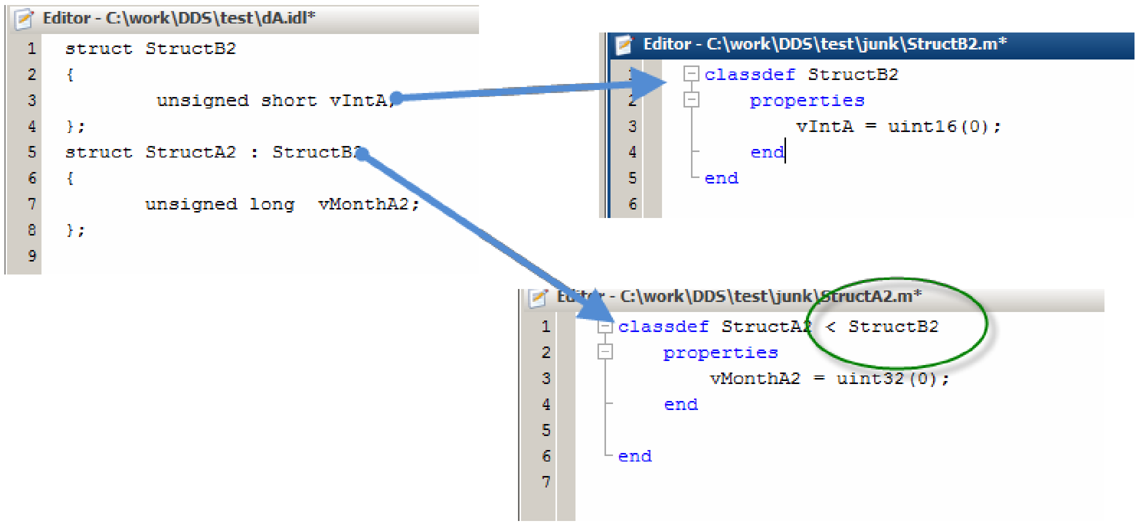


Figure 33. IDL Inheritance in MATLAB.

7.1.10 Simulink Data Dictionary

The DDS Blockset is compatible with the Simulink Data Dictionary. The following steps should be followed:

1. Import IDL information inth a Simulink Data Dictionary using the DDS.import command. See section 7.1.2.
2. Link the data dictionary to your Simulink model
   a. set_param(‘myModel’,’DataDictionary’,’myDD.sldd’);

7.2 Simulink Blocks

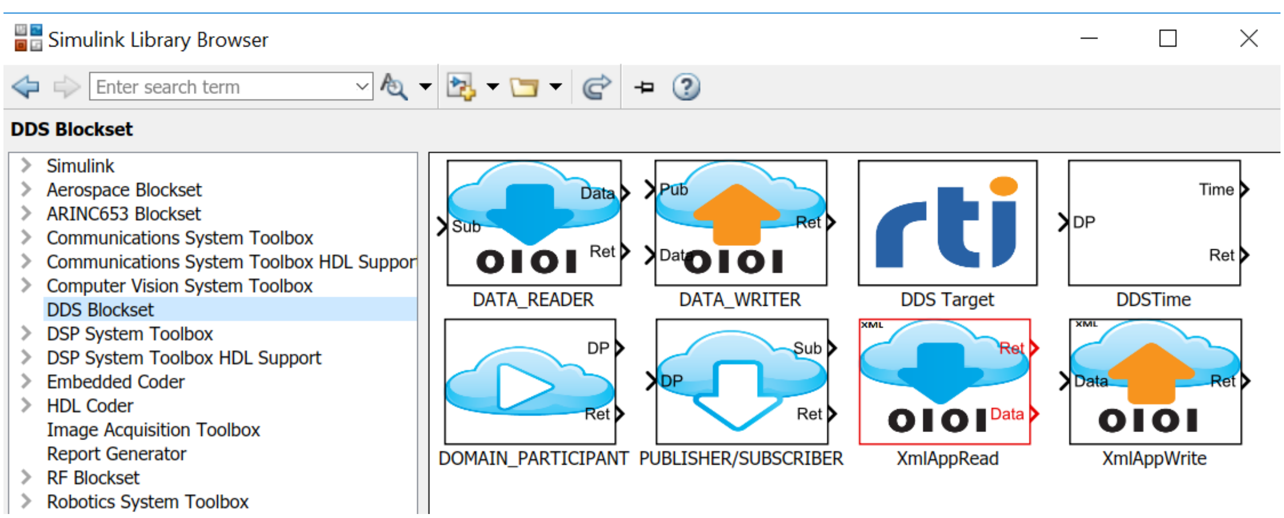


Figure 34. DDS Blockset.

7.2.1 Return codes

Most of the DDS blocks will have an output port indicating the status of the block. The following table defines possible values for this signal. The enumerated type DDS_RETCODE_TYPE is available to use in Simulink and MATLAB.

Table 4. DDS Return Codes.

Enumerated Type	Integer equivalent
DDS_RETCODE_OK	0
DDS_RETCODE_ERROR	1
DDS_RETCODE_UNSUPPORTED	2
DDS_RETCODE_BAD_PARAMETER	3
DDS_RETCODE_PRECONDITION_NOT_MET	4
DDS_RETCODE_OUT_OF_RESOURCES	5
DDS_RETCODE_NOT_ENABLED	6
DDS_RETCODE_IMMUTABLE_POLICY	7
DDS_RETCODE_INCONSISTENT_POLICY	8
DDS_RETCODE_ALREADY_DELETED	9
DDS_RETCODE_TIMEOUT	10
DDS_RETCODE_NO_DATA	11
DDS_RETCODE_ILLEGAL_OPERATION	12
DDS_RETCODE_NOT_ALLOWED_BY_SEC	13

7.2.2 DDS Target

This block controls the code generated for the DDS blocks in the Simulink model. There are three configuration items for this block:

- DDS Target: Controls which version of DDS the generated code will be compatible with.
  - RTI Connext DDS(default)
  - RTI Connext Micro DDS
- TypeSystem: Controls which type system the generated code will be compatible with. When DDS Target is set to Micro DDS, only static typing is supported.
  - Static(default)
  - Dynamic
- Discovery Mode: Controls the code generated for discovering other domain participants. Only used when DDS Target is Micro DDS.
  - Static
  - Dynamic
- IDL file: If the user has an IDL file for defining Topic Types used in the Simulink model, this file can be placed here. If present, then this IDL file will be used with rtiddsgen when TypeSystem == Static. If not provided, then Simulink will generate an IDL file for use with rtiddsgen.

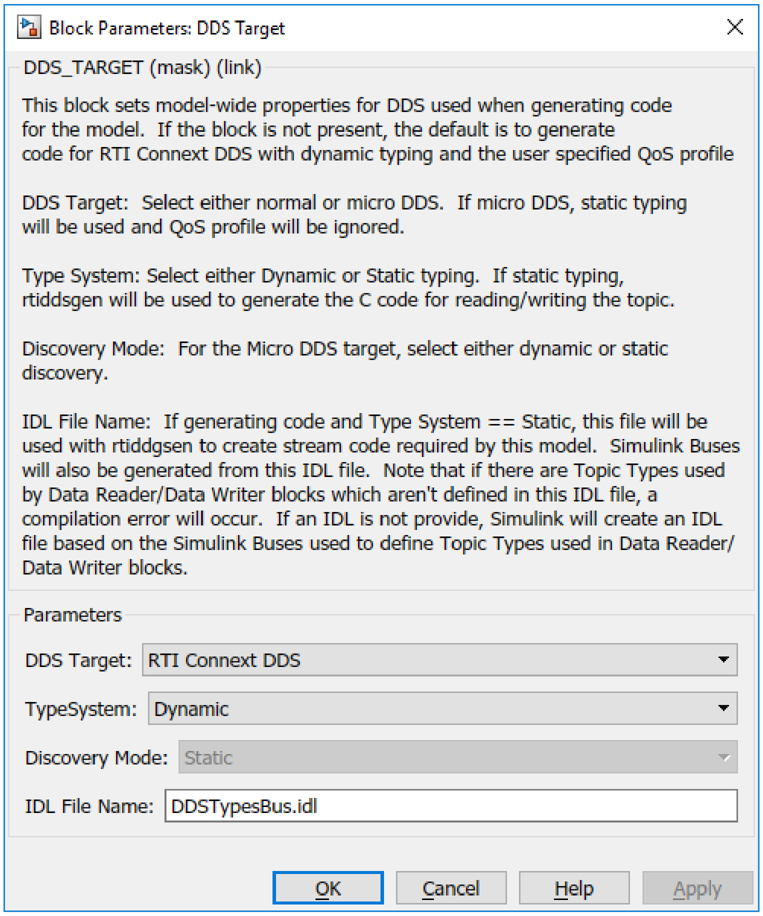


Figure 35. DDS Target Block Dialog.

7.2.3 Domain Participant

This block creates a DDS Domain Participant. Quality of Service settings are obtained from the user specified library and profile, contained on a QoS XML file.

- Outport 1 is the address of the created Domain Participant, or NULL if not successful.
- Outport 2 is the return code from the create_participant() DDS service. Refer to the RTI Connext DDS documentation for explanation of the return codes from the various Connext DDS functions.

RetCode	Description
1	If DDS_DomainParticipantFactory_create_participant_with_profile returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK

There are three configuration items for this block:

- QoS Profile: Specify a QoS profile name in the form of lib::profile or leave blank to use a default QoS. This link describes the rules RTI Connext DDS uses for locating and loading QoS profiles. Refer to section 10 for detailed information regarding QoS profiles and code generation. If left blank, the rules for locating a default QoS profile will be used. This link provides details.
- Domain ID: This is the Domain ID number that will be used when creating this domain participant.
- Sample Time: This has no effect on the operation of this block, as its logic executes during initialization to establish a domain participant. It should be left as -1 so that sample time is back-propagated from the data reader/writer block.

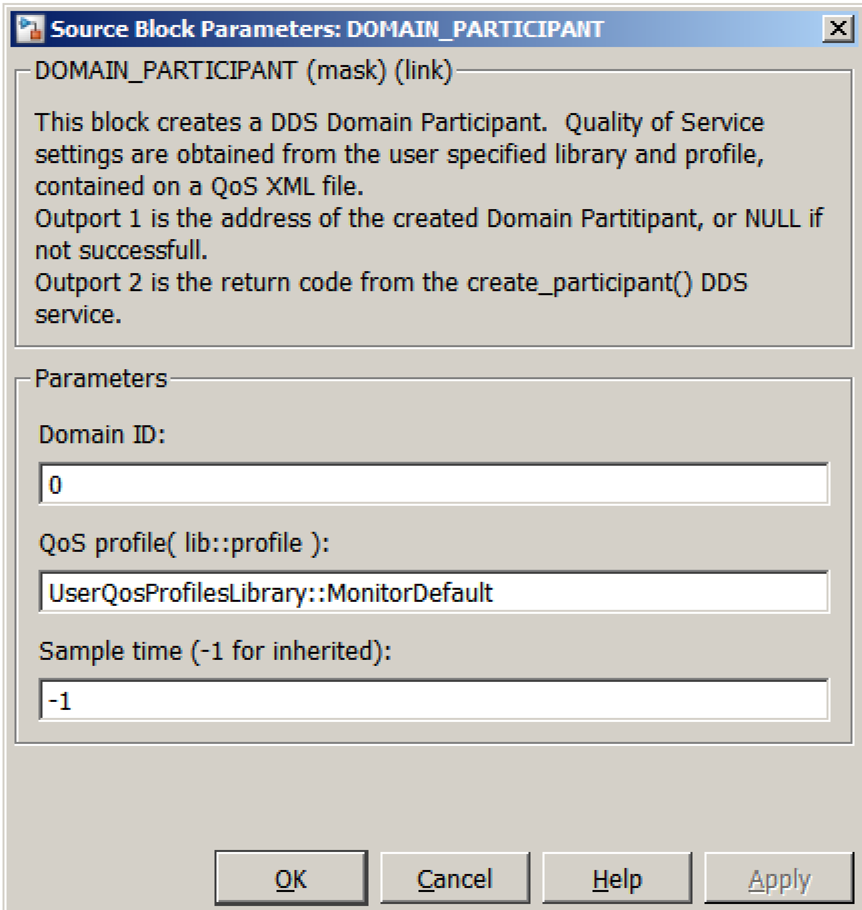


Figure 36. Domain Participant Block Dialog.

7.2.4 Publisher/Subscriber

This block can be configured as either a DDS Publisher or Subscriber.

- Input port 1 must be connected to a DOMAIN_PARTICIPANT block.
- Outport 1 should be connected to a DataWriter (if configured as a Publisher) or a DataReader (if connected to a Subscriber).
- Outport 2 contains a ReturnCode for each of the DDS service called by this block. Refer to the RTI Connext DDS documentation for explanation of the return codes from the various Connext DDS functions.

RetCode	Description
1	If DDS_DomainParticipant_create_subscriber_with_profile returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK

There are three configuration items for this block:

- Sub/Pub: Select either publisher or subscriber from the dropdown list.
- QoS Profile: Specify a QoS profile name in the form of lib::profile or leave blank to use a default QoS. This link describes the rules RTI Connext DDS uses for locating and loading QoS profiles. Refer to section 10 for detailed information regarding QoS profiles and code generation. If left blank, the rules for locating a default QoS profile will be used. This link provides details.

- Sample Time: This has no effect on the operation of this block, as its logic executes during initialization to establish a domain participant. It should be left as -1 so that sample time is back-propagated from the data reader/writer block.

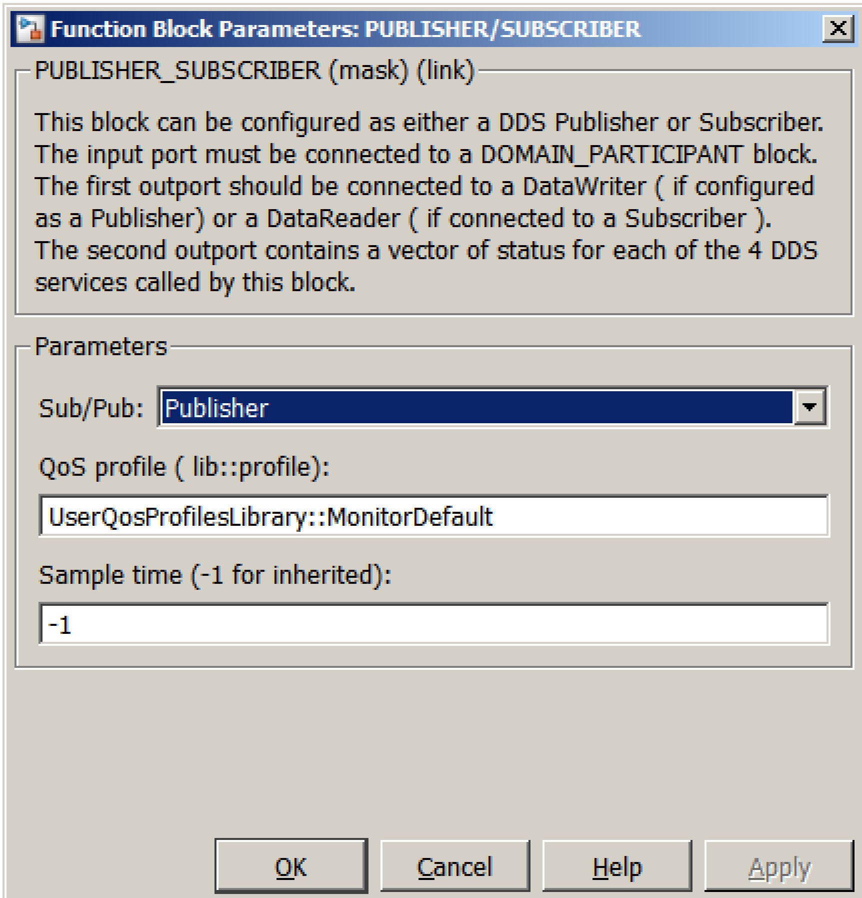


Figure 37. Publisher/Subscriber Block Dialog.

7.2.5 Data Writer

This block writes Topic data to DDS.

- Input port1 must be connected to a Publisher block.
- Input port 2 must be a bus signal of the same type as the Topic/Bus Object Name.
- Output port 1 contains a vector of size 11 that holds status information for each of the DDS services called to write data to DDS. Refer to the RTI Connext DDS documentation for explanation of the return codes from the various Connext DDS functions.

Table 5. DDS Return Codes for Data Writer.

RetCode Index	Description
1	If DDS_DynamicDataTypeSupport_new() returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK
2	Status returned by DDS_DynamicDataTypeSupport_register_type()
3	If DDS_DomainParticipant_create_topic_with_profile() returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK
4	Not used
5	Not used
6	Status returned by DDS_DomainParticipantFactory_get_datawriter_qos_from_profile
7	If DDS_Publisher_create_datawriter() returns NULL, DDS_RET_CODE_ERROR, else DDS_RETCODE_OK
8	Not used
9	Not used
10	If DDS_DynamicDataTypeSupport_create_data() returns NULL, DDS_RET_CODE_ERROR, else DDS_RETCODE_OK
11	Status returned by DDS_DynamicDataWriter_write()

There are four configuration items for this block.

- Topic Type/Bus Object Name: This is the name of the Simulink Bus object that is the data type for input port 2. This Bus Object name will be used for the DDS Topic Type when the Topic type is registered with DDS. Data readers wishing to read this topic data must use the same Topic Type/Topic name combination.
- Topic Name: This is the name that will be used, along with the Topic Type, when registering this Topic with DDS. Data readers wish to read this topic data must use the same Topic Type/Topic name combination.
- Sample Time: The sample time controls the rate at which this topic data will be written to DDS. If inherited, Simulink will use implicit rules for determining the sample time.
- QoS Profile: Specify a QoS profile name in the form of lib::profile or leave blank to use a default QoS. This link describes the rules RTI Connext DDS uses for locating and loading QoS profiles. Refer to section 10 for detailed information regarding QoS profiles and code generation. If left blank, the rules for locating a default QoS profile will be used. This link provides details.

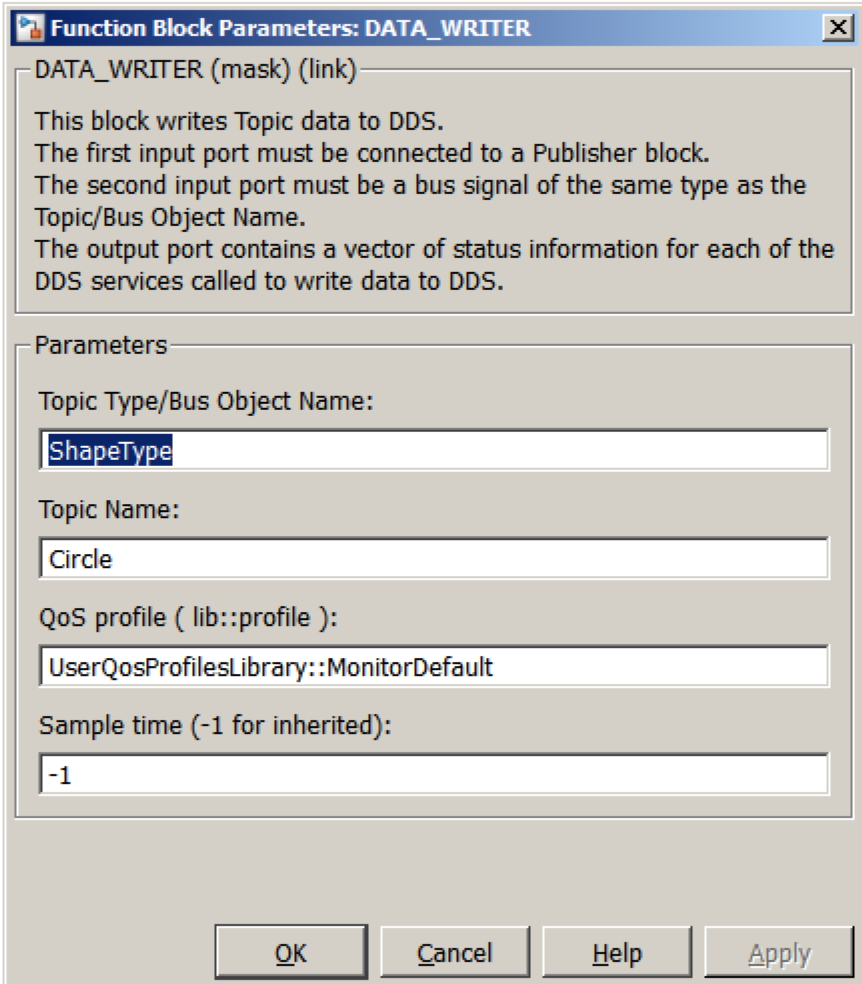


Figure 38. Data Writer Block Dialog.

7.2.6 Data Reader

This block reads data for the specified Topic/Bus Object Name/Topic Name.

- The first inport must be connected to a SUBSCRIBER block.
- The first outport contains the data read from DDS. The signal must a bus and the same type as the Topic/Bus Object Name.
- The second outport contains a vector of status information for each of the DDS services used to read the data. Refer to the RTI Connext DDS documentation for explanation of the return codes from the various Connext DDS functions.
- The optional third output contains the SampleInfo data structure. It contains a large amount of metadata provided by DDS for the received sample. Refer to RTI documentation for a detailed description of this data structure.

Table 6. Data Reader Return Codes.

RetCode Index	Description
1	If DDS_DynamicDataTypeSupport_new() returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK
2	Status returned by DDS_DynamicDataTypeSupport_register_type()
3	If DDS_DomainParticipant_create_topic_with_profile() returns NULL, DDS_RETCODE_ERROR else DDS_RETCODE_OK
4	Not used
5	Not used
6	Status returned by DDS_DomainParticipantFactory_get_datareader_qos_from_profile
7	If DDS_Publisher_create_datareader() returns NULL, DDS_RETCODE_ERROR, else DDS_RETCODE_OK
8	If WaitSet is enabled, then return value from DDS_WaitSet_wait. If Filter is used, and filter is not matched, return `DDS_RETCODE_NO_DATA,` else return DDS_RETCODE_OK.
9	If sampleInfo.valid_data = TRUE, status returned by DDS_DynamicDataReader_read()or DDS_DynamicDataReader_take(). If sampleInfo.valid_data = FALSE, DDS_RETCODE_NO_DATA
10	When configured for Topic filtering, if DDS_DomainParticipant_create_contentfilteredtopi_with_filter() returns NULL, DDS_RET_CODE_ERROR, else DDS_RETCODE_OK
11	Not used.

There are 11 configuration items for this block:

- Topic Type/Bus Object Name: This is the name of the Simulink Bus object that is the data type for input port 2. This Bus Object name will be used for the DDS Topic Type when the Topic type is registered with DDS. Data readers wishing to read this topic data must use the same Topic Type/Topic name combination.
- Topic Name: This is the name that will be used, along with the Topic Type, when registering this Topic with DDS. Data readers wish to read this topic data must use the same Topic Type/Topic name combination.
- QoS Profile: Specify a QoS profile name in the form of lib::profile or leave blank to use a default QoS. This link describes the rules RTI Connext DDS uses for locating and loading QoS profiles. Refer to section 10 for detailed information regarding QoS profiles and code generation. If left blank, the rules for locating a default QoS profile will be used. This link provides details.
- Sample Time: The sample time controls the rate at which this topic data will be written to DDS. If inherited, Simulink will use implicit rules for determining the sample time.
- Outports for Sample Info: When this box is checked a third outport will be added to output the SampleInfo for the received data sample. The definition of the fields in the SampleInfo bus can be found here.
- Waitset: If this box is checked, a waitset will be used to wait for the next available data. When not checked, the block will poll DDS for available data. If not data is available, the block will return `DDS_RETCODE_NO_DATA` . If waitset is enabled, a read condition will be created with the following settings. The Simulink simulation will be blocked until data is received or a timeout occurs. If the waitset timesout, the block will return `DDS_RETCODE_TIMEOUT`.
  - `DDS_NOT_READ_SAMPLE_STATE`
  - `DDS_ANY_VIEW_STATE`
  - `DDS_ANY_INSTANCE_STATE`

- If a waitset is enabled, this is the timeout used. Otherwise, this parameter is ignored.
- Read()/Take(): Select read() or take() for obtaining the DDS data. Read() will leave the DDS data in DDS memory. Take() will remove the data from DDS memory.
- FilterType: Select from this list to enable Content Topic Filtering. Select from: No Filter, DDS_SQLFILTER_NAME or DDS_STRINGMATCHFILTER_NAME.
- FilterExpression: SQL filter expression. Refer to the DDS User's Manual for a complete description of the SQL expression syntax.
- FilterParameters: If any parameters are used in the filter expression (i.e. %0, %1, etc.), then you must provide a cell array of strings, one for each parameter in the filter expression. Literal constants must be in the form of a string (i.e. '23'). Strings must inside single quotes (i.e. ' ' 'PURPLE' ' '). Workspace variables can be used. In this case, the workspace variable must be a string (i.e. x = '35' myColor = ' ' 'GREEN' ' ').

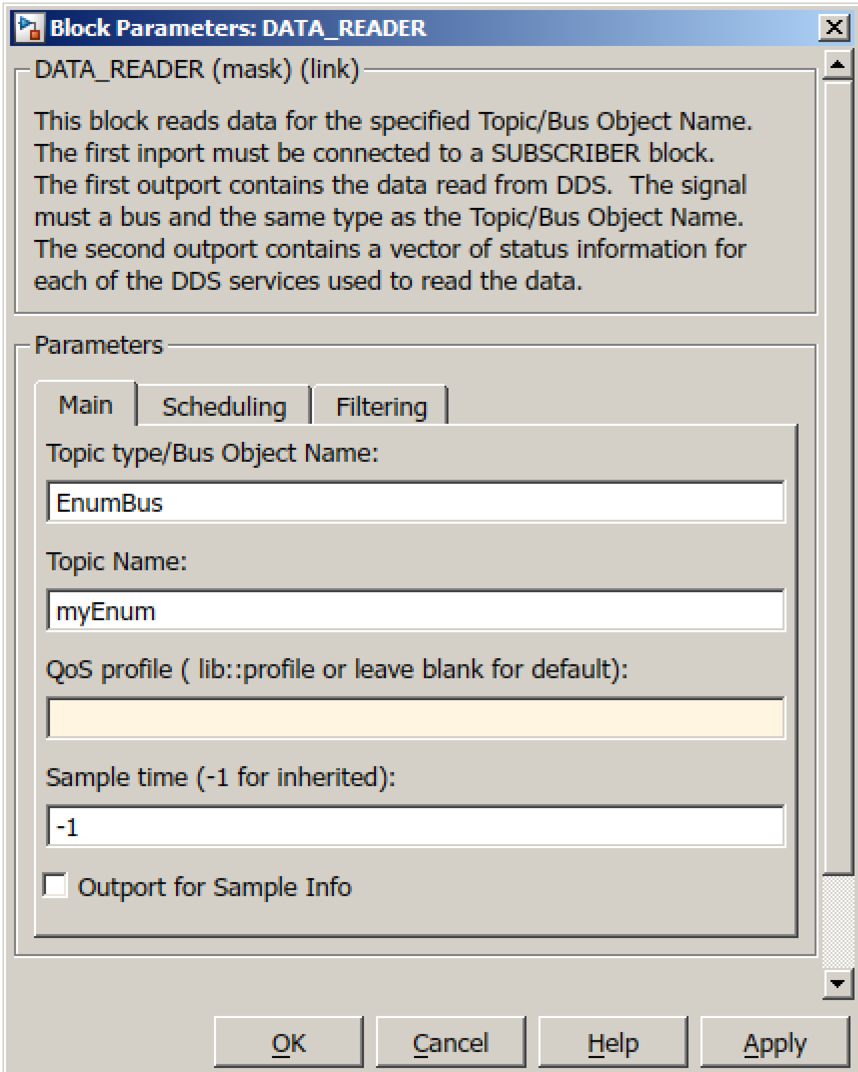
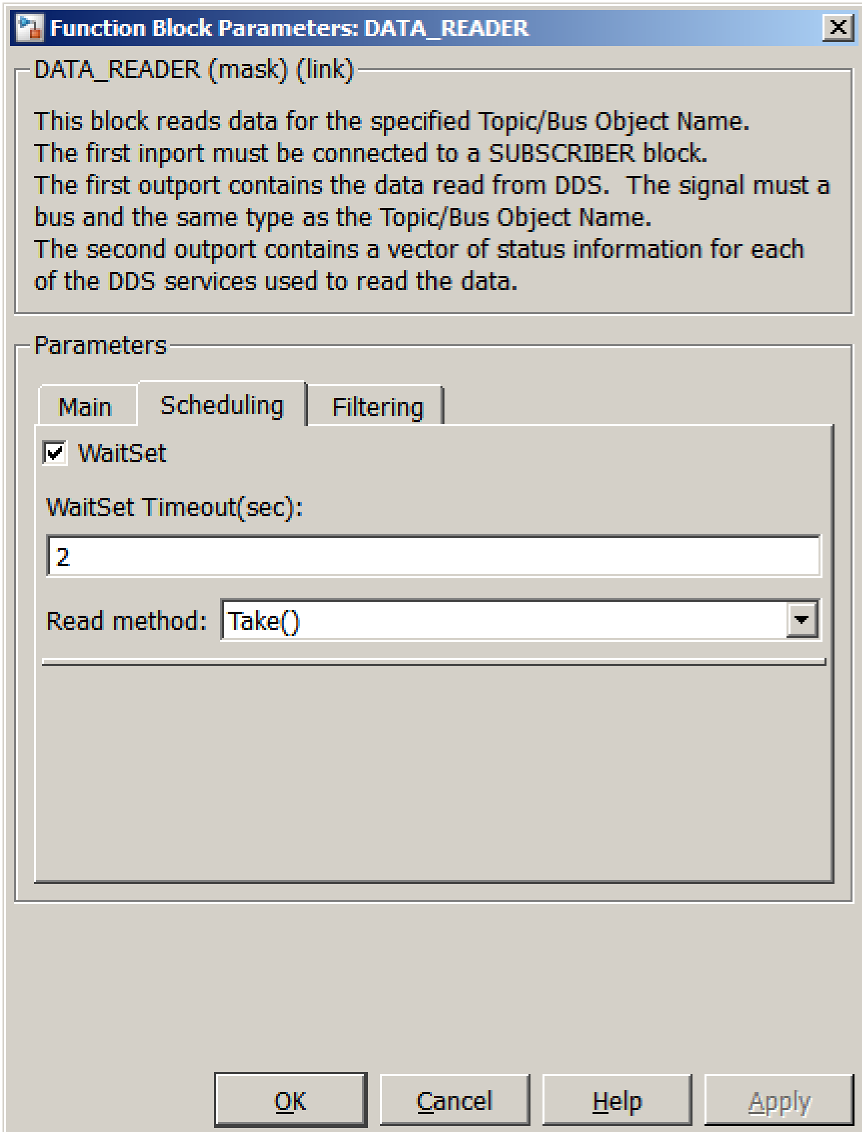


Figure 39. Data Reader Block Dialog.


Function Block Parameters: DATA_READER
DATA_READER (mask) (link)
This block reads data for the specified Topic/Bus Object Name.
The first inport must be connected to a SUBSCRIBER block.
The first outport contains the data read from DDS. The signal must a bus and the same type as the Topic/Bus Object Name.
The second outport contains a vector of status information for each of the DDS services used to read the data.
Parameters
Main
Scheduling
Filtering
WaitSet
WaitSet Timeout(sec):
2
Read method:
Take()
OK
Cancel
Help
Apply

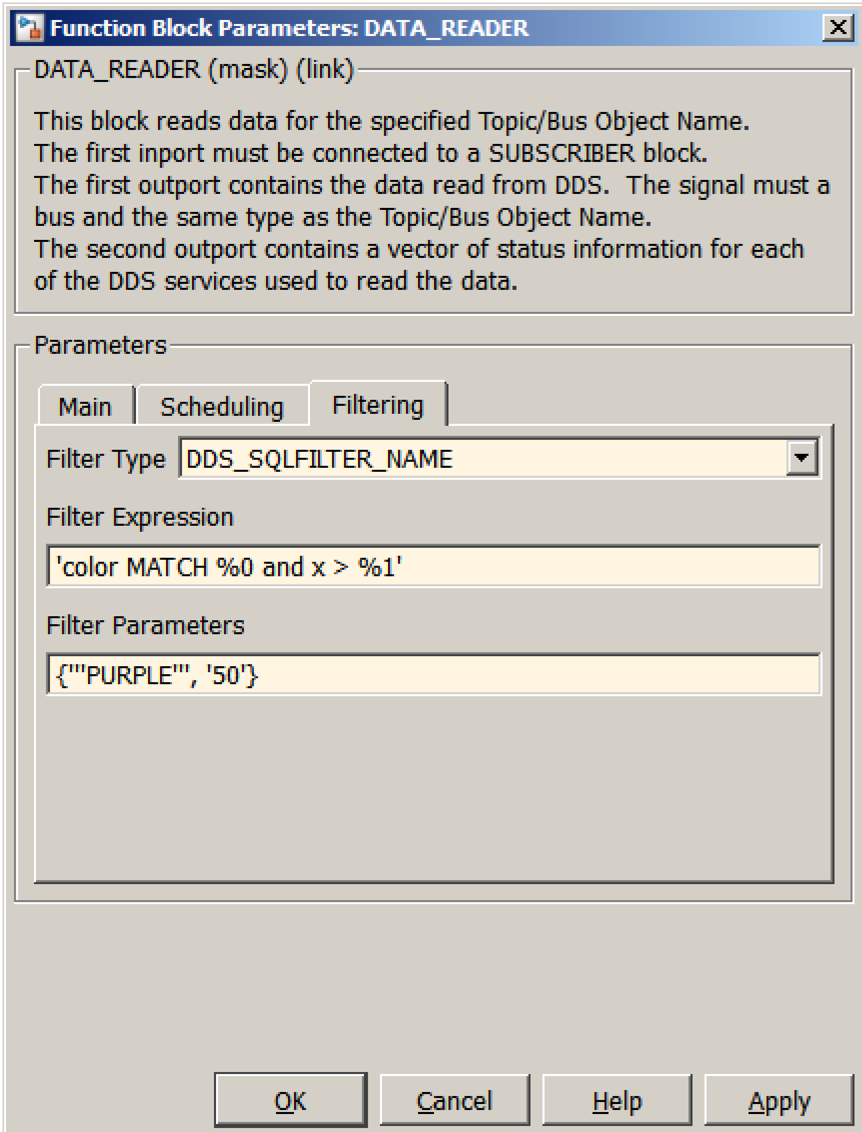


Figure 40. Data Reader Block Dialog - Filtering.

7.2.7 DDSTime

This block returns the current system time from DDS.

- Outport 1 is a Simulink Bus of type DDS_Time_t. The first field is int32 seconds. The second field is unit32 nanoseconds.
- Outport 2 is the return code with status of the DDS service DDS_DomainParticipant_get_current_time().

7.2.8 XML Application Creation Read

This block uses the XML Application creation capabilities of DDS to read/take a DDS sample. When using this approach, a single block replaces the traditional Domain Participant/Subscriber/Data Reader blocks. Refer to 7.5 for a detailed description of the XML Application Simulink blocks and associated XML configuration file.

There are 2 or 3 output ports for this block.

- The first outport contains the status of the read/take operation. Refer to Table 4 for a description of this output port.
- The second outport contains the data read from DDS. The signal is a bus whose datatype will be obtained from the XML configuration file based on the values entered in the block dialog.
- The optional third output contains the SampleInfo data structure. It contains a large amount of metadata provided by DDS for the received sample. Refer to RTI documentation for a detailed description of this data structure.

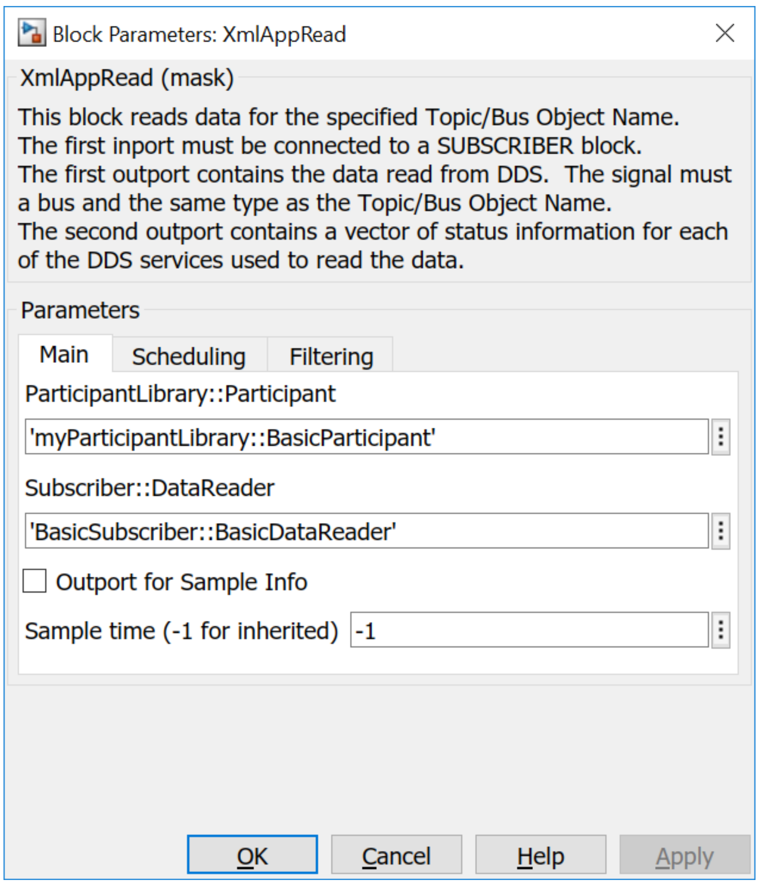


Figure 41. XML Application Create Data Reader.

There are 10 configuration items for this block:

- ParticpantLibrary::Particpant: The particpant library and particpant tags from the XML file used to define this particular XML Application Create Read block.
- Subscriber::DataReader: The subscriber and data reader tags from the XML file used to define this data reader.
- Sample Time: The sample time controls the rate at which this topic data will be written to DDS. If inherited, Simulink will use implicit rules for determining the sample time.
- Outports for Sample Info: When this box is checked a third outport will be added to output the SampleInfo for the received data sample. The definition of the fields in the SampleInfo bus can be found here.
- Waitset: If this box is checked, a waitset will be used to wait for the next

available data. When not checked, the block will poll DDS for available data. If not data is available, the block will return `DDS_RETCODE_NO_DATA` . If waitset is enabled, a read condition will be created with the following settings. The Simulink simulation will be blocked until data is received or a timeout occurs. If the waitset timesout, the block will return `DDS_RETCODE_TIMEOUT`.

   - `DDS_NOT_READ_SAMPLE_STATE`
   - `DDS_ANY_VIEW_STATE`
   - `DDS_ANY_INSTANCE_STATE`

- If a waitset is enabled, this is the timeout used. Otherwise, this parameter is ignored.
- Read()/Take(): Select read() or take() for obtaining the DDS data. Read() will leave the DDS data in DDS memory. Take() will remove the data from DDS memory.
- FilterType: Select from this list to enable Content Topic Filtering. Select from: No Filter, DDS_SQLFILTER_NAME or DDS_STRINGMATCHFILTER_NAME.
- FilterExpression: SQL filter expression. Refer to the DDS User's Manual for a complete description of the SQL expression syntax.
- FilterParameters: If any parameters are used in the filter expression (i.e. %0, %1, etc.), then you must provide a cell array of strings, one for each parameter in the filter expression. Literal constants must be in the form of a string (i.e. '23'). Strings must inside single quotes (i.e. ' ' 'PURPLE' ' '). Workspace variables can be used. In this case, the workspace variable must be a string (i.e. x = '35' myColor = ' ' 'GREEN' ' ').

7.2.9 XML App Creation Write

This block uses the XML Application creation capabilities of DDS to write a DDS sample. When using this approach, a single block replaces the traditional Domain Participant/Publisher/Data Writer blocks. Refer to 7.5 for a detailed description of the XML Application Simulink blocks and associated XML configuration file.

The block has 1 input port and 1 output port.

- Input port 1 must be a bus signal containing the data to be written. The type of this port must match the type defined in the XML file and referenced by the block dialog parameters.
- The first outport contains the status of the write operation. Refer to Table 4 for a description of this output port.

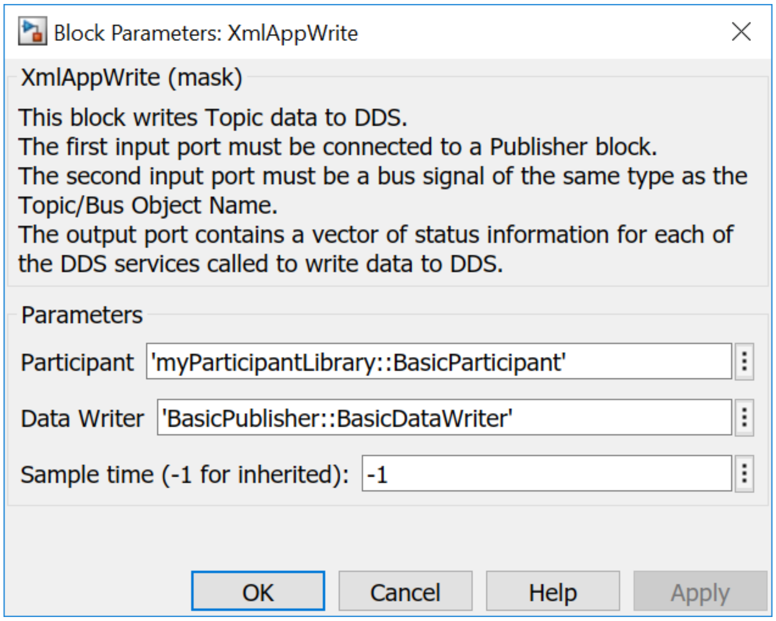


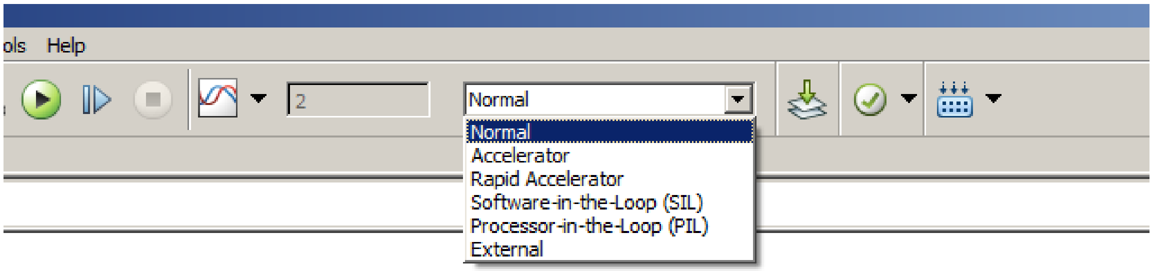
Figure 42. XML Application Create Data Writer.

There are three configuration items for this block.

- ParticpantLibrary::Particpant: The particpant library and particpant tags from the XML file used to define this particular XML Application Create Read block.
- Publisher::DataWriter: The publisher and data writer tags from the XML file used to define this data reader.
- Sample Time: The sample time controls the rate at which this topic data will be written to DDS. If inherited, Simulink will use implicit rules for determining the sample time.

7.3 Simulating with Accelerator Modes

When a Simulink model is simulated, the model can be configured to execute in one of six different simulation modes.



For the Accelerator and Rapid Accelerator modes, Simulink converts the Simulink model to C code and compiles that code into a MEX file. The simulation executes the compiled C code to achieve improved performance.

7.3.1 Accelerator Mode

Since code will be generated, compile and linked with the RTI DDS libraries, a model configuration parameter must be set for proper compilation.

Windows:
```
set_param(<model>,'AccelMakeCommand', 'make_rtw MEX_OPTS="-DRTI_WIN32"')
```

Linux and MacOS:
```
set_param(<model>,'AccelMakeCommand', 'make_rtw MEX_OPTS="-DRTI_UNIX"')
```

7.3.2 Rapid Accelerator Mode

There is not a similar configuration setting as described in the previous section for Rapid Accelerator mode. In order to run in Rapid Accelerator mode on Linux computers, the user will need to modify the make file and rebuild the Accelerator mode .mex file. The make file is located in : /slprj/raccel/<model>. Edit the file <model>.mk and add the –DRTI_UNIX macro definition to the following line.

```
GEN_SAMPLE_MAIN      = 0

OPTIMIZATION_FLAGS   = /Od /Oy- /DNDEBUG /DRTI_UNIX
ADDITIONAL_LDFLAGS   =

RACCEL_PARALLEL_EXECUTION = 0
```

Once this is done, rerun the make file by typing the following at the MATLAB prompt:
system('gmake –f <model>.mk')

This will rebuild the accelerated model MEX file. You can now return to the Simulink model window and push the "run" button to run the Accelerated mode simulation.

7.4 Code Generation from Simulink Models

This section provides additional information on the code generated from Simulink models containing DDS Blocks

7.4.1 Quality of Service

When the DDS Target is set to RTI Connext DDS, the generated code will use the same QoS profile specified in the DDS Block mask and used during simulation. RTI Connext DDS Micro does not support QoS profiles. As a result, the QoS profile specified in the Block dialog will be ignored. Rather, the generated code will have hard-code QoS settings, along with a preprocessor macro that the user can define to over-ride the default settings, if desired. Following is an example code fragment for a Domain participant block:

```
#if defined(SIMULINK_DOMAIN_PARTICIPANT_0_QOS)
      SIMULINK_DOMAIN_PARTICIPANT_0_QOS
#else
      /* use default Qos  */
      struct DDS_DomainParticipantQos dp_qos =
        DDS_DomainParticipantQos_INITIALIZER;
      char* peer = "127.0.0.1";        /* default to loopback */
      OSAPI_Stdio_snprintf(dp_qos.discovery.discovery.name,8,"MATLAB");
      DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers,1);
      DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers,1);
      *DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers,0) =
              DDS_String_dup(peer);
      dp_qos.resource_limits.max_destination_ports = 32;
      dp_qos.resource_limits.max_receive_ports = 32;
      dp_qos.resource_limits.local_topic_allocation = 2;
      dp_qos.resource_limits.local_type_allocation = 2;
      dp_qos.resource_limits.local_reader_allocation = 1;
      dp_qos.resource_limits.local_writer_allocation = 1;
      dp_qos.resource_limits.remote_participant_allocation = 8;
      dp_qos.resource_limits.remote_reader_allocation = 8;
      dp_qos.resource_limits.remote_writer_allocation = 8;
#endif
      mEnum_B.DOMAIN_PARTICIPANT = DDS_DomainParticipantFactory_create_participant
        (DDS_TheParticipantFactory, 0, &dp_qos, NULL, DDS_STATUS_MASK_NONE);
    }
```

Macro to over-ride default QoS

Default QoS

Figure 43. DDS Connect Micro DDS QoS Example.

7.4.2 DDS Type System

The Simulink blockset supports two options for generating the code that registers the DDS Topic Types: Static and Dynamic. For the Static type system, the rtiddsgen utility is used to generate C code to statically define Topic Type and C code to read/write samples between the DDS memory and the applications memory.

With the Dynamic type system, the DynamicData APIs are used to register Topic Types and send/receive sample data. Applications that use the DynmaicData API will be slower than the Static type system.

7.5 XML Application Creation

RTI Connext DDS Professional includes a set of APIs that allow an application to create DDS entities (participants, subscribers, publishers, readers, writers) from an XML file. This is accomplished by adding additional information into the XML file that previously contained QoS information. Figure 44 contains an example Application Creation XML file. The highlighted tags from the XML file are entered into the dialog box for the XMLAppWrite Simulink block shown in Figure 45.

```xml
    <types>
        <struct name="BasicType">
            <member name="x" type="int32"/>
            <member name="y" type="int32"/>
        </struct>
    </types>

    <domain_library name="MathWorks">
        <domain name="BasicDomain" domain_id="0">
            <register_type name="BasicTypeRegistered" type_ref="BasicType"/>
            <topic name="BasicTopicName" register_type_ref="BasicTypeRegistered"/>
        </domain>
    </domain_library>

    <domain_participant_library name="myParticipantLibrary">
        <domain_participant name="BasicParticipant" domain_ref="MathWorks::BasicDomain">
            <publisher name="BasicPublisher">
                <data_writer name="BasicDataWriter" topic_ref="BasicTopicName"/>
            </publisher>
            <subscriber name="BasicSubscriber">
                <data_reader name="BasicDataReader" topic_ref="BasicTopicName">
                </data_reader>
            </subscriber>
        </domain_participant>
    </domain_participant_library>
```

Figure 44. Example Application Creation XML File.

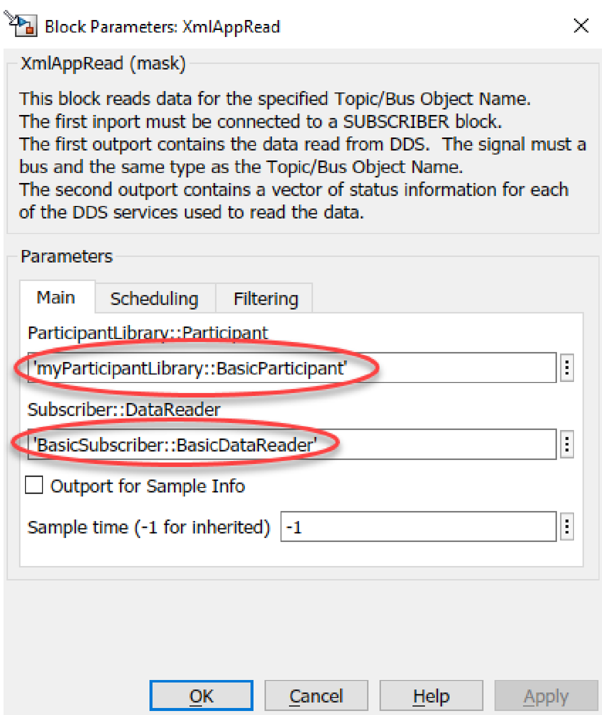


Figure 45. XML Application Create Data Reader Block Dialog.

The XML file can be created with a text editor, or it can be created using the RTI System Designer. Refer to rti.com for more details on RTI System Designer. Figure 46 shows RTI System Designer being used to edit the XML file shown in Figure 44.

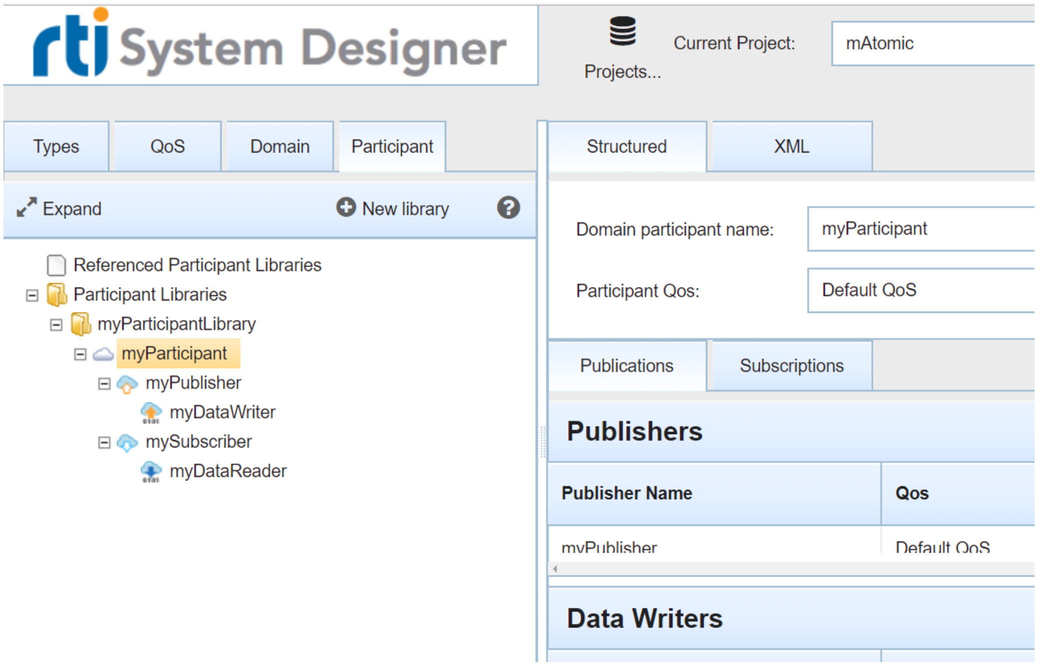


Figure 46. RTI System Designer.

Note that the XML file also contains data type information. When using XML Application Creation blocks, an IDL file is no longer required. The following utility can be used to convert an IDL file to equivalent XML format.

```
DDS.Utilities.convertIDLtoXML(<myIDL.idl>)
```

RTI DDS uses the same rules used for QoS XML files to locate and load the XML file. For example, the content shown in Figure 44 can be placed in USER_QOS_PROFILES.xml in the current MATLAB working directory.

Once the XML file has been created, the DDS.import() utility must be used to create buses in the MATLAB workspace/Simulink Data Dictionary that will be used by your Simulink model. Simply call the DDS.import() utility for the XML file that contains your XMP Application Creation information. For example, of you have placed your XML code into USER_QOS_PROFILES.xml, then the following command will create corresponding buses/enums.

```
DDS.Import('USER_QOS_PROFILES.xml')
```

8 MATLAB Toolbox

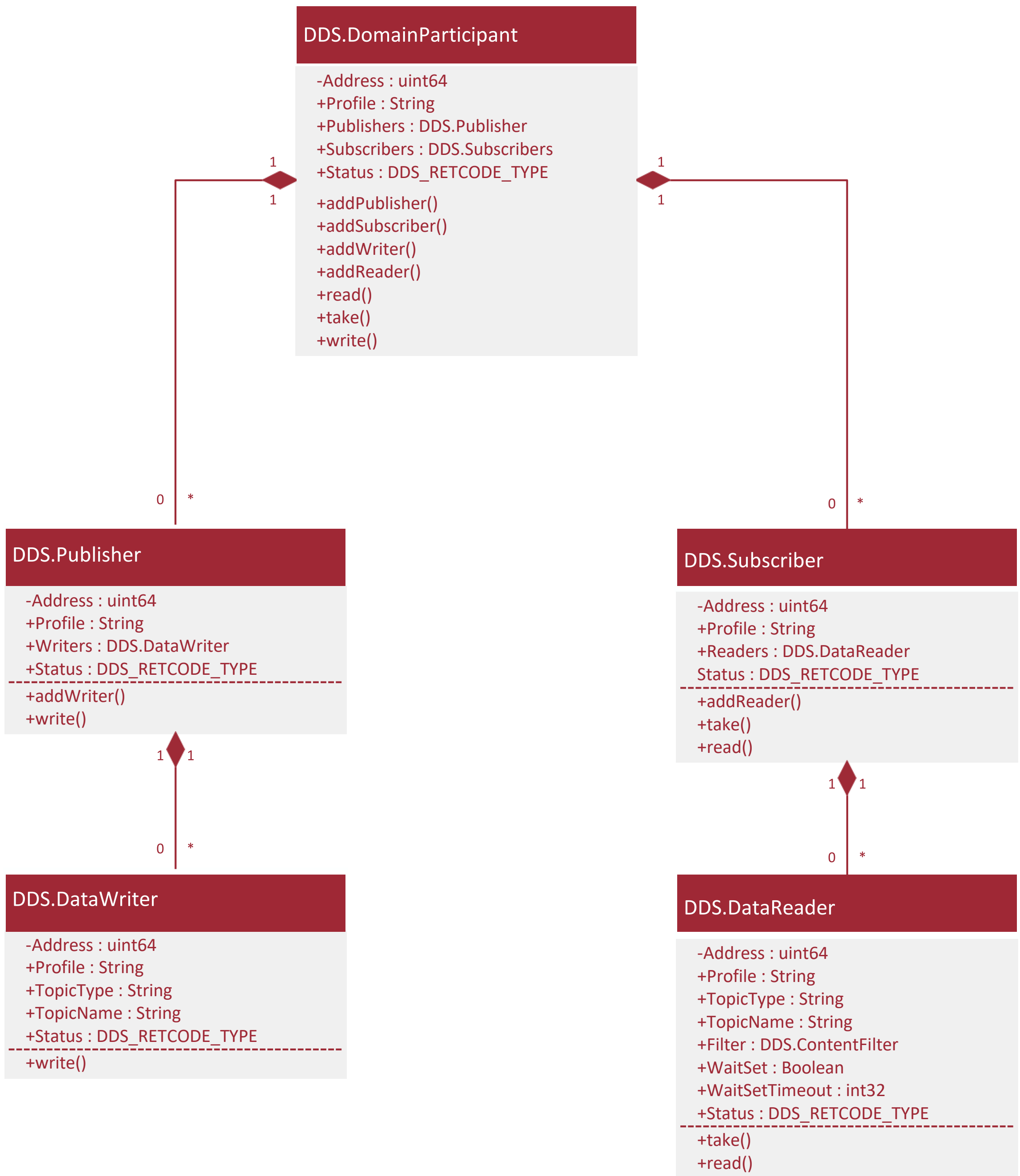
Refer to 7.1.3 for a detailed description of how to define Topic Types for the MATLAB toolbox.

Refer to the DDS Blockset documentation in MATLAB for a detailed description of each class and method

8.1 DDS Functions

DDS.discoveryMonitor	Return a list of particpants, publishers and subscribers for a given domainID
DDS.export	Export a Simulink Bus to an IDL file.
DDS.getProfiles	Returns a list of all available QoS profiles.
DDS.import	Import IDL file into Simulink Bus or MATLAB or Class
DDS.rtiddsgen	Generate C code for the provided IDL file using RTI DDS Connext rtiddsgen
DDS.version	Return version of RTI Connext DDS and micro DDS

8.2 DDS Classes



8.3 MATLAB Performance

By default, the DDS MATLAB functions uses class instances to send/receive sample data. However, MATLAB handles structures more efficiently than MATLAB classes. If you will be sending/receiving samples at a high rate, it is recommended that you convert class instances to structures before sending data and that you provide a preallocated structure for receiving data. Use the function DDS.Utilities.toStruct to convert a DDS class instance to a struct. Following is an example:

```
>DDS.import('ShapeType.idl','matlab');
>myShape = ShapeType;      % create an instance of ShapeType class
>myShape.x = int32(10);
>myShape.y = int32(20);
>myShape.shapesize = int32(25);
>myShape.color = 'RED';
>myShapeStruct = DDS.Utilities.toStruct(myShape);
>
>dp = DDS.DomainParticipant;
>dp.addWriter('ShapeType','Circle');
>dp.write(myShapeStruct)
>
>dp.addReader('ShapeType','Triangle');
>sampleStruct = dp.read(myShapeStruct);
```

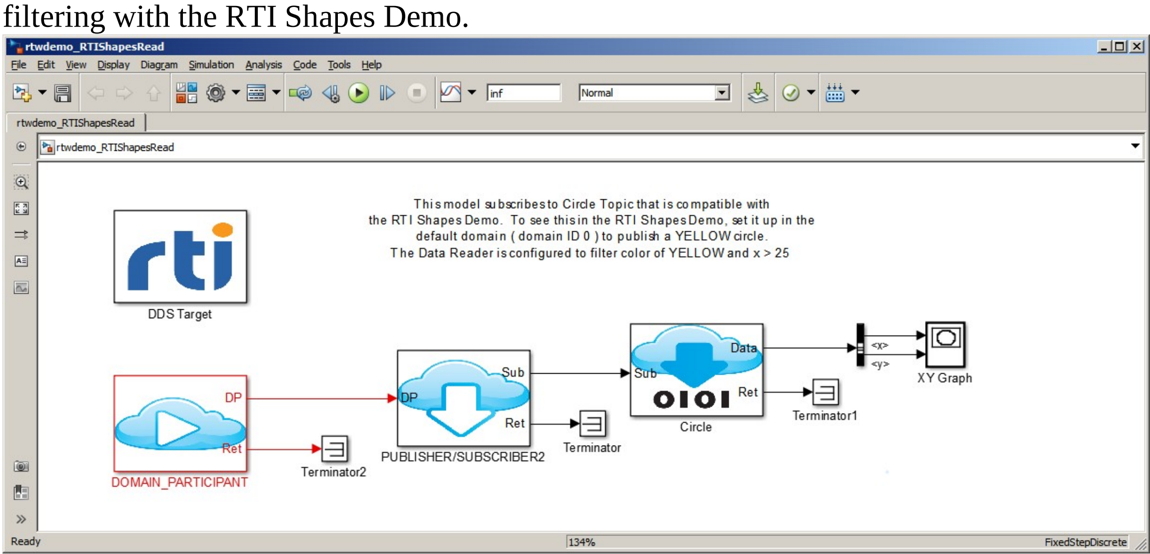
9 Topic Content Filtering

Both the Simulink blockset and MATLAB toolbox support content filtering on topic data. In both cases, the filter is defined and applied to the data reader.

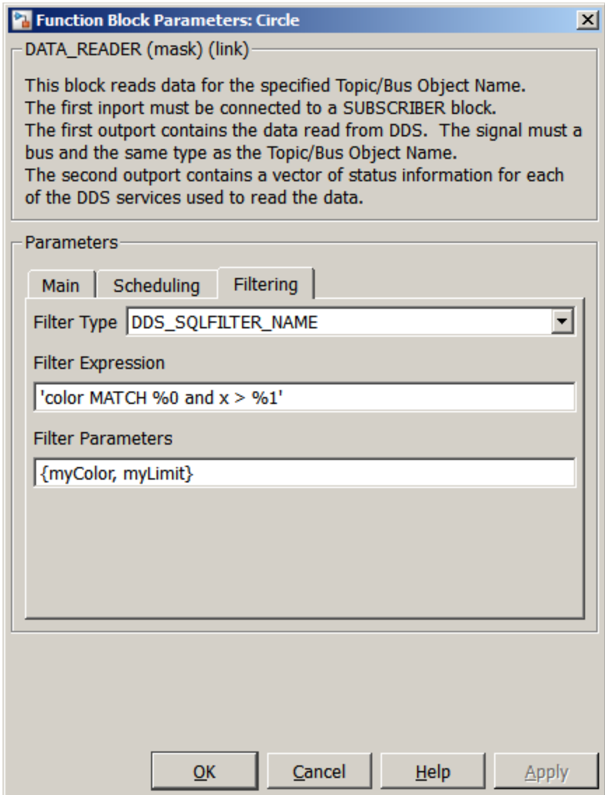
9.1 Simulink

Filtering for the Data Read block is available on the "Filtering" tab of the block dialog. Refer to section 7.2.5 for details of the block.

The example model rtwdemo_RTIShapesRead that is part of the PSP demonstrates topic filtering with the RTI Shapes Demo.



In this example, a topic filter is defined.



9.2 MATLAB

Filtering for the MATLAB DDS is only supported when using MATLAB classes to define DDS Topic Types. Before creating a Data Reader, the user must first create and initialize the topic filter using the DDS.contentFilter class. In the following example, a content filter is created to look for all YELLOW shapes with x position > 25.

```
>> myFilter = DDS.contentFilter;
>> myFilter.FilterExpression = 'color MATCH %0 and x > %1';
>> myFilter.FilterParameters = {'''YELLOW''', '25'};
>> myFilter

myFilter =

  contentFilter with properties:

          FilterType: 'DDS_SQLFILTER_NAME'
    FilterExpression: 'color MATCH %0 and x > %1'
    FilterParameters: {''YELLOW''  '25'}
```

Default

Once the filter is defined, a Data Read is created with this newly created filter as follows:

```
>> dp = DDS.DomainParticipant;
>> dp.addReader('ShapeType', 'myShape','UserQosProfilesLibrary::MonitorDefault', ...
[], true, 3, ReadMethodType.TAKE, [], Filter );
```

10 Quality of Service (QoS)

DDS provides a significant amount of configurability for DDS operation via Quality of Service parameters. To simplify the DDS Blockset/Toolbox, QoS profiles are used for configuring QoS parameters for Simulink and MATLAB. Simulink and MATLAB follow the RTI DDS Connext rules for file name and path search rules to locate QoS profile libraries. The DDS Blockset provides a function for determining the list of available profiles.

- DDS.getProfiles – returns a list of available QoS profiles. Refer to RTI Connext DDS documentation for rules on defining and naming profile libraries.

The code generated from a Simulink model for a DDS block will have one of two forms, based on whether or not a QoS Profile is specified. For example, QoS profile myLibrary::myProfile is specified for a Domain Participant, the generated code will look like this:

```
DDS_DomainParticipantFactory_create_participant_with_profile(
      DDS_TheParticipantFactory,
      23,
      "myLibrary",
      "myProfile",
      NULL,
      DDS_STATUS_MASK_NONE);
```

If the QoS Profile is left blank, the code is generated to use a default QoS profile:

```
DDS_DomainParticipantFactory_create_participant(
      DDS_TheParticipantFactory,
      23,
      &DDS_PARTICIPANT_QOS_DEFAULT,
      NULL,
      DDS_STATUS_MASK_NONE);
```

This link explains how the values for a default QoS profile are determined.

11 Limitations

This section describes known limitations with the DDS Blockset/Toolbox.

11.1 Installation

- Ubuntu. The installer is built with InstallJammer. There is a known issue when running installers built with InstallJammer on Unbuntu version 14.04 and newer. An error message similar to the following will occur:

```
./DDSBlockset-R2016a_v2.7.0_glnxa64-Install
unknown user id: 11114
   while executing
"id user"
   (procedure "::InstallJammer::CommonInit" line 223)
   invoked from within
"::InstallJammer::CommonInit"
   (procedure "::InstallJammer::InitInstall" line 19)
```

```
    invoked from within
"::InstallJammer::InitInstall"
    (file "/installkitvfs/main2.tcl" line 27120)
    invoked from within
"source [file join $::installkit::root main2.tcl]"
    (file "/installkitvfs/main.tcl" line 3)
```

The workaround for this error is to run the installer from a command shell window with the sudo command as follows:

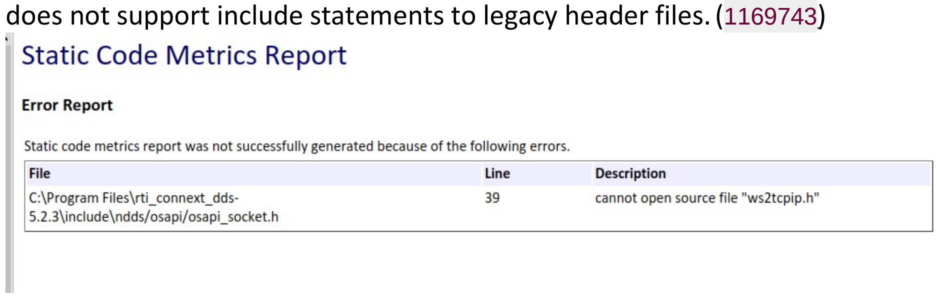
```
sudo ./DDSBlockset-R2016a_v2.x.x_glnxa64-Install --mode console
```

11.2 Simulink

- Sequences
  - Static Data Type code generation will generate incorrect code
  - Sequence of IDL structs are not allowed. Simulink does not allow a nested bus to be a sequence. It must have fixed length.
  - A vector of an IDL struct that contains a sequence element is not allowed. Simulink does not allow an element in a nested array of buses to be a variable length.
- Simulink Coder converts all multi-dimension arrays to one-dimension vectors. This causes a conflict with C structures defined for multi-dimensional arrays by rtiddsgen for Static TypeSystem. To work around this issue, header files created by rtiddsgen are not included in the code from Simulink, but rather the Simulink generated header files (with flattened matrices) are used. Function prototypes required from the rtiddsgen header files are hard coded in the Simulink generated headers.
  - One workaround is to not include header files from rtiddsgen into the code generated for the Simulink model. The risk of doing this is that code from rtiddsgen will be linked with code from Simulink model that is compiled with different header files to define the same typedef. If there are any differences in the struct definitions other than flattened matrices, this will likely result in a runtime error. If the user wishes to assume this risk, then the following preference can be enabled:
    - setpref('DDSBlockset','MatrixSupport','true');
- The IDL **string** data type is not supported for static type code generation. Simulink does not have built-in support for strings. Strings are therefore treated in Simulink as an array of unsigned bytes. In order for the Simulink Blocks to differentiate between a vector of bytes and a string, an alias data type, DDS_CharArray, was created. This alias type is used by the DDS Blockset infrastructure to differentiate between strings and byte vectors. Code generation with Static Data typing will be incorrect for any read/write access to the string in the Simulink model. Only dynamic typing should be used.
- Model Reference. If using DDS blocks in referenced models, the following rules must be followed:
  - There must be a DDSTarget block in each referenced model
  - An IDL file must be specified. The IDL file must be the same in all referenced models. See section 7.2.1.
  - The settings in the DDS Target block must be the same for all referenced models
- Importing IDL files with #include statement. If an IDL file includes another IDL file that is not on the MATLAB path rtiddsgen will fail. To work around this issue, add the '-I' switch to the DDS.import command as shown in the following example:
  - DDS.import('dC.idl','-Ic:\work\DDS\test\junk\slprj','f')
- Multitasking models. If your model has multiple rates and the model is configured for multitasking and DDS entity blocks execute in different rate groups you must select sample times for the DDS blocks so that the Domain

Participant block always runs at the same rate or faster than the connected Publisher/Subscriber blocks. The Publisher/Subscriber blocks must run at the same rate or faster than the Data Reader/Data Writer blocks. This required because initialization of DDS blocks occurs during the first-time step. Proper initialization requires that the DP block initialize before the Publisher/Subscriber block, and the publisher/subscriber blocks initialize before the Data Reader and Data Writer blocks.

- Data Reader Callbacks are not supported in Simulink or MATLAB. The user must use either a polling or WaitSet architecture for reading topic samples.
- C++ and C++ (class) code generation is not supported. Code generated with this code generation option will result in C++ code that may not compile.
- Only time-based WaitSets are supported.
- Accelerated mode builds on Linux will fail with a compilation error related to data types in the RTI DDS header files. Resolve this by changing the following parameter setting in the model:
  - set_param(model,'AccelMakeCommand', 'make_rtw MEX_OPTS="-DRTI_UNIX -DMX_COMPAT_32"')
- The Static Code Metrics report will give the following error. Currently Embedded Coder does not support include statements to legacy header files. (1169743)



Static Code Metrics Report

Error Report

Static code metrics report was not successfully generated because of the following errors.

File	Line	Description
C:\Program Files\rti_connext_dds-5.2.3\include\ndds/osapi/osapi_socket.h	39	cannot open source file "ws2tcpip.h"

11.3 MATLAB

- MATLAB code generation is not supported. If the user wishes to generated code for a MATLAB algorithm that includes DDS functionality, a Simulink model can be constructed which exercises the MATLAB algorithm via a MATLAB Function block. Any DDS function calls in the MATLAB code need to be replaced with corresponding Simulink DDS Blocks.
- Only time-based WaitSets are supported.

11.4 IDL Import

- DDS.import() does not support IDL multiple inheritance. However, the user can manually create Buses or MATLAB classes that represent Topics with multiple inheritance and send/receive topics of this type. Refer to section 7.1.9 for details.
- The IDL “@Optional” keyword is not supported. The keyword will be ignored when importing IDL into MATLAB and Simulink. Topic types used in MATLAB and Simulink will have all fields set to mandatory. As a result,

MATLAB and Simulink may not be able to communicate with other DDS Participants configured to send/receive topics with Optional fields.

- The IDL keyword @Extensibility MUTABLE_EXTENSIBILITY is not supported.
- If the IDL file has a large number of nested <module>, the workspace objects or MATLAB classes may have identifiers longer than the 63-character limit of MATLAB. Refer to 7.1.7 for a workaround.

11.5 IDL Export

- DDS.export() supports only Simulink Buses. MATLAB classes are not supported for export to IDL.

12 MacOS Support

Beginning with MacOS version 10.11, the System Integrity Protection (SIP) security feature puts restrictions on the use of DYLD_LIBRARY_PATH to add paths to the library search path. SIP prevents applications or spawned processes from inheriting the DYLD_LIBRARY_PATH environment variable. This hardware support package relied on DYLD_LIBRARY_PATH to locate the RTI Connext Libraries for S-Functions and MEX functions. As a result, library path information is added to both RTI Connext DDS libraries and DDS Blockset S-Functions and MEX functions based on the environment variable RTI_LD_LIBRARY_PATH.

DDS.Rpath.add() – Adds rpath information
DDS.Rpath.delete() – Deletes rpath information
DDS.Rpath.check() – Checks to see if libraries, sfunctions and MEX functions have correct rpath information.

The DDS.Rpath.add() – Function needs to be called once after the blockset is first installed. It will need to be rerun whenever the RTI DDS Connext libraries are moved or updated. These functions required that the environment variable RTI_LD_LIBRARY_PATH be set to the location of the RTI DDS Connext libraries.

12.1 Background

This section contains a detailed description of the rpath information added to RTI Connext DDS libraries and DDS Blockset S-Functions and MEX functions.

- The 3 RTI Connext DDS libraries used by MATLAB and Simulink are libnddsc.dylib, libnddscpp.dylib, libnddscore.dylib. These three libraries are updated as follows:
    - @rpath added as prefix to other RTI COnnext DDS libraries
    - The install name is updated to include @rpath prefix
- All DDS Blockset S-Functions and MEX functions are updated as follows:
    - @rpath added as prefix to other RTI COnnext DDS libraries
    - The location of the RTI COnnext DDS libraries is added to the rpath

Since the location of the RTI Connext Libraries on the customer’s computer is not known when the blockset installer is prepared, a utility is provided that the customer must run after the blockset has been installed. This utility will make the changes described above based on the library path referenced by environment variable RTI_LD_LIBRARY_PATH.

Useful MacOS commands to obtain info about a dylib.
otool -L dylibname : displays list of linked libraries
otool -D dylibname : displays the id for the dylib
otool -l dylibname : lists detailed info about dylib, including rpath.

Initially, the RTI DDS libraries have no rpath information and the install_name is the same as the library name:

```
Load command 14
      cmd LC_FUNCTION_STARTS
  cmdsize 16
  dataoff 785352
 datasize 5664
Load command 15
      cmd LC_DATA_IN_CODE
  cmdsize 16
  dataoff 791016
 datasize 0
[mmcbroom@bat4409maci:/sandbox/mmcbroom/rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0] ...
% otool -D libnddsc.dylib
libnddsc.dylib:
libnddsc.dylib
[mmcbroom@bat4409maci:/sandbox/mmcbroom/rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0] ...
% otool -L libnddsc.dylib
libnddsc.dylib:
        libnddsc.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1225.1.1)
        libnddscore.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.1.0)
[mmcbroom@bat4409maci:/sandbox/mmcbroom/rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0] ...
%
```


Changes:

1. Use install_name_tool to add @rpath before all dependent RTI libraries: libnddsc.dylib, libnddscore.dylib, libnddscpp.dylib

    ```
    Xcrun install_name_tool -change libnddscore.dylib
    @rpath/libnddscore.dylib libnddscore.dylib
    ```

```
% otool -L libnddsc.dylib
libnddsc.dylib:
        libnddsc.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1225.1.1)
        @rpath/libnddscore.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.1.0)
[mmcbroom@bat4409maci:/sandbox/mmcbroom/rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0] ...
```

2. Use install_name_tool to add @rpath to install_name

```
Xcrun install_name_tool -id @rpath/libnddsc.dylib
libnddsc.dylib
```

```
% otool -D libnddsc.dylib
libnddsc.dylib:
@rpath/libnddsc.dylib
```

DDS MEX functions and s-functions are then linked to these libraries.

1. Use install_name_tool Use install_name_tool to add location of RTI DDS Connext libraries to rpath

```
compatibility version 1.0.0
Load command 15
      cmd LC_FUNCTION_STARTS
  cmdsize 16
  dataoff 8488
 datasize 8
Load command 16
      cmd LC_DATA_IN_CODE
  cmdsize 16
  dataoff 8496
 datasize 0
[mmcbroom@bat4401maci:~/Documents/MATLAB/Add-Ons/Toolboxes/DDS Blockset/code] ...
% otool -L mexGetVersion.mexmaci64
mexGetVersion.mexmaci64:
        @rpath/libnddsc.dylib (compatibility version 0.0.0, current version 0.0.0)
        @rpath/libnddscore.dylib (compatibility version 0.0.0, current version 0.0.0)
        @rpath/libnddscpp.dylib (compatibility version 0.0.0, current version 0.0.0)
        @rpath/libmx.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.1.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1225.1.1)
[mmcbroom@bat4401maci:~/Documents/MATLAB/Add-Ons/Toolboxes/DDS Blockset/code] ...
%
```

==>

```
      cmd LC_FUNCTION_STARTS
  cmdsize 16
  dataoff 8488
 datasize 8
Load command 16
      cmd LC_DATA_IN_CODE
  cmdsize 16
  dataoff 8496
 datasize 0
Load command 17
          cmd LC_RPATH
      cmdsize 80
         path /sandbox/mmcbroom/rti_connext_dds-5.2.3/lib/x64Darwin15clang7.0 (offset 12)
[mmcbroom@bat4401maci:~/Documents/MATLAB/Add-Ons/Toolboxes/DDS Blockset/code] ...
% otool -L mexGetVersion.mexmaci64
mexGetVersion.mexmaci64:
	@rpath/libnddsc.dylib (compatibility version 0.0.0, current version 0.0.0)
	@rpath/libnddscore.dylib (compatibility version 0.0.0, current version 0.0.0)
	@rpath/libnddscpp.dylib (compatibility version 0.0.0, current version 0.0.0)
	@rpath/libmx.dylib (compatibility version 0.0.0, current version 0.0.0)
	/usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.1.0)
	/usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1225.1.1)
[mmcbroom@bat4401maci:~/Documents/MATLAB/Add-Ons/Toolboxes/DDS Blockset/code] ...
```

If the utility to add rpath information to either the RTI DDS libraries or the DDS Blockset sfunctions or MEX functions is not run, you will receive errors similar to the following:

```
>> DDS.version
Error using DDS.version
Invalid MEX-file '/home/mmcbroom/Documents/MATLAB/Add-Ons/Toolboxes/DDS
Blockset/code/mexGetVersion.mexmaci64':
dlopen(/home/mmcbroom/Documents/MATLAB/Add-Ons/Toolboxes/DDS
Blockset/code/mexGetVersion.mexmaci64, 6): Library not loaded: @rpath/libnddsc.dylib
  Referenced from: /home/mmcbroom/Documents/MATLAB/Add-Ons/Toolboxes/DDS
  Blockset/code/mexGetVersion.mexmaci64
  Reason: image not found.
```

13 Updating to a New Version of DDS RTI Connext

When updating to a new version of RTI Connext DDS and/or RTI Connext Micro DDS, the user need only exit MATLAB and then update the environment variables described in section **Error! Reference source not found.** When MATLAB is restarted, the DDS Blockset will use these environment variables to locate the new version of RTI Connext.

14 Using the DDS Toolbox with MATLAB Compiler

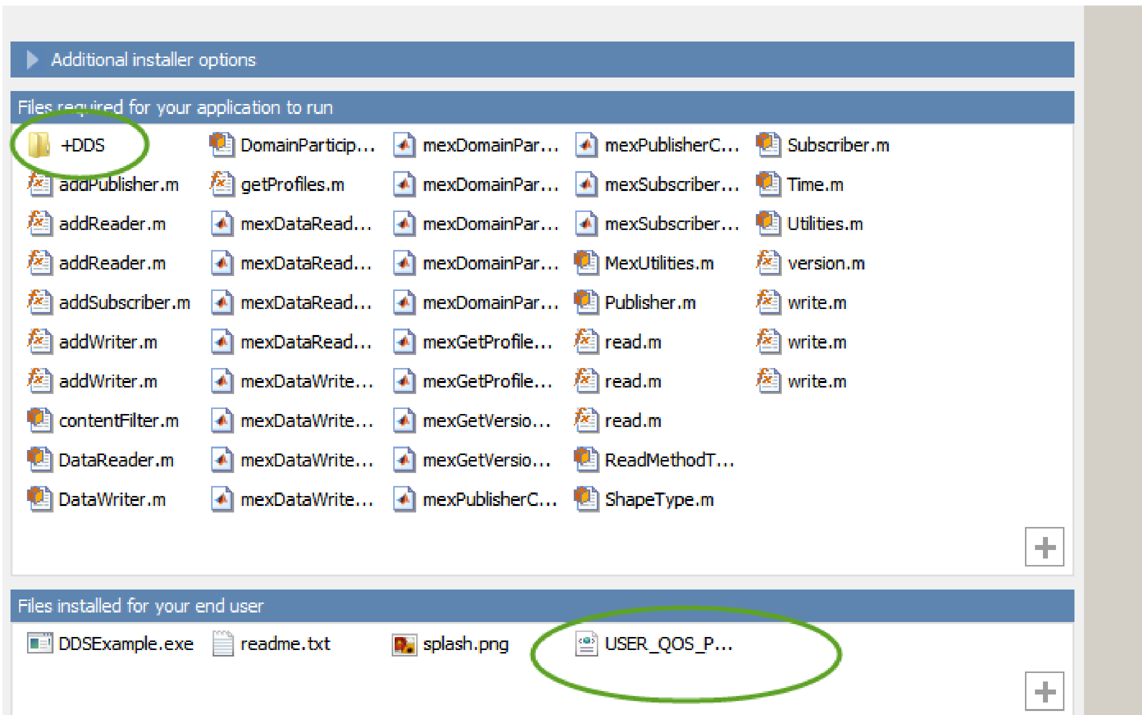
MATLAB Compiler can be used to deploy MATLAB applications that utilize the DDS Toolbox functions. The following additional files need to be added to the application.

Use the –a option of the mcc function to add the following files and directories to the .exe created by MATLAB Compiler. Without these additional files/directories, the deployed application will fail.

- All *.mex* files in the <matlabroot>/toolbox/psp/tools/DDSBlockset
- All .p and .m files in <matlabroot>/toolbox/psp/tools/DDSBlockset/+DDS

```
mcc('-v', '-a',
fullfile(matlabroot,'toolbox','psp','tools','DDSBlockset'), '-a',
fullfile(matlabroot,'toolbox','psp','tools','DDSBlockset','+DDS'), '-
a', 'USER_QOS_PROFILES.xml', '-m', 'myScript.m');
```

If a QoS XML file is being used, add to the “Files installed for end user” section.

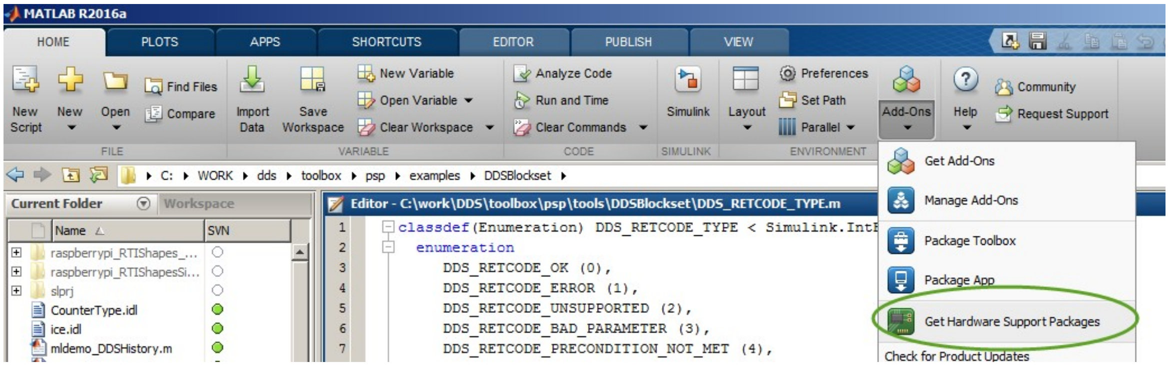


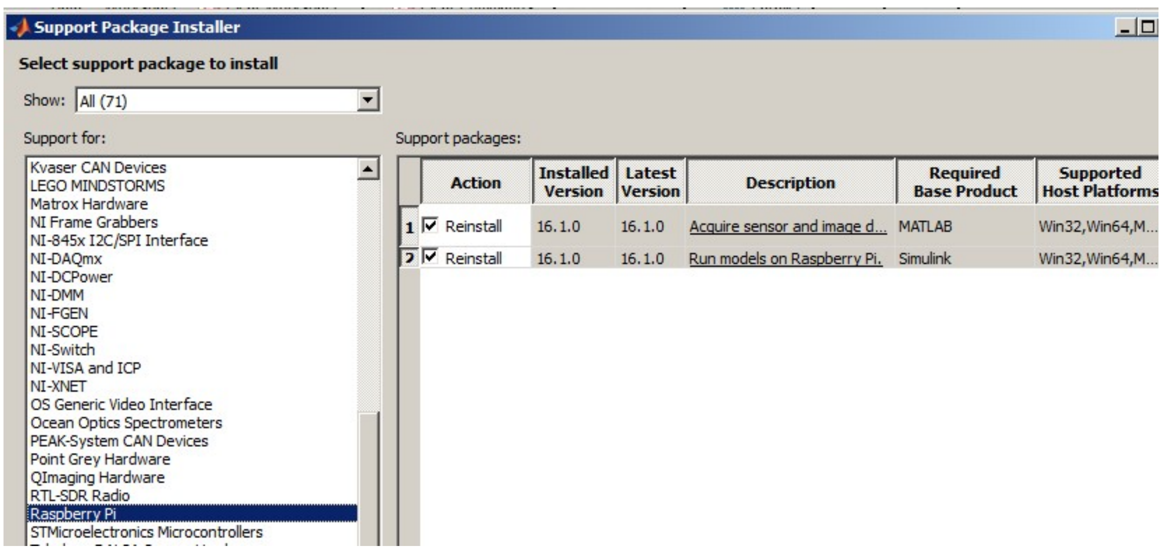
An example MATLAB Compiler project, DDSExample.prj, is installed in the <matlabroot>/toolbox/psp/examples/DDSBlockset directory.

15 Using the DDS Blockset with Raspberry Pi

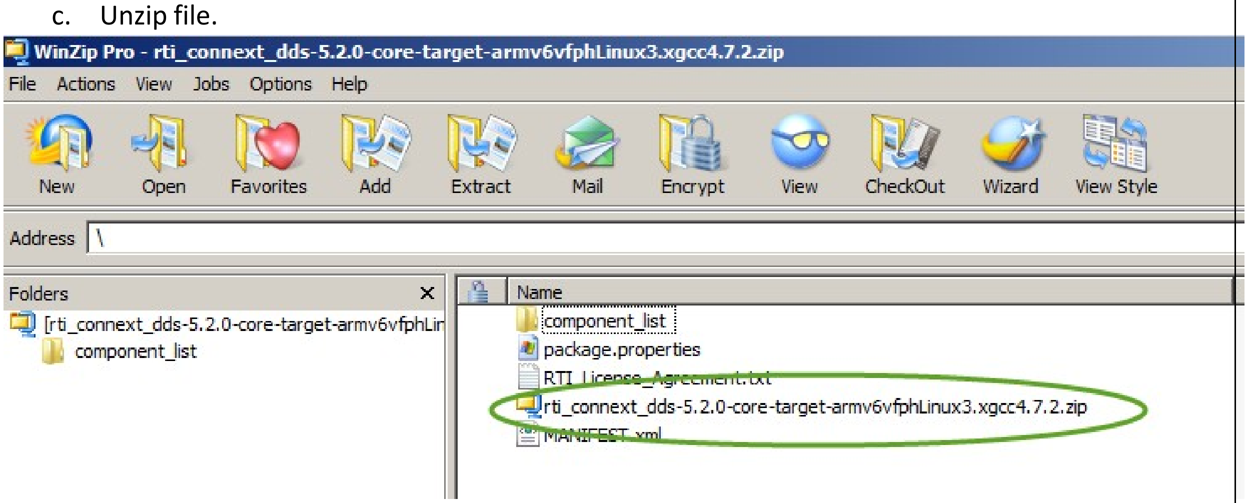
Steps for using Simulink DDS Blockset on Raspberry Pi:

1. Install MATLAB and Simulink Raspberry Pi support packages.

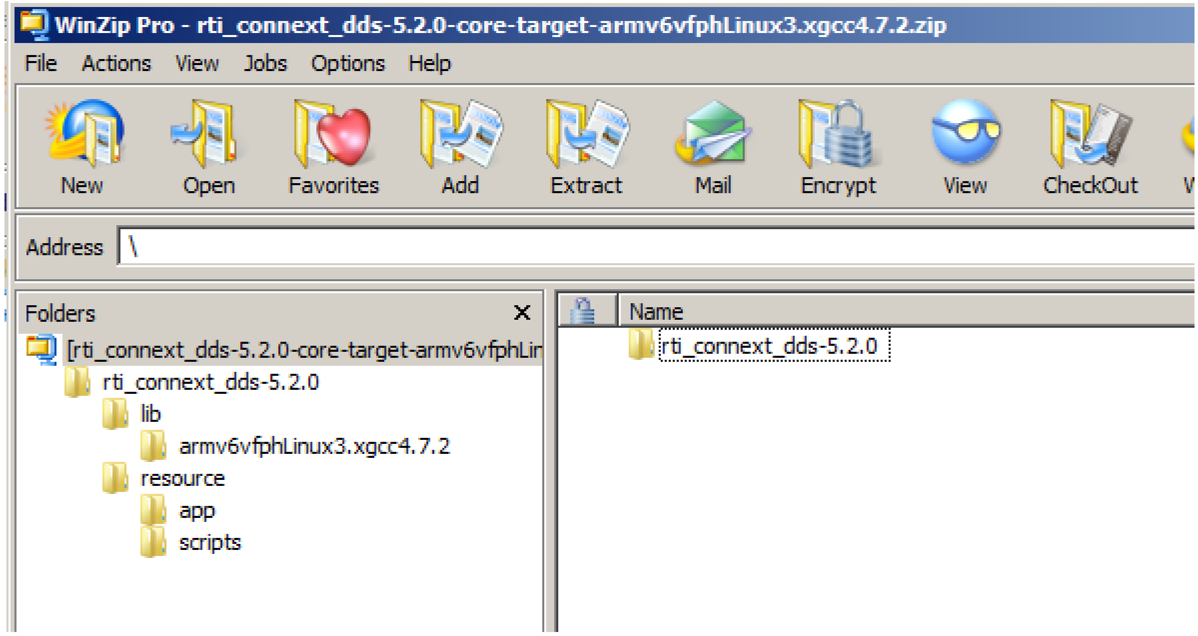




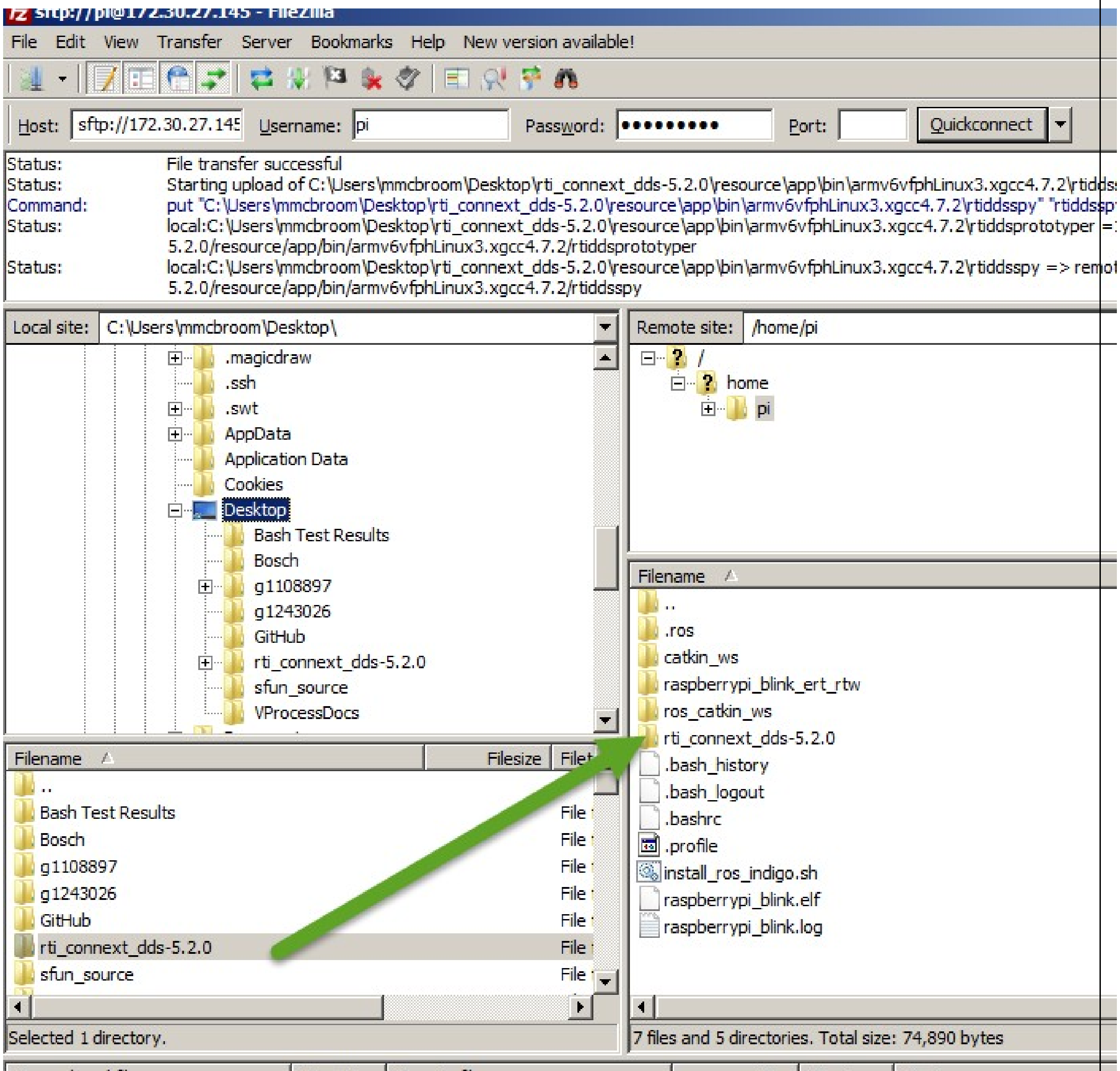
2. Run raspberrypi_gettingstarted example model to confirm all hardware support packages are installed correctly and Simulink can run.
3. Install DDS libraries on the Raspberry Pi:
    a. Get a copy of RTI DDS libraries for Raspberry Pi from here and save on your host computer.
    b. Change file extension of downloaded file to .zip.
    c. Unzip file.



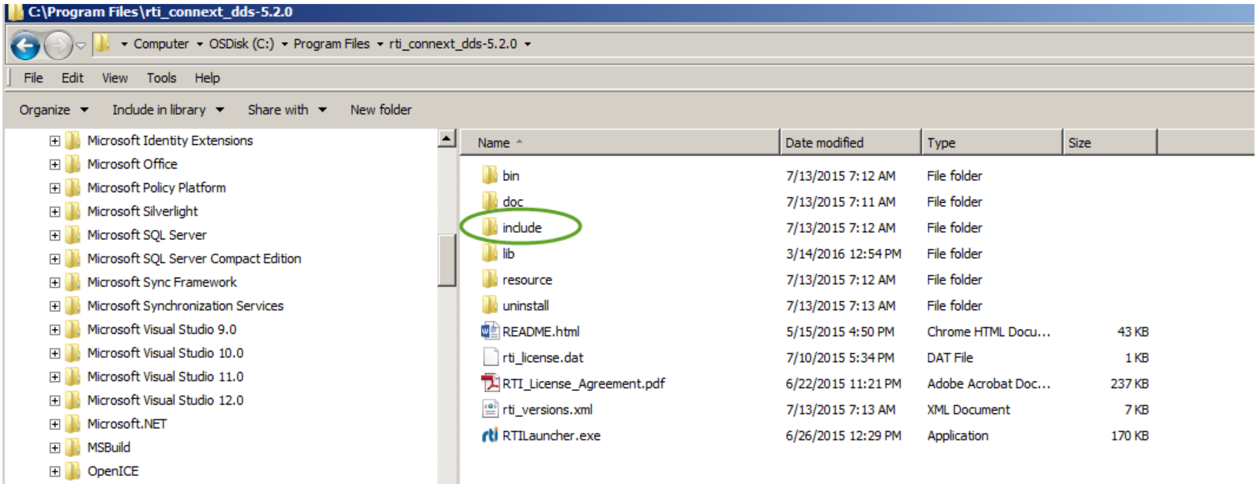
Then, unzip the file inside. It should have the following contents:



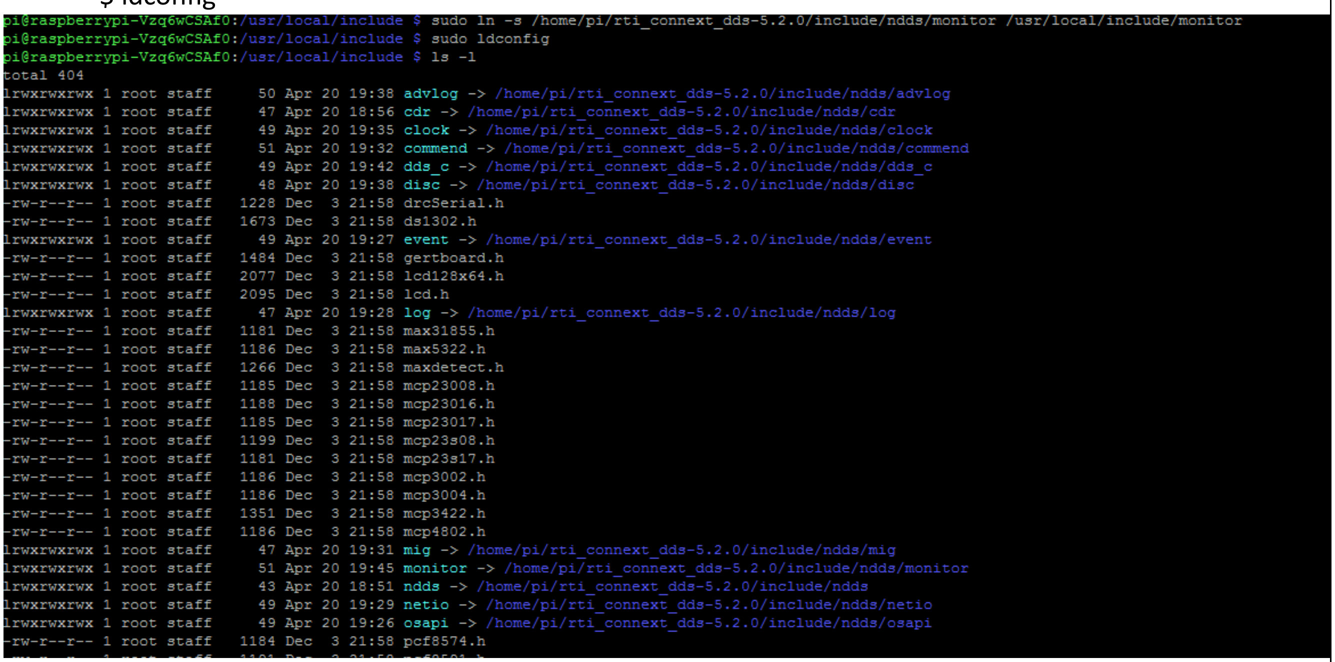
Use FTP to copy the folder rti_connext_dds-5.2.0 to your Raspberry Pi. Copy to location /home/pi/



4. The previous step only addresses binaries that are Raspberry Pi specific. We also need to copy the RTI DDS Connext include files to the target so that compile and link will work on Raspberry Pi.



5. Create soft links to add DDS include paths to user includes. This is required for the gcc compiler to be able to location the DDS include files.
   $ ln –s /home/pi/rti_connext_dds-5.2.0/include/ndds /usr/local/include/ndds
   $ ln –s /home/pi/ rti_connext_dds-5.2.0/include/ndds/advlog /usr/local/include/advlog
   $ ln –s /home/pi/ rti_connext_dds-5.2.0/include/ndds/cdr /usr/local/include/cdr
   Repeat for all directories in /home/pi/ rti_connext_dds-5.2.0/include/ndds/
   $ ldconfig



```
pi@raspberrypi-Vzq6wCSAf0:/usr/local/include $ sudo ln -s /home/pi/rti_connext_dds-5.2.0/include/ndds/monitor /usr/local/include/monitor
pi@raspberrypi-Vzq6wCSAf0:/usr/local/include $ sudo ldconfig
pi@raspberrypi-Vzq6wCSAf0:/usr/local/include $ ls -l
total 404
lrwxrwxrwx 1 root staff   50 Apr 20 19:38 advlog -> /home/pi/rti_connext_dds-5.2.0/include/ndds/advlog
lrwxrwxrwx 1 root staff   47 Apr 20 18:56 cdr -> /home/pi/rti_connext_dds-5.2.0/include/ndds/cdr
lrwxrwxrwx 1 root staff   49 Apr 20 19:35 clock -> /home/pi/rti_connext_dds-5.2.0/include/ndds/clock
lrwxrwxrwx 1 root staff   51 Apr 20 19:32 commend -> /home/pi/rti_connext_dds-5.2.0/include/ndds/commend
lrwxrwxrwx 1 root staff   49 Apr 20 19:42 dds_c -> /home/pi/rti_connext_dds-5.2.0/include/ndds/dds_c
lrwxrwxrwx 1 root staff   48 Apr 20 19:38 disc -> /home/pi/rti_connext_dds-5.2.0/include/ndds/disc
-rw-r--r-- 1 root staff 1228 Dec  3 21:58 drcSerial.h
-rw-r--r-- 1 root staff 1673 Dec  3 21:58 ds1302.h
lrwxrwxrwx 1 root staff   49 Apr 20 19:27 event -> /home/pi/rti_connext_dds-5.2.0/include/ndds/event
-rw-r--r-- 1 root staff 1484 Dec  3 21:58 gertboard.h
-rw-r--r-- 1 root staff 2077 Dec  3 21:58 lcd128x64.h
-rw-r--r-- 1 root staff 2095 Dec  3 21:58 lcd.h
lrwxrwxrwx 1 root staff   47 Apr 20 19:28 log -> /home/pi/rti_connext_dds-5.2.0/include/ndds/log
-rw-r--r-- 1 root staff 1181 Dec  3 21:58 max31855.h
-rw-r--r-- 1 root staff 1186 Dec  3 21:58 max5322.h
-rw-r--r-- 1 root staff 1266 Dec  3 21:58 maxdetect.h
-rw-r--r-- 1 root staff 1185 Dec  3 21:58 mcp23008.h
-rw-r--r-- 1 root staff 1188 Dec  3 21:58 mcp23016.h
-rw-r--r-- 1 root staff 1185 Dec  3 21:58 mcp23017.h
-rw-r--r-- 1 root staff 1199 Dec  3 21:58 mcp23s08.h
-rw-r--r-- 1 root staff 1181 Dec  3 21:58 mcp23s17.h
-rw-r--r-- 1 root staff 1186 Dec  3 21:58 mcp3002.h
-rw-r--r-- 1 root staff 1186 Dec  3 21:58 mcp3004.h
-rw-r--r-- 1 root staff 1351 Dec  3 21:58 mcp3422.h
-rw-r--r-- 1 root staff 1186 Dec  3 21:58 mcp4802.h
lrwxrwxrwx 1 root staff   47 Apr 20 19:31 mig -> /home/pi/rti_connext_dds-5.2.0/include/ndds/mig
lrwxrwxrwx 1 root staff   51 Apr 20 19:45 monitor -> /home/pi/rti_connext_dds-5.2.0/include/ndds/monitor
lrwxrwxrwx 1 root staff   43 Apr 20 18:51 ndds -> /home/pi/rti_connext_dds-5.2.0/include/ndds
lrwxrwxrwx 1 root staff   49 Apr 20 19:29 netio -> /home/pi/rti_connext_dds-5.2.0/include/ndds/netio
lrwxrwxrwx 1 root staff   49 Apr 20 19:26 osapi -> /home/pi/rti_connext_dds-5.2.0/include/ndds/osapi
-rw-r--r-- 1 root staff 1184 Dec  3 21:58 pcf8574.h
```

6. Create soft link for DDS library.

7. If your Simulink model uses a QoS XML file, you will need to copy this file to the Raspberry Pi. Also, if your Raspberry Pi is on a different subnet, you may need to copy the NDDS_DISCOVERY_PEERS file to the Raspberry Pi.