ADLINK

VORTEX

OpenSplice Simulink Guide

Release 6.x

Contents

1

Introduction

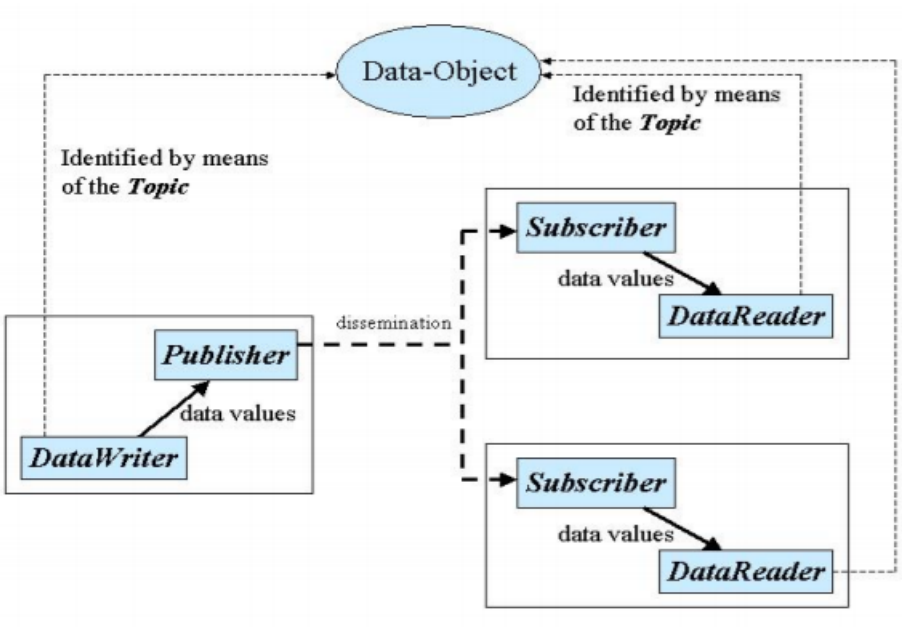
The DDS Simulink Integration provides users with DDS custom blocks to model DDS communication between Simulink models and pure DDS applications.

1.1 DDS

What is DDS?

"The Data Distribution Service (DDS™) is a middleware protocol and API standard for data-centric connectivity from the Object Management Group® (OMG®). It integrates the components of a system together, providing low-latency data connectivity, extreme reliability, and a scalable architecture that business and mission-critical Internet of Things (IoT) applications need."

"The main goal of DDS is to share the right data at the right place at the right time, even between time-decoupled publishers and consumers. DDS implements global data space by carefully replicating relevant portions of the logically shared dataspace." DDS specification



Further Documentation

http://portals.omg.org/dds/

http://ist.adlinktech.com/

1.2 Simulink

What is Simulink?

"Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis." Simulink Help

2

Installation

This section describes the procedure to install the Vortex DDS Simulink Integration on a Linux or Windows platform.

2.1 System Requirements

- Operating System: Windows or Linux
- MATLAB Simulink installed
- Java 1.7 or greater

2.2 OpenSplice (OSPL) and DDS Simulink Installation

Steps:

1. Install OSPL. The DDS Simulink Integration is included in this installer.
2. Setup OSPL license. Copy the license.lic file into the appropriate license directory.

   /INSTALLDIR/ADLINK/Vortex_v2/license
3. MATLAB and Simulink files are contained in a tools/matlab folder

   Example: */INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux/tools/matlab*

2.3 OpenSplice (OSPL) Configuration

By default OSPL uses single process configuration.

If however, shared memory configuration is used, additional OSPL configuration steps need to be taken to work with MATLAB Simulink.

2.3.1 Linux

OSPL-9882 Linux: MATLAB/Simulink hangs when connecting to shared memory domain

Description On Linux, a MATLAB script or Simulink model connecting to a Vortex OpenSplice domain via shared memory will hang.

Resolution MATLAB, like Java applications requires that the environment variable LD_PRELOAD be set to reference the active Java installations libjsig.so library. The MATLAB user interface uses Java, and thus requires the same signal handling strategy as Java applications connecting to Vortex OpenSplice. The precise syntax for setting the LD_PRELOAD environment variable will depend on the shell being used.

The libjsig.so file you specify in LD_PRELOAD should match the Java installation used by MATLAB. By default, MATLAB uses a private java installation. If you have not explicitly specified a MATLAB java version (by setting

the MATLAB_JAVA environment variable), you should use the libjsig.so library that ships with MATLAB. You can find the library with the following command:

find *MATLAB-install-dir* -name libjsig.so

As an example, MATLAB R2016b installed in the default location, LD_PROLOAD should contain:
/usr/local/MATLAB/R2016b/sys/java/jre/glnxa64/jre/lib/amd64/libjsig.so

If you have set MATLAB_JAVA, then you should use the libjsig.so from that installation. For example, on Oracle JVMs, LD_PRELOAD should contain this value:

$JAVA_HOME/jre/lib/amd64/libjsig.so

2.3.2 Windows

OSPL-10018 MATLAB: Shared Memory Database Address on Windows needs to be changed from default

Description On a Windows 64-bit system, an OpenSplice system configured with Shared Memory, MATLAB cannot connect to the OpenSplice domain if the Shared Memory Database Address is set to its default value of 0x40000000. The error log (ospl-error.log) will show entries such as: Report : Can not Map View Of file: Attempt to access invalid address. Internals : OS Abstraction/code/os_sharedmem.c/1764/0/1487951812.565129500

Resolution Use the configuration editor to change the default data base address. Use the 'Domain' tab, and select the 'Database' element in the tree. If necessary, right click the Database element to add an 'Address' element. Change the address. In general, a larger number is less likely to be problematic. On a test machine, appending two zeros to the default address allowed for successful connections.

2.4 Simulink Setup

Steps:

1. Open command shell and run script to setup environment variables.

   Linux

   - Open a Linux terminal.
   - Navigate to directory containing release.com file.

     /INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux
   - Run release.com. (Type in ". release.com" at command line.)

   Windows

   - Open a command prompt.
   - Navigate to directory containing release.bat file.

     INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.win64
   - Run release.bat. (Type in "release.bat" at command line.)

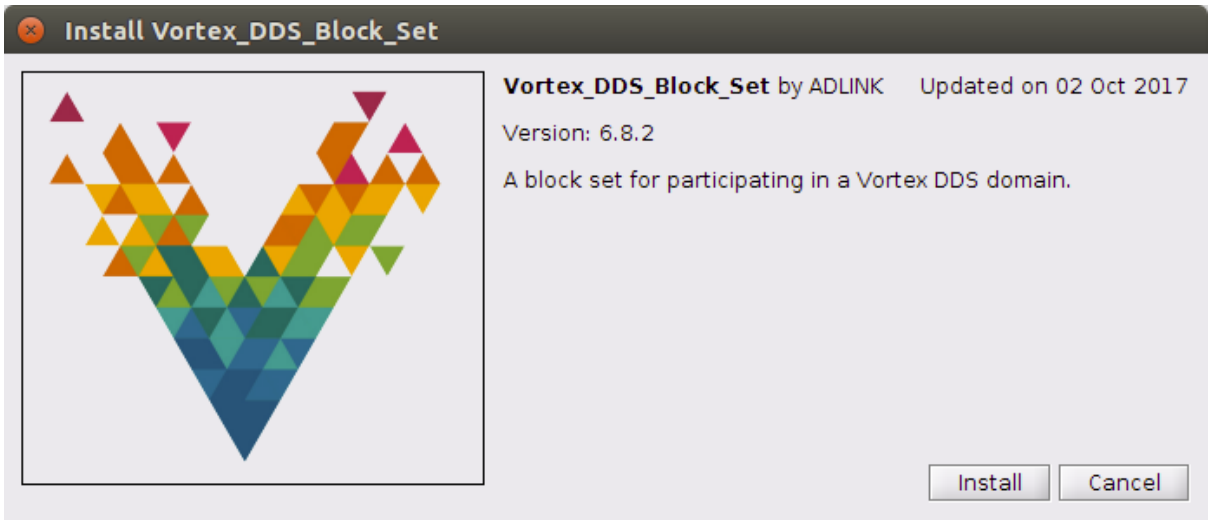
2. Start MATLAB using the **SAME** command shell used in Step 1.

   NOTE: If MATLAB is NOT started from a command shell with the correct OSPL environment variables set, exceptions will occur when attempting to use DDS Simulink blocks.

3. In MATLAB, navigate to file "Vortex_DDS_Block_Set.mltbx" by typing:

```
cd(fullfile(getenv('OSPL_HOME'),'tools','matlab'))
```

4. Double click on the file "Vortex_DDS_Block_Set.mltbx". This will bring up a dialog entitled **Install Vortex_DDS_Block_Set**. Select **Install**.



Setup is complete!

2.5 Examples

Example models have been provided in the examples folder.

../tools/matlab/examples/simulink

3

Vortex DDS Blocks

The DDS Simulink Integration provides a block library with custom blocks to model reading and writing data with DDS.

The Vortex DDS Simulink block library provides blocks which correspond to DDS entities. (Each DDS block is covered in its own section in this user guide.)

The following DDS block types are provided:

- Topic
- Domain
- Publisher
- Subscriber
- Writer
- Reader

3.1 Optional DDS Blocks and Ports

Some of the DDS blocks are optional. (Domain, Publisher and Subscriber)

When the optional blocks are not added to model diagrams, defaults are used. This allows for simpler model diagrams. If the model requires block parameter customization, the optional blocks can be added to a model to use non-default settings.

Many of the ports for the DDS blocks are also optional. They can be toggled on or off in the **Block Parameters** dialog, in the **Ports** tab.

3.2 QoS Profiles

In DDS - "The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service."

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

The QoS profile of a block is set in the **QoS** tab of the **Block Parameters** dialog. (This dialog is opened by double clicking on a selected block.)

Please see section *QoS Provider* for more information.

3.3 Simulink Block Sample Time

"What is sample time?"

"The sample time of a block is a parameter that indicates when, during simulation, the block produces outputs and if appropriate, updates its internal state." -Simulink documentation

The DDS blocks have different sample times set. The only DDS block that allows for the user specification of a sample time is the **Reader** block. A reader's sample time can be set in the **Block Parameters Data** tab.

A sample time of 0 means that the block step will only execute once.

A sample time of -1 means that the block will inherit its sample time from its inputs or from the parent model.

DDS Block Type	Sample Time
Topic	-1 (inherits) uneditable
Domain	-1 (inherits) uneditable
Publisher	-1 (inherits) uneditable
Subscriber	-1 (inherits) uneditable
Writer	-1 (inherits) uneditable
Reader	• default -1 (inherits) editable • Inherits from inputs or model • Valid values: -1 and Numeric > 0

4

Simulink Bus to DDS Topic Mapping

Simulink data is represented in buses whose types are not compatible with DDS topic data types.

When using the Simulink Vortex DDS library, the user must create Simulink buses that will be mapped to DDS topic types.

The Vortex DDS **Reader**, **Writer** and **Topic** blocks have block parameters that require a Simulink bus type. On data writes, the Simulink bus types are converted to DDS topic types and on data reads, the DDS topic types are converted to Simulink bus types.

The user can generate/create the Simulink bus definitions by either generating them from an IDL file, or by using the Simulink bus editor.

4.1 Generate Simulink bus definitions from an IDL file

DDS Topic Types can be described in an IDL file. The public Vortex.idlImportSl function can be called to generate Simulink bus definitions from an IDL file.

idlImportSl(IDLFILENAME,DICTIONARYFILE)

Given an IDLFILENAME, invokes the idlpp tool to generate a MATLAB script. This script is then used to create Simulink.Bus objects and import them into the specified data dictionary DICTIONARYFILE. The values are inserted into the 'Design Data' section of the data dictionary. If the target data dictionary already contains definitions for the bus or enum names, they are overwritten. If idlpp returns non-zero, the function returns immediately with the error code.

Input Arguments:

IDLFILENAME A character array with the value of the IDL file name to process.

DICTIONARYFILE A character array with the value of the data dictionary file name.

Steps:

1. In the IDL file, ensure that any topic structures have the OSPL specific annotation *pragma keylist* defined.

   If you want a given IDL structure to serve as the topic type, the structure requires an OSPL specific annotation. The #pragma keylist <data-type-name> <key>* declaration should be defined after the structure declaration.

   IMPORTANT NOTE: The IDL file has to have a blank line after the pragma keylist declaration. (BUG)

   More information can be found at: **IDLPreProcGuide_**

2. In MATLAB, navigate to the directory that contains the IDL file. Set this directory to be the MATLAB **Current Folder**.

3. Call the idlImportSl function in the MATLAB command window.

   Example: >> Vortex.idlImportSl('ShapeType.idl', 'shape.sldd')

4.2 Overriding default values for Vortex.idlImportSl

The `Vortex.idlImportSl` function makes a number assumptions when generating a Simulink data dictionary from your IDL file. These are documented in the following table:

IDL Element	Default Simulink Equivalent
struct name	Unqualified bus name
enum name	Unqualified enum name
sequence<*T*, *N*> name	*seqN_T*
sequence<*T*> name	*seq_T*
unbounded string	maximum 256 characters
unbounded sequence	maximum 16 elements

The following subsections illustrate how to override each of these defaults.

4.2.1 Overriding struct and enum names

IDL allows you to describe a hierarchical set of namespaces for structs and enums by defining modules. Simulink, however, has a flat namespace for all busses and enumerations. This can present problems when the same unqualified struct name is used to two or more different modules.

Consider the following idl:

```
module A {
  enum Status {
     INFO,
     WARNING,
     ERROR
  };
  struct Message {
      long id;
      Status status;
  };
  #pragma keylist Message id
};

module B {
  enum Status {
     SUCCESS,
     FAILURE
  };
  struct Message {
      long id;
      Status status;
  };
  #pragma keylist Message id
};
```

In it, both `Status` and `Message` are used in the context of both module A and module B. When you run `Vortex.idlImportSl` against it, you will receive the following messages:

```
Error: IDL element B::Status translates to the Simulink artifact named Status,
which already corresponds to another IDL element A::Status.
Edit the idl_defaults.properties file to assign unique Simulink names to each IDL element.

Error: IDL element B::Message translates to the Simulink artifact named Message,
which already corresponds to another IDL element A::Message.
Edit the idl_defaults.properties file to assign unique Simulink names to each IDL element.
```

To help you solve the problem, the IDL import function generates a *properties file* with the same name as the original IDL file, but with a *properties* extension. Here is the generated properties file:

```
# <initial comments removed for brevity>
#
# To change the generated Simulink element, change one or more values in this file, and then
# re-run the IDL import.

A::Message#name = Message
A::Status#name = Status
B::Message#name = Message
B::Status#name = Status
```

The properties file shows the names the IDL import attempted to used. Each line is of the format:

```
<qualified-IDL-name>#name = <simulink-name>
```

You can modify any value after the equals sign (=), to create a unique name. Once you have finished editing the properties file, re-run the IDL import to update actual Simulink entities.

Suppose we make the following changes to the properties file:

```
A::Message#name = MessageA
A::Status#name = StatusA
B::Message#name = Message
B::Status#name = Status
```

On re-running `Vortex.idlImportSl`, the generated Simulink data dictionary would have contents as seen in the figure below.

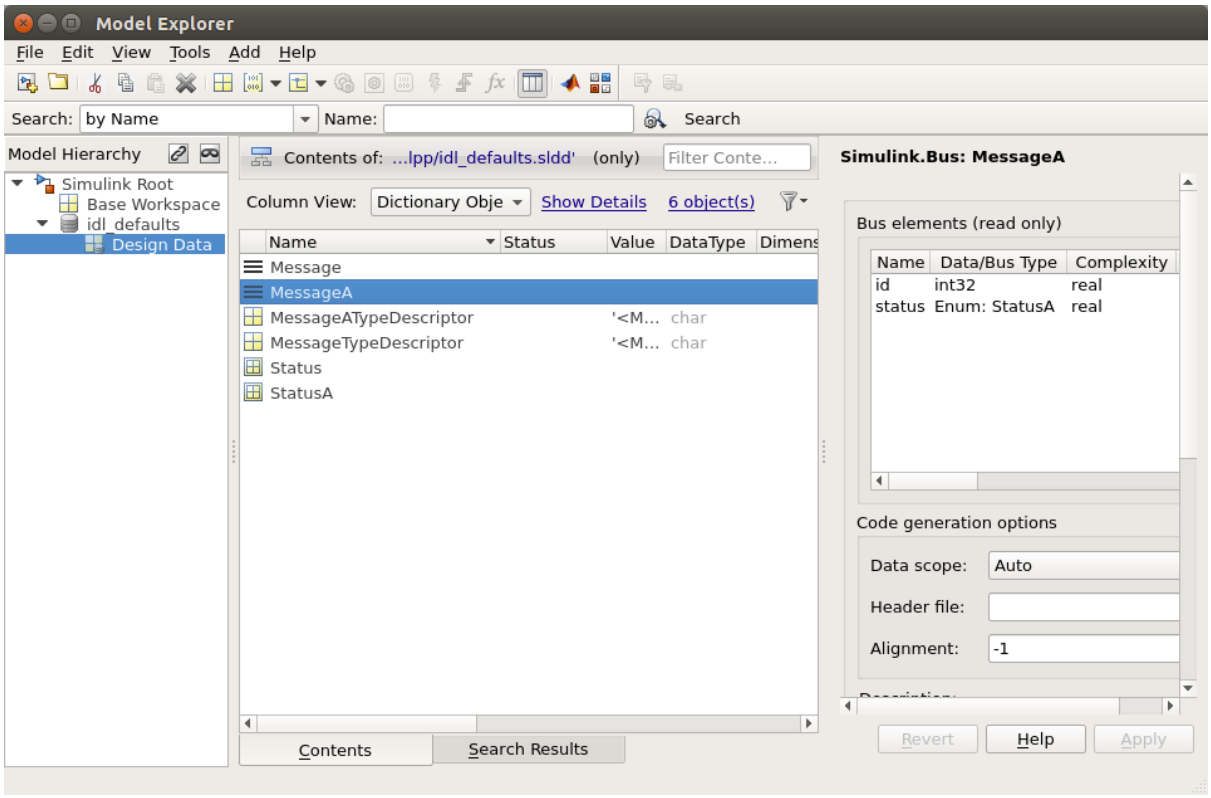


Figure 4.1: *View of generated Simulink Data Dictionary, showing renamed busses and enumerations.*

4.2.2 Changing unbound string maximum sizes

Although IDL unbound strings can be of any length, in Simulink they are mapped to fixed length arrays of int8. The default length the `Vortex.idlImportSl` assigns to such arrays is 256 characters. You can change this default value, per string, using the generated properties file.

Suppose we modify A::Message to contain a string field:

```
module A {
    enum Status {
       INFO,
       WARNING,
       ERROR
    };
    struct Message {
        long id;
        Status status;
        string body;
    };
    #pragma keylist Message id
};
```

On running the IDL import, the properties file would be updated to show a new line:

```
A::Message.body#stringMax = 256
```

Changes the value to something new (say 128), and rerun the IDL import to update the generated Simulink bus.

4.2.3 Changing sequence attributes

IDL import generates a simulink bus for each IDL sequence discovered. For an unbound sequence, the default name for such sequences is *seq_T*, where *T* is the type of elements in the sequence. For bounded sequences, the default sequence name is *seqN_T*, where *N* is the declared upper bound of the sequence.

The following IDL enhances our example IDL to use a sequence of string field for the body attribute:

```
module A {
    enum Status {
       INFO,
       WARNING,
       ERROR
    };
    struct Message {
        long id;
        Status status;
        sequence<string> body;
    };
    #pragma keylist Message id
};
```

On running IDL import, the generated property file now contains the following lines:

```
A::Message.seq_string#name = seq_string
A::Message.seq_string#seqMax = 16
A::Message.seq_string#stringMax = 256
```

You can edit the value of `A::Message.seq_string#name` to change the name of the generated Simulink bus representing the sequences. You can change the maximum number of elements stored in the Simulink representation of the sequence by editing `A::Message.seq_string#seqMax`. Finally, because the type of elements in this sequence are unbounded strings, you can also change the maximum size of each string in the sequence (`A::Message.seq_string#stringMax`).

4.3 Add Simulink bus definitions using bus editor

Users can also model the Simulink buses using the Simulink bus editor.

Please see *Tutorial* for an example with detailed steps.

4.4 Bus definition limitations

The DDS Simulink integration has some limitations. Provided below is a table of unsupported types. Some of these bus definition limitations will be removed in later release(s).

Please refer to *Appendix A* for more detailed implementation details.

Unsupported Simulink Types
long long
unsigned long long
wchar
wstring
any
long double
union
inheritance

See also IDL PreProcessor Guide chapter Keys.

5

QoS Provider

Each Vortex DDS block has a QoS that can be set using the **Block Parameters**.

The following section explains how the QoS is set for a DDS entity using the QoS Provider.

5.1 QoS Provider File

Quality of Service for DDS entities is set using XML files based on the XML schema file DDS_QoSProfile.xsd. These XML files contain one or more QoS profiles for DDS entities. An example with a default QoS profile for all entity types can be found at DDS_DefaultQoS.xml.

Note: Sample QoS Profile XML files can be found in the examples directories.

5.2 QoS Profile

A QoS profile consists of a name. The file contains QoS elements for one or more DDS entities. A skeleton file without any QoS values is displayed below to show the structure of the file.

```
<dds xmlns="http://www.omg.org/dds/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="file:DDS_QoSProfile.xsd">
    <qos_profile name="DDS QoS Profile Name">
        <datareader_qos></datareader_qos>
        <datawriter_qos></datawriter_qos>
        <domainparticipant_qos></domainparticipant_qos>
        <subscriber_qos></subscriber_qos>
        <publisher_qos></publisher_qos>
        <topic_qos></topic_qos>
    </qos_profile>
</dds>
```

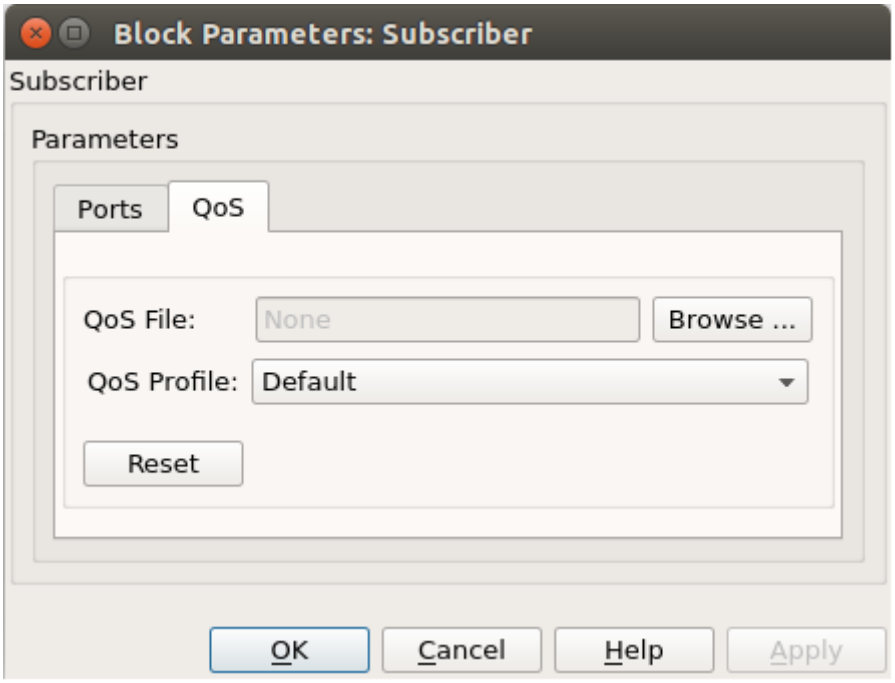
Example: Specify Publisher Partition

The example below specifies the publisher's partitions as A and B.

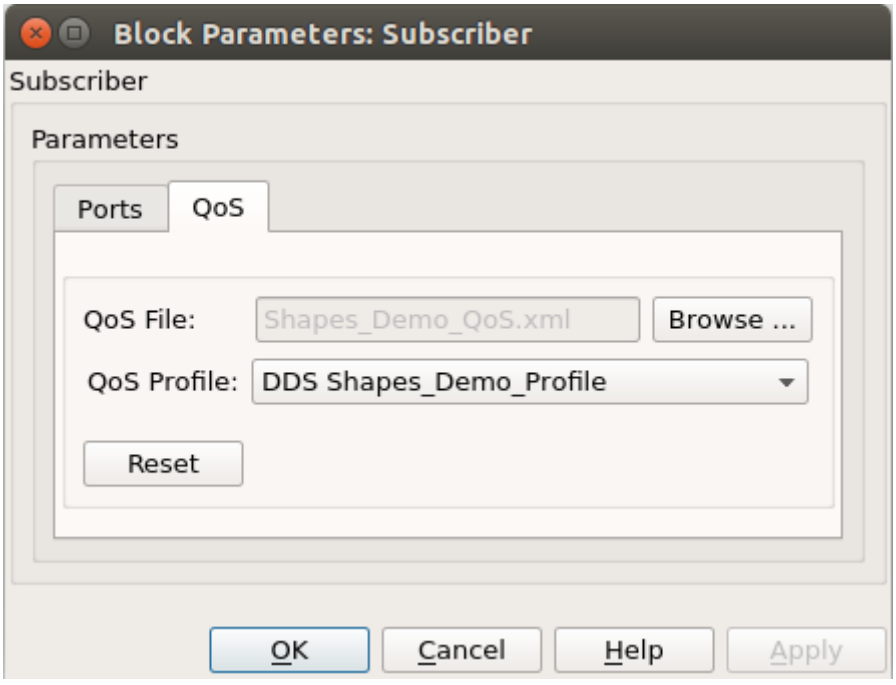
```
<publisher_qos>
    <partition>
        <name>
            <element>A</element>
            <element>B</element>
        </name>
    </partition>
</publisher_qos>
```

5.3 Setting QoS Profile in Simulink

QoS profiles are set using the Simulink block's parameters dialog under the QoS tab. If the QoS File parameter is set to None the default QoS settings will be used. The Reset button sets the parameters to the default values.

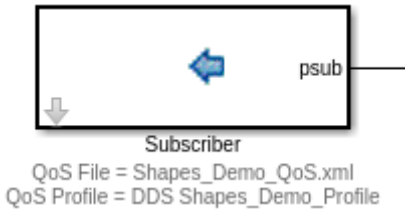


A QoS Provider file can be selected by browsing to the XML file. Once the file is chosen the file name is displayed and the user is presented with a drop down list of all QoS Providers in the file.



Note: Seeing the QoS Profile in the drop down list only guarantees the QoS Profile exists in the file. It does not mean the qos tag exists for the entity. The user is responsible for verifying the entity qos tag exists in the file.

Simulink block annotations are visible by default to display the QoS File Name and the QoS Profile settings.



5.4 Known Limitations

See QoS Provider Known Limitations for a list of limitations on QoS Provider support.

6

Topic Block

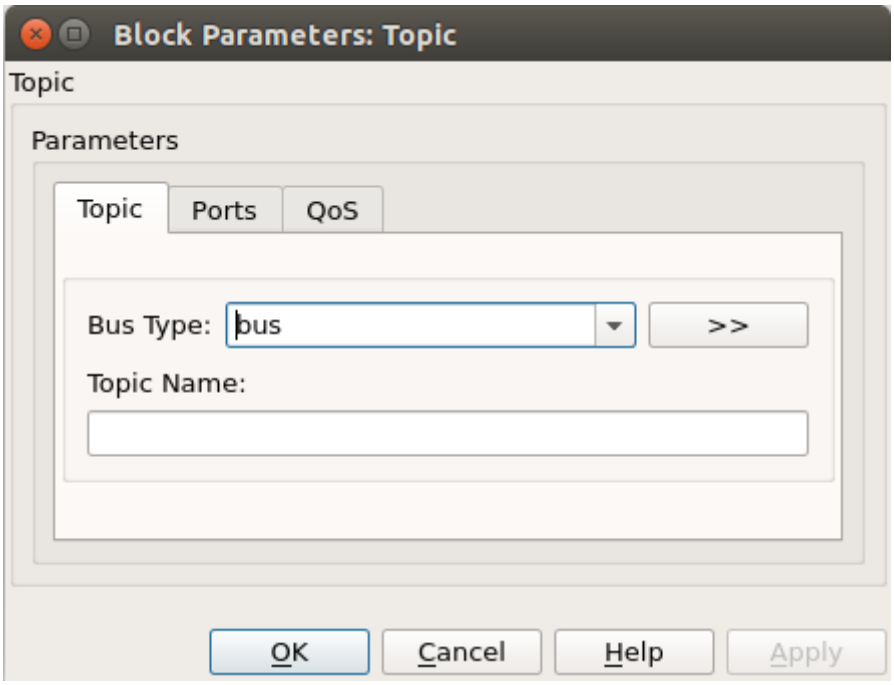
The topic block represents a DDS topic type. The DDS topic corresponds to a single data type. In DDS, data is distributed by publishing and subscribing topic data samples.

For a DDS Topic type definition, a corresponding BUS should be defined in the MATLAB workspace. The name of the BUS and the fields and field types should correspond to the DDS topic IDL definition.

In Simulink, a BUS definition can be used as an input or output signal of the Simulink building blocks.

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	topic	DDS Topic entity instance	Writer, Reader

6.1 Topic Block Parameters



6.1.1 Topic Tab

The output port named **topic**, needs to be configured by the user. No defaults are provided. To configure the **topic** output port, edit the required parameters in the **Block Parameters / Topic** tab. The following topic parameters must be specified: **Bus Type** and **Topic Name**.

6.1.2 Ports Tab

The **Ports** tab allows the user to toggle on or off optional ports.

6.1.3 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

7

Domain Block

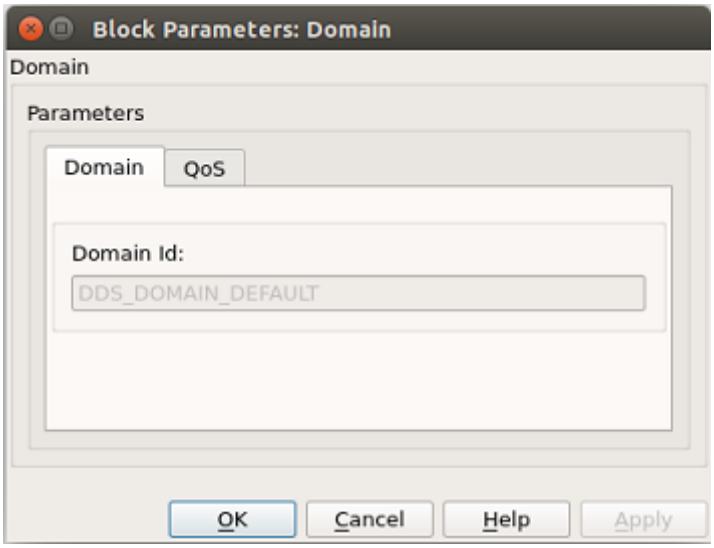
The Domain block represents a DDS domain participant entity.

In DDS - "A domain participant represents the local membership of the application in a domain. A domain is a distributed concept that links all the applications able to communicate with each other. It represents a communication plane: only the publishers and subscribers attached to the same domain may interact."

This block is optional on a DDS Simulink model diagram. If it is not present on a model diagram, a default participant will be created by either a topic, writer or reader block.

Port Type	Optional	Name	Description	Output consumed by
Output	no	pp	DDS Domain Participant entity instance	Publisher, Subscriber, Topic

7.1 Domain Block Parameters



7.1.1 Domain Tab

The domain id is read only. The domain id is the OSPL default domain id specified in the OSPL configuration file.

7.1.2 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

8

Publisher Block

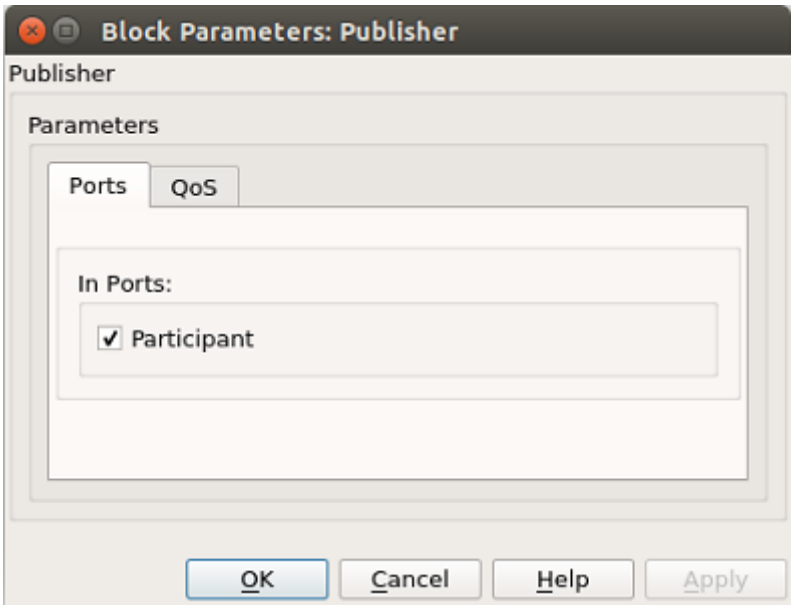
The Publisher block represents a DDS publisher entity.

In DDS, a publisher is “an object responsible for data distribution. It may publish data of different data types.”

This block is optional on a DDS Simulink model diagram. If it is not present on a model diagram, each writer will create a default publisher.

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	ppub	DDS Publisher entity instance	Writer

8.1 Publisher Block Parameters



8.1.1 Ports Tab

The **Ports** tab allows the user to toggle on or off optional ports.

8.1.2 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

9

Subscriber Block

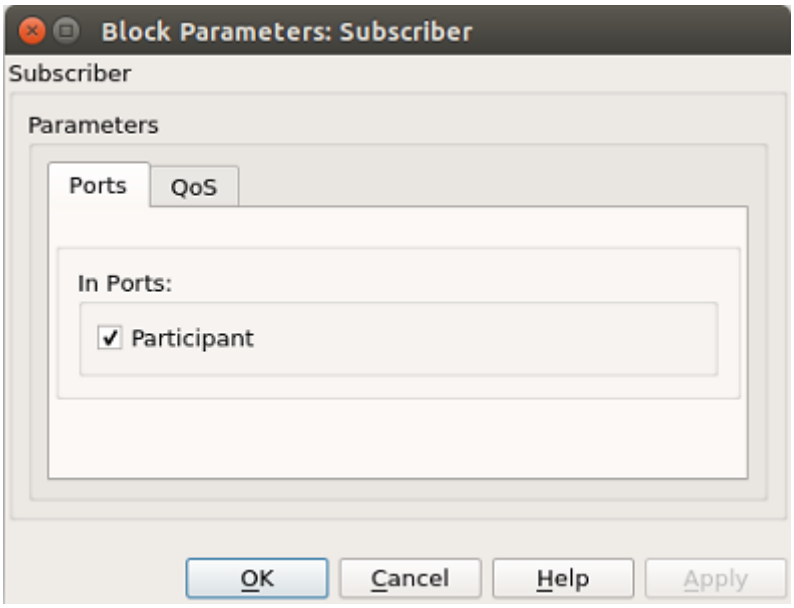
The Subscriber block represents a DDS subscriber entity.

In DDS, a subscriber is "an object responsible for receiving published data and making it available to the receiving application. It may receive and dispatch data of different specified types."

This block is optional on a DDS Simulink model diagram. If it is not present on a model diagram, each reader will create its own default subscriber.

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	psub	DDS Subscriber entity instance	Reader

9.1 Subscriber Block Parameters



9.1.1 Ports Tab

The **Ports** tab allows the user to toggle on or off optional ports.

9.1.2 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

10

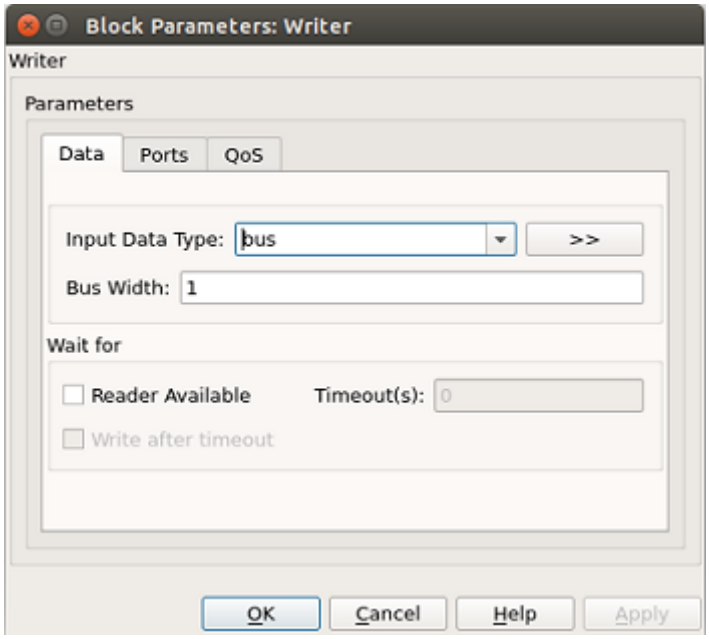
Writer Block

The Writer block represents a DDS data writer entity.

In DDS - “The DataWriter is the object the application must use to communicate to a publisher the existence and value of data-objects of a given type.”

Port Type	Op-tional	Name	Description	Output consumed by
Input	yes	ppub	DDS Publisher entity instance	
Input	no	topic	DDS Topic entity instance	
Input	no	data	BUS	
Input	yes	action	0 write, 1 dispose, 2 write dispose, 3 no operation	
Output	yes	status	0 for successful writer creation	
Output	yes	samples written	Number of samples written	User

10.1 Writer Block Parameters



10.1.1 Data Tab

The **Data** tab is used to set the input data type (BUS) for the **data** input port and the **bus width**.

The bus width is the maximum number of samples that can be written per block step. The user must configure the source blocks that feed the Writer's data port so that it produces an array of the right size.

Valid values for the bus width are: integers >= 1.

The Reader Available field in the Wait for section is used for specifying if the Writer should wait for the Reader to become available. The associated Timeout field is to specify how long (in seconds) the Writer should wait for the Reader to become available.

The Write after timeout field can only be enabled when the Reader Available field is checked. It specifies if the Writer should write after the Wait for Reader Available timeout.

10.1.2 Ports Tab

The **Ports** tab allows the user to toggle on or off optional ports.

10.1.3 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

11

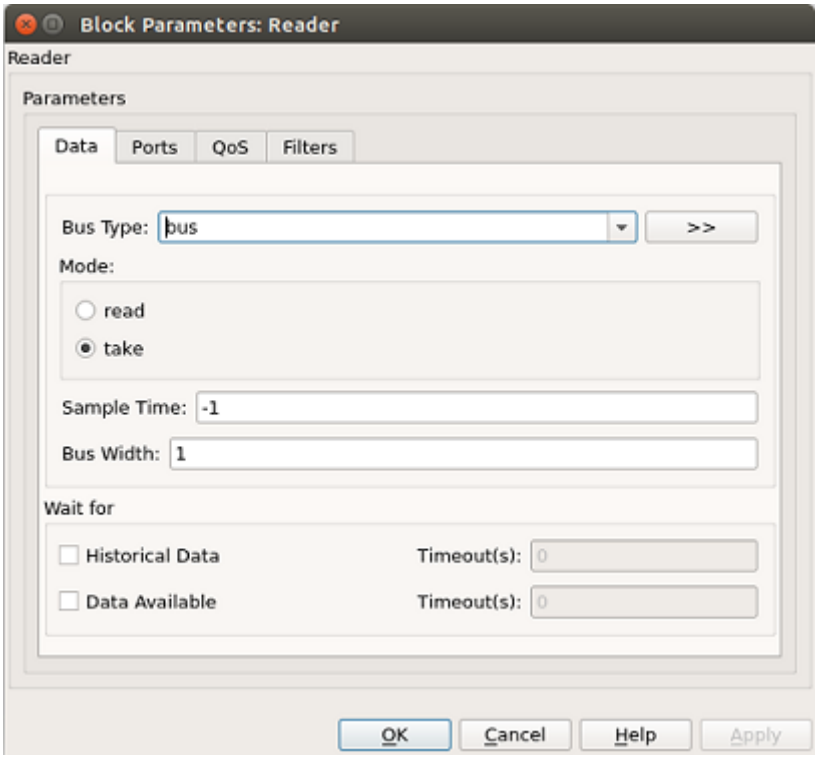
Reader Block

The Reader block represents a DDS data reader entity.

In DDS - “To access the received data, the application must use a typed DataReader attached to the subscriber.”

Port Type	Optional	Name	Description	Output consumed by
Input	yes	psub	DDS Subscriber entity instance	
Input	no	topic	DDS Topic entity instance	
Output	yes	status	0 for successful reader creation	
Output	no	data	BUS	user
Output	yes	info	BUS	user
Output	yes	samples read	Number of samples read	user

11.1 Reader Block Parameters



11.1.1 Data Tab

The **Data** tab is used to set:

- Bus Type

  The output data type (BUS) for the **data** output port

- Mode: take or read

  Specify whether the reader block is accessing the samples using DDS take or DDS read.

- Sample Time

  "The sample time of a block is a parameter that indicates when, during simulation, the block produces outputs and if appropriate, updates its internal state." -Simulink documentation

  Default is -1, meaning it will inherit the Simulink sample time from inputs or the model. Valid values: -1 and Numeric > 0

- Bus Width

  The bus width is the maximum number of samples that can be read or take(n) per block step. Valid values for the bus width are: integers >= 1.

- Wait for

  Checking the Historical Data field in the Wait for section specifies that the Reader will wait for historical data to arrive. The Timeout field is for setting time period (in seconds) determining how long the Reader should wait for the historical data. If the timeout is reached, then any remaining historical data may be interleaved with new data.

  The Data Available field is for specifying whether the Reader should read only if the data is available. The following Timeouts field determines how long the Reader should wait for the availability of data. If the timeout is reached, then the block returns no data and the simulation continues.

11.1.2 Ports Tab

The **Ports** tab allows the user to toggle on or off optional ports.

11.1.3 QoS Tab

The **QoS** tab is used to set the QoS profile. By default, the OSPL default profile is used.

In DDS - The Data-Distribution Service (DDS) relies on the usage of QoS. A QoS (Quality of Service) is a set of characteristics that controls some aspect of the behavior of the DDS Service.

Each DDS block has an associated QoS profile. By default, the OSPL default profile is used. An XML file that specifies QoS profiles can be used to set the QoS of a DDS block.

11.1.4 Filters Tab

The filters tab allows for the filtering of incoming samples. The filtering can happen based on a query and/or on a sample read condition(s).

Query

Expression: The expression is a SQL condition.

Parameters: N parameters in the format {'a', 'b'} Each parameter element must be a char array (string).

Note: Query expressions are only validated at runtime.

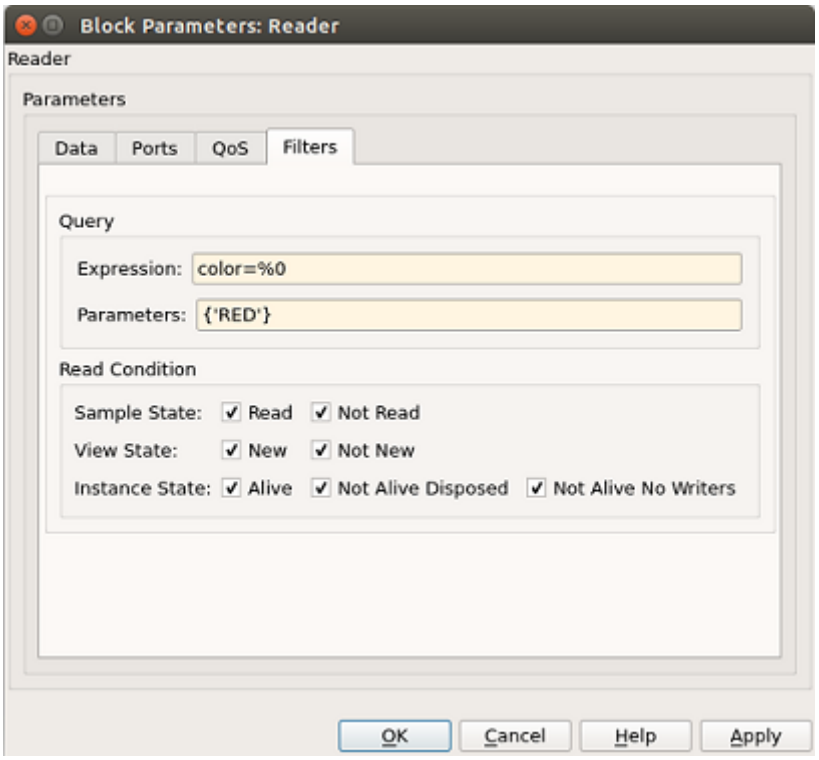
Read Condition

The read conditions specified will filter the samples that are read or take(n).

Example: For a reader, the Sample State has **Read** selected and **Not Read** deselected.

Only samples with a Sample State **Read** will be processed with read or take. Any samples with the **Not Read** sample state will not be read or take(n).

Note: At least one read condition must be selected for each category of Sample State, View State, or Instance State. If not, an error will be thrown when a diagram simulation is run.



12

Tutorial

To demonstrate the capabilities of the DDS Simulink Integration, this tutorial will create two Simulink models. One model will write DDS samples, and the second model will read the DDS samples.

Both models will be run simultaneously, and use a DDS system for communication.

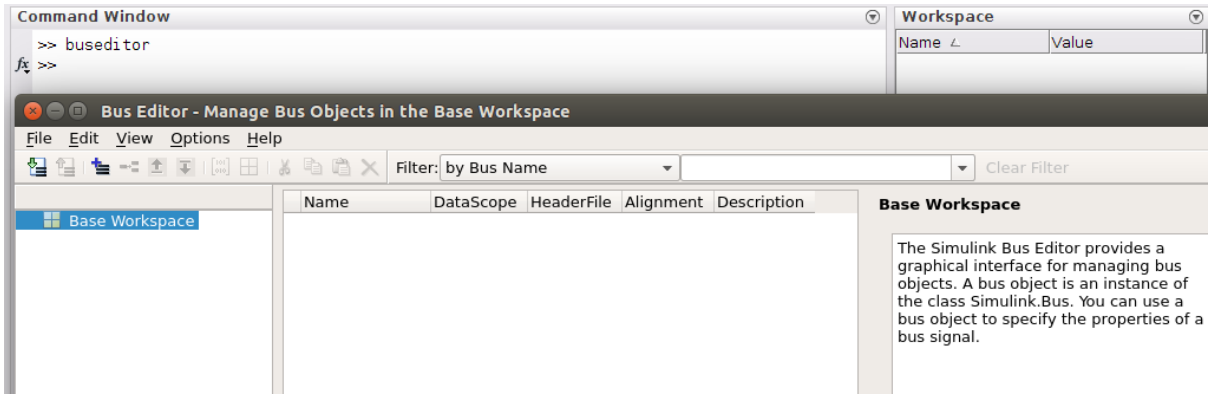
A Simulink bus named ShapeType is created as part of this tutorial. The bus can be created by either using the Simulink bus editor, or by generation from an IDL file. Both options are covered in this tutorial.

12.1 Create ShapeType Bus Using Simulink Bus Editor

Buses define the data which will be published and read. Both the read and write Simulink models will make use of a bus to read and write sample data.

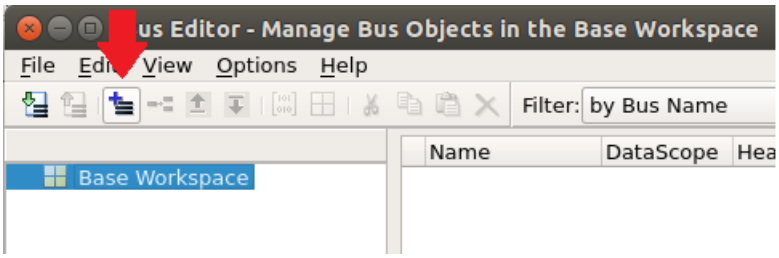
For this tutorial, we will create a BUS named **ShapeType**.

12.1.1 Open the bus editor from the MATLAB command window



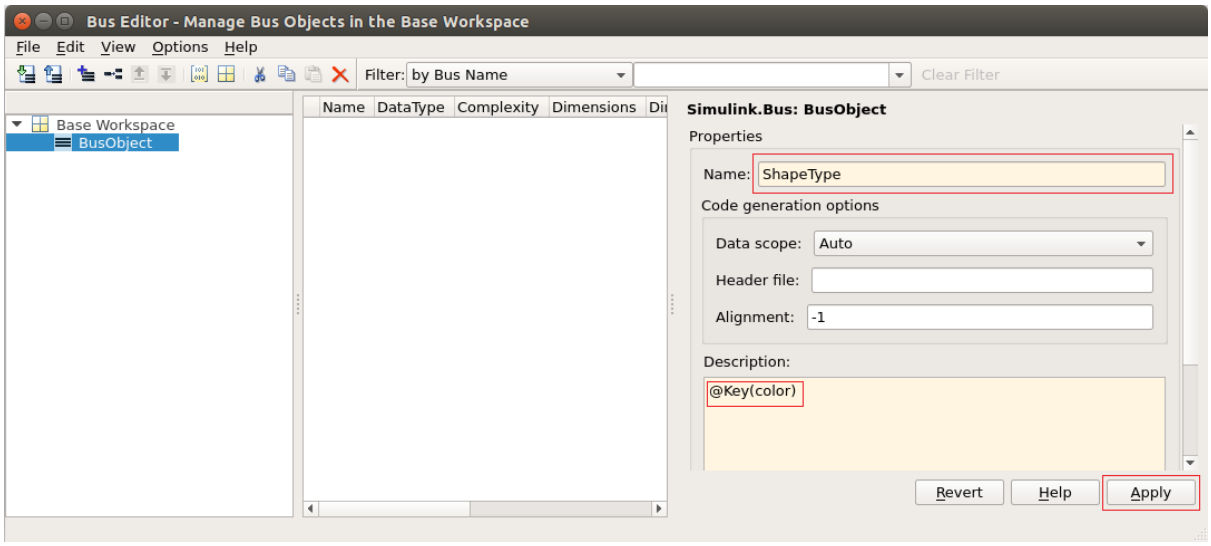
12.1.2 Add a new BUS named ShapeType

12.1.2.1 Select Add Bus button



12.1.2.2 Set Bus name and Key

- Set the new bus name to: **ShapeType**.
- Set Description to: @Key(color). This sets the Topic key.
- Select **Apply** button to save.

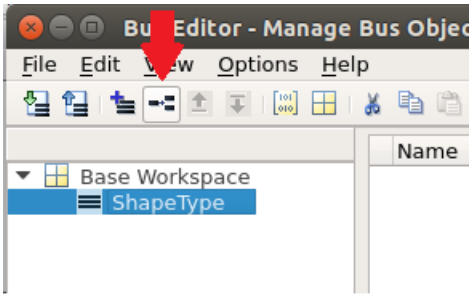


Note: If the Key is not set, the topic block's Key annotation in the model will be shown empty and it will result in a keyless topic. Keyless topics have only one instance.

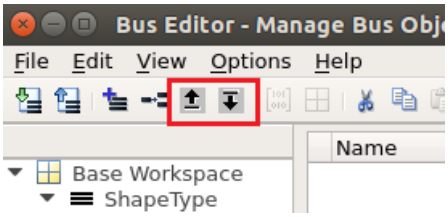
12.1.3 Add BusElements

The **ShapeType** bus will have 4 bus elements: color, x, y and shapesize.

A BusElement can be added to the ShapeType bus by selecting the **Add/Insert BusElement** button.

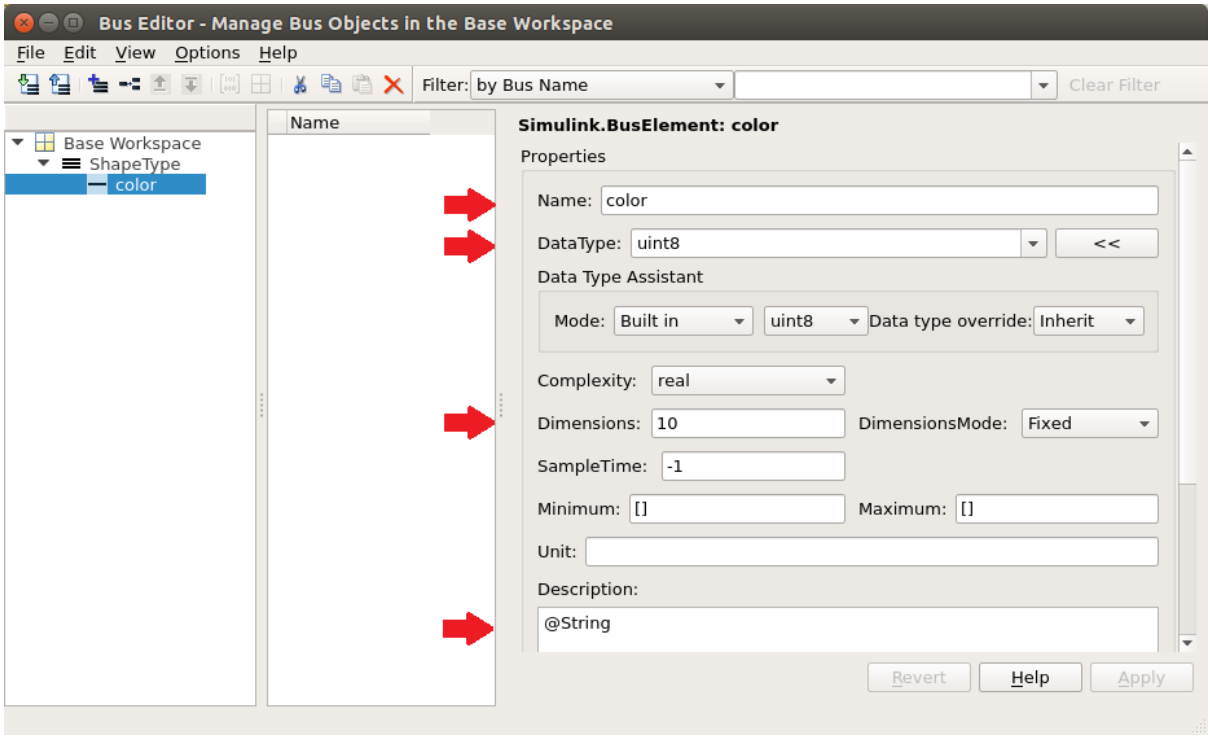


A BusElement can be moved up or down using the **Move Element Up** and **Move Element Down** buttons.



12.1.3.1 Add color

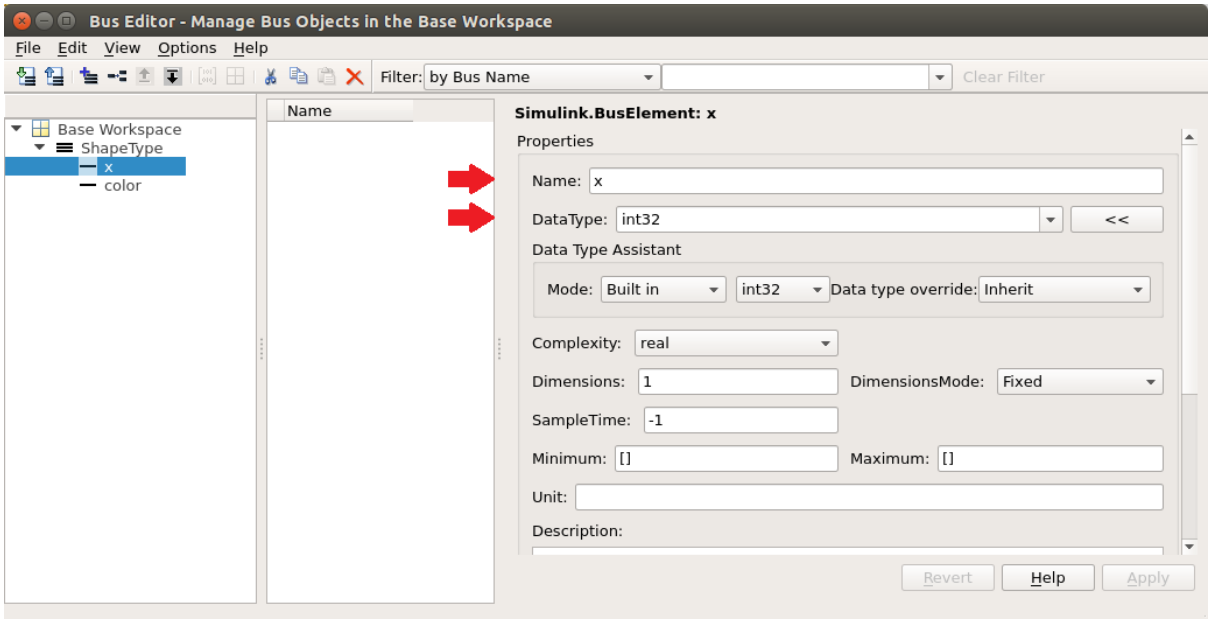
- Select **Add/Insert BusElement** button
- Set name to: color
- Set DataType to: uint8
- Set Dimensions to: 10
- Set Description to: @String
- Select **Apply** to save



Note: This creates a DDS 'string' type. Bus elements of type 'int8' or 'uint8' with an annotation of @String in the Description field define a DDS string. Dimension set to 10 means Simulink is going to read only the first ten bytes of the string. You can also use the '@BString' annotation to define a DDS bounded string. The dimension of the field is treated as the maximum string size.

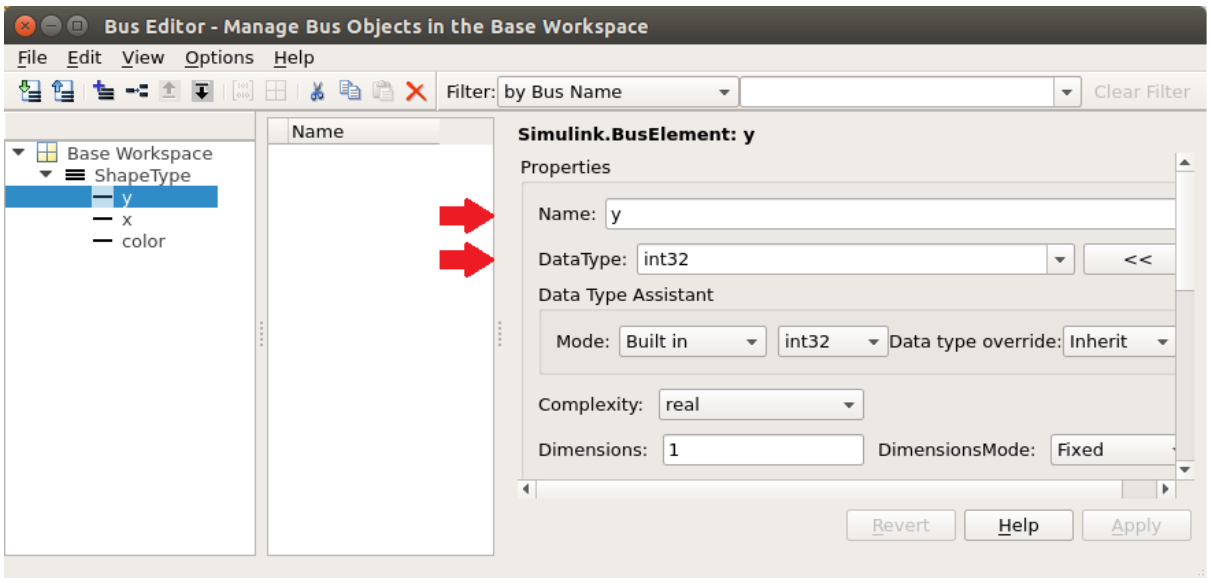
12.1.3.2 Add x

- Select **Add/Insert BusElement** button
- Set name to: x
- Set DataType to: int32
- Select **Apply** to save



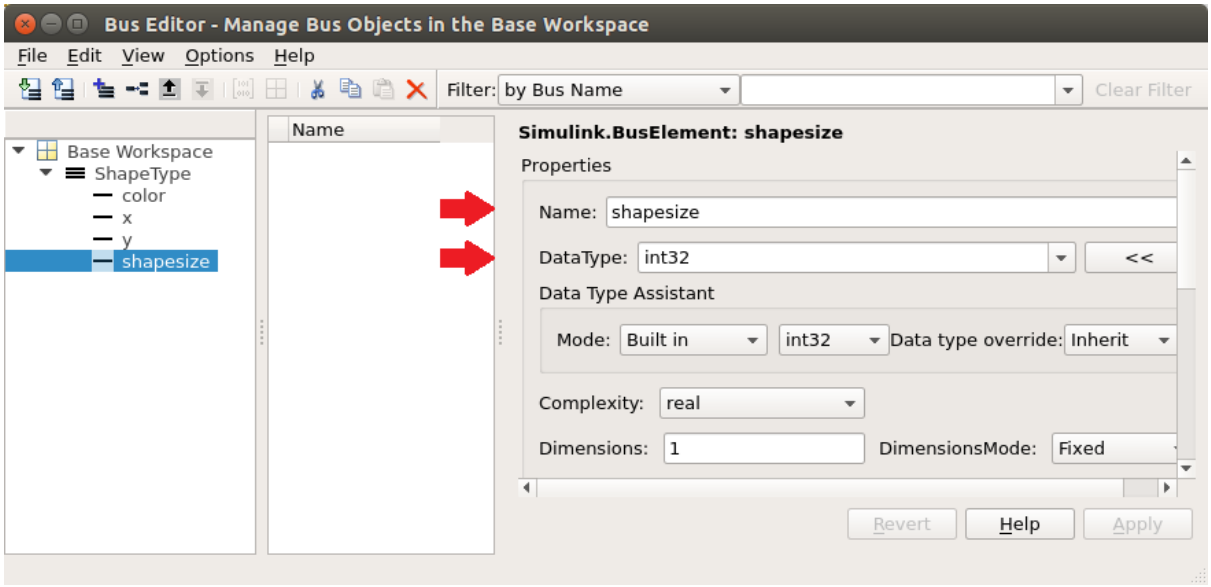
12.1.3.3 Add y

- Select **Add/Insert BusElement** button
- Set name to: y
- Set DataType to: int32
- Select **Apply** to save



12.1.3.4 Add shapesize

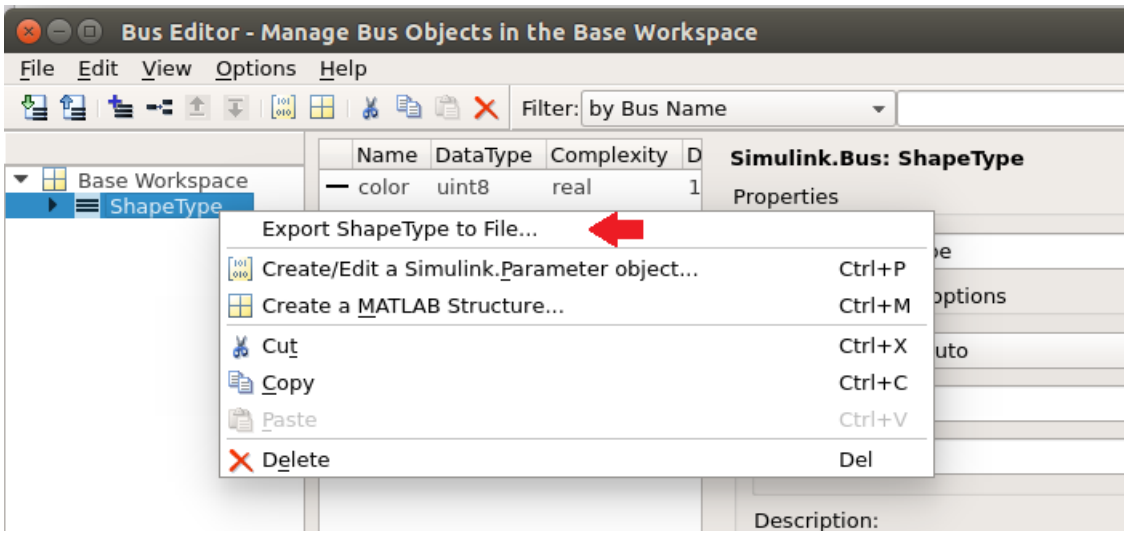
- Select **Add/Insert BusElement** button
- Set name to: shapesize
- Set DataType to: int32
- Select **Apply** to save



12.1.4 Export BUS objects

When bus objects are added to the MATLAB workspace, they will be lost on MATLAB close or workspace clear. To persist the bus objects, they can be exported.

A quick way to export the **ShapeType** bus using the bus editor, is to right click on the bus and select **Export ShapeType to File...**



The **ShapeType** bus is complete.

12.2 Create ShapeType Using IDL

The public Vortex.idlImportSl function can be called to generate Simulink bus definitions from an IDL file. The generated bus definitions are inserted into the 'Design Data' section of a data dictionary.

From the Simulink documentation - *"A data dictionary is a persistent repository of data that are relevant to your model. You can also use the base workspace to store design data that are used by your model during simulation."*

The data dictionary can then be referenced from your models.

12.2.1 Create IDL File

Create an IDL file to define your ShapeType topic structure. For this tutorial we will name the file *ShapeType.idl*.

```
 struct ShapeType {
   string color; //@Key
   long x;
   long y;
   long shapesize;
 };
#pragma keylist ShapeType color
```

IMPORTANT NOTE: The IDL file has to have a blank line after the pragma keylist declaration. (known bug)

12.2.2 Generate Simulink bus definitions from an IDL file

Steps:

1. In MATLAB, navigate to the directory that contains the *ShapeType.idl* file. Set this directory to be the MATLAB **Current Folder**.
2. Call the idlImportSl function in the MATLAB command window.

   >> Vortex.idlImportSl('ShapeType.idl', 'shape.sldd')

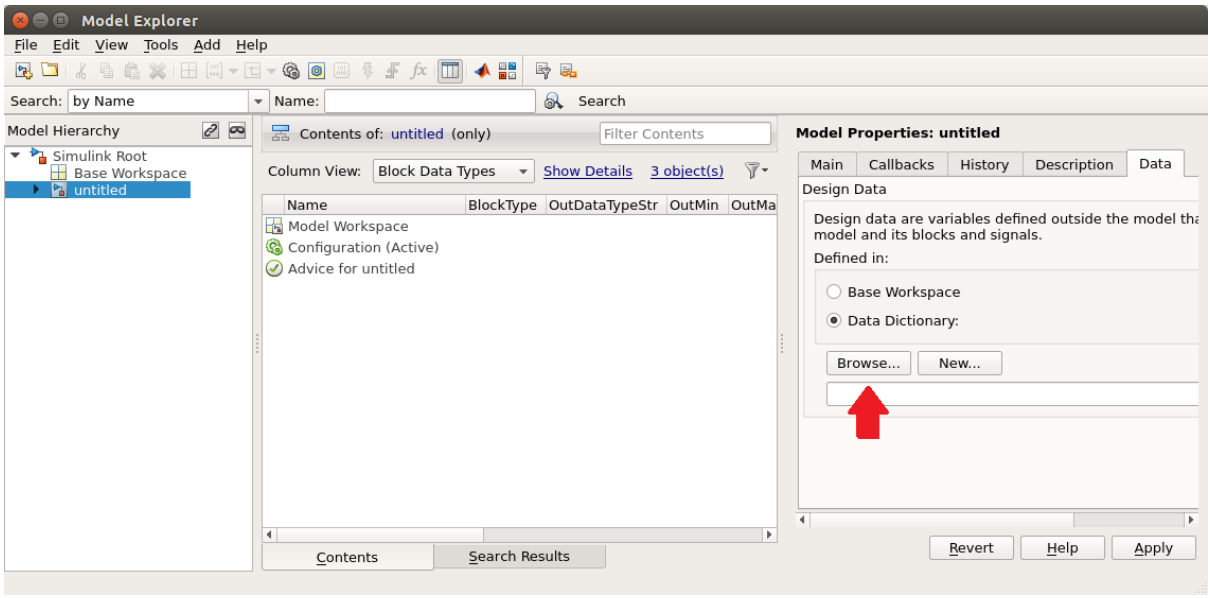
   where:

   'ShapeType.idl' is the name of the IDL file

   'shape.sldd' is the name of the target data dictionary for the generated bus definitions

12.2.3 Model Explorer

To make use of the bus definitions generated into the data dictionary, Simulink models can specify design data using the **Model Explorer**.



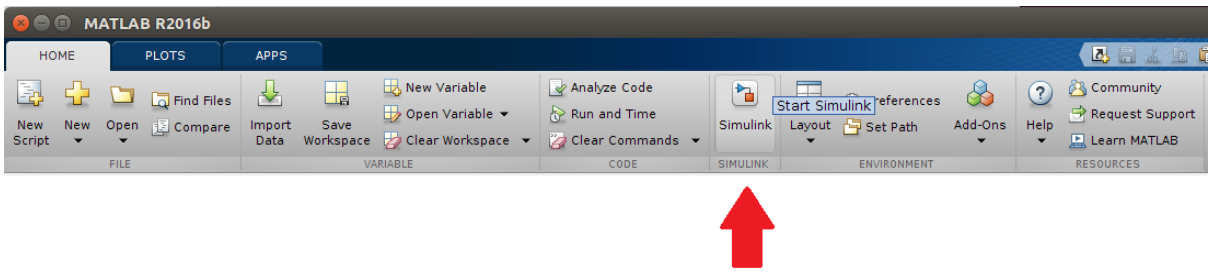
12.3 Shapes Write Model

This section outlines how to create a new DDS Simulink model that will write sample data for the topic type **ShapeType**.

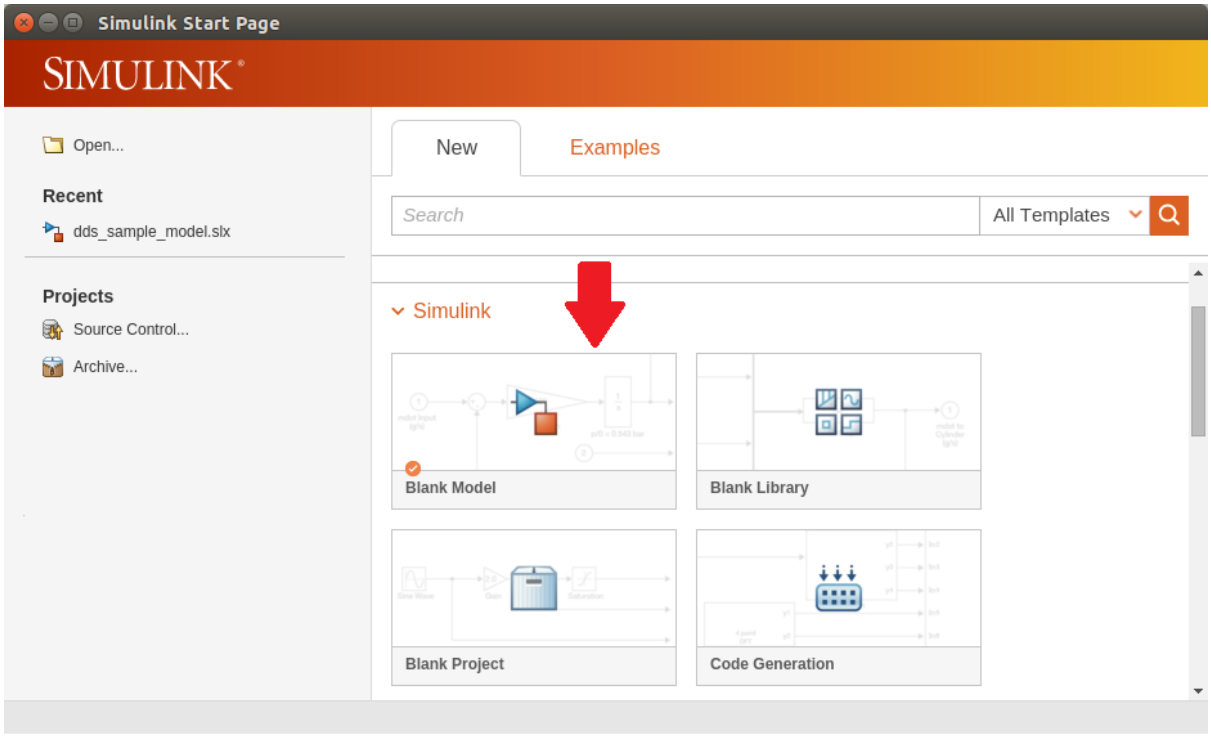
Although not necessary, this model will use the optional Domain and Publisher blocks.

12.3.1 Create a new Simulink model

12.3.1.1 Start Simulink



12.3.1.2 Add a new blank model

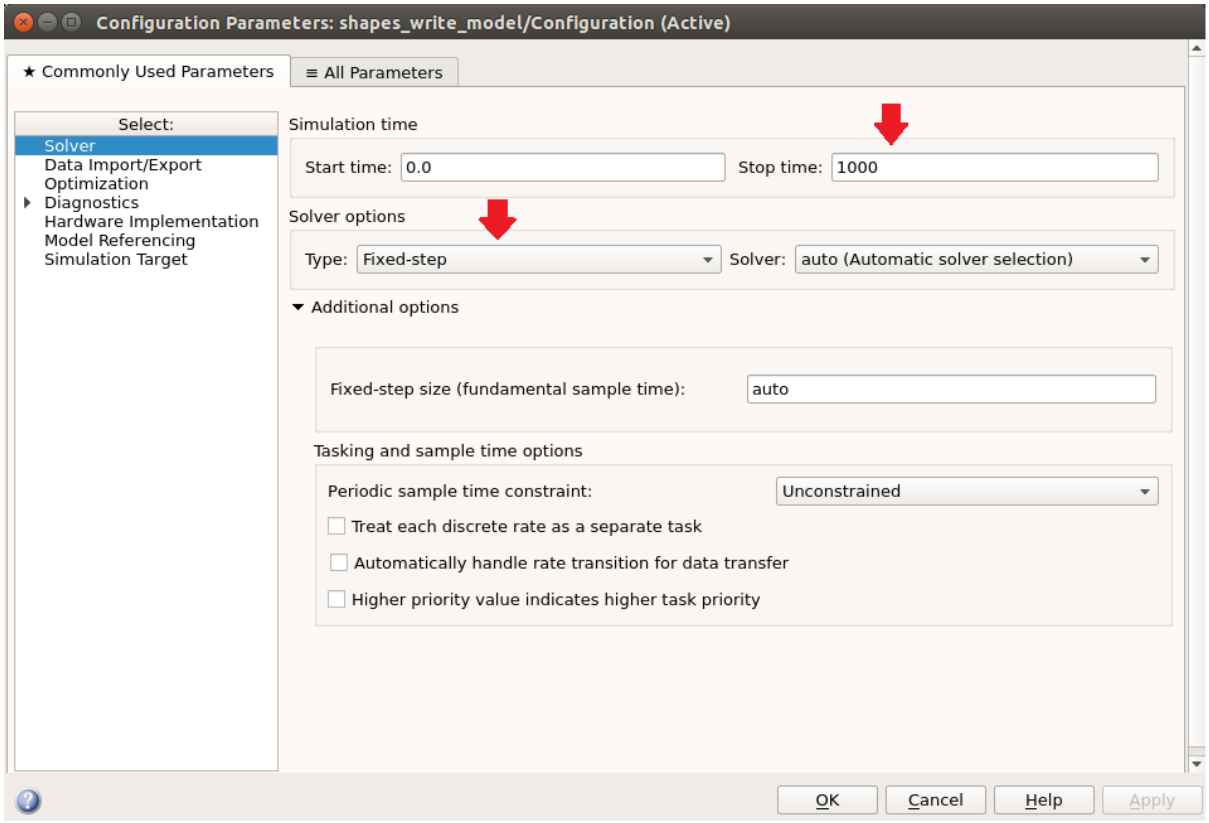


12.3.1.3 Save As...

- Save the model as *"shapes_write_model.slx"*.

12.3.1.4 Model Settings

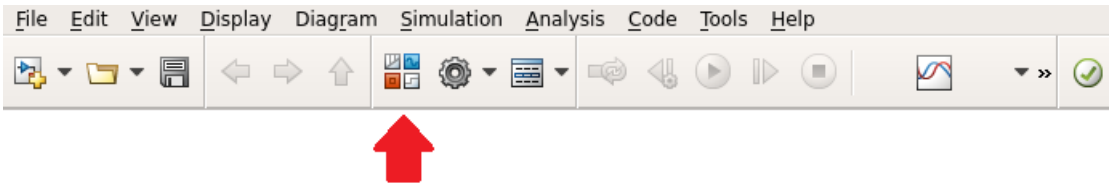
- Open Model Configuration Parameters dialog, by selecting menu **Simulation / Model Configuration Parameters**.
- Set the simulation stop time to 1000.0 seconds. (Note: "Simulation time is not the same as clock time. For example, running a simulation for 10 seconds usually does not take 10 seconds. Total simulation time depends on factors such as model complexity, solver step sizes, and computer speed." Simulink Help documentation)
- Set the Solver Type to Fixed-step.

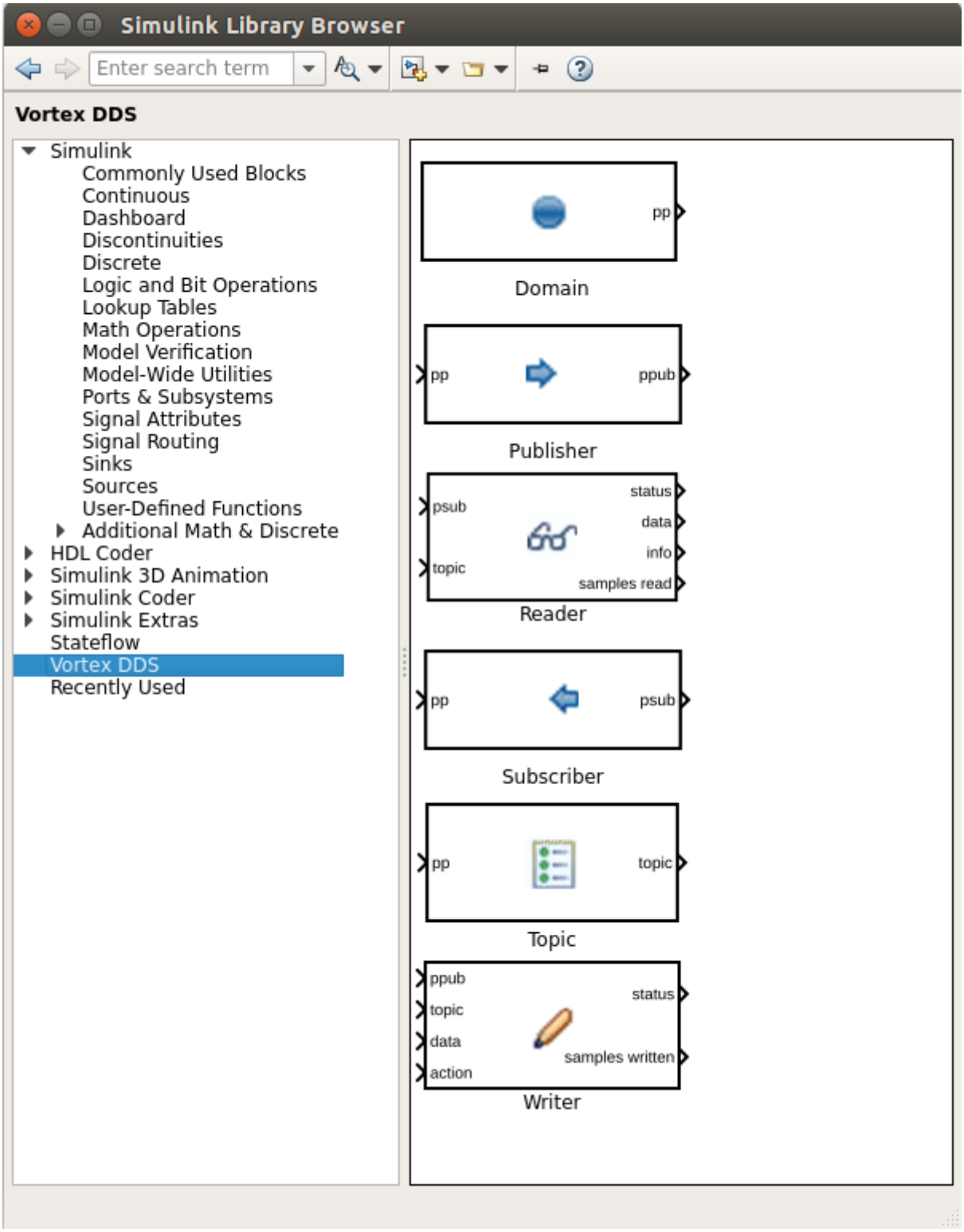


12.3.2 Add Simulink DDS Blocks

12.3.2.1 Open the Simulink Library Browser

- Open the Simulink Library Browser
- Browse to and select **Vortex DDS** to view DDS custom blocks



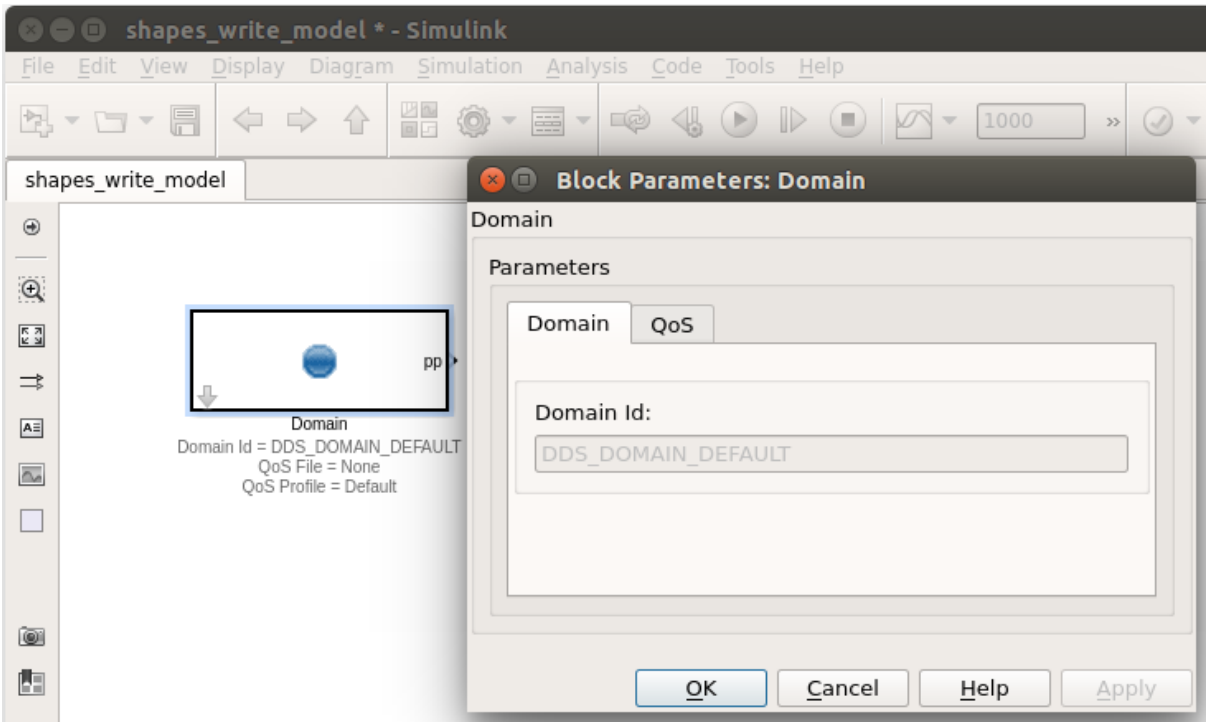

Simulink Library Browser
Enter search term
Vortex DDS
Simulink
Commonly Used Blocks
Continuous
Dashboard
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functions
Additional Math & Discrete
HDL Coder
Simulink 3D Animation
Simulink Coder
Simulink Extras
Stateflow
Vortex DDS
Recently Used
pp
Domain
pp
ppub
Publisher
psub
topic
status
data
info
samples read
Reader
pp
psub
Subscriber
pp
topic
Topic
ppub
topic
data
action
status
samples written
Writer

12.3.2.2 Add a Domain block

- Drag a Domain participant block from the Simulink Library Browser onto the Simulink model diagram.

12.3.2.3 Set domain block properties

- To set a block's parameters, double click on the block to bring up the **Block Parameters** dialog.
- The domain id is read only and set to DDS_DOMAIN_DEFAULT. This default is specified in the OSPL configuration file.
- The **QoS** tab defaults to the OSPL defaults.
- For this tutorial example, we are going to use the defaults, therefore no block parameters need to be specified.



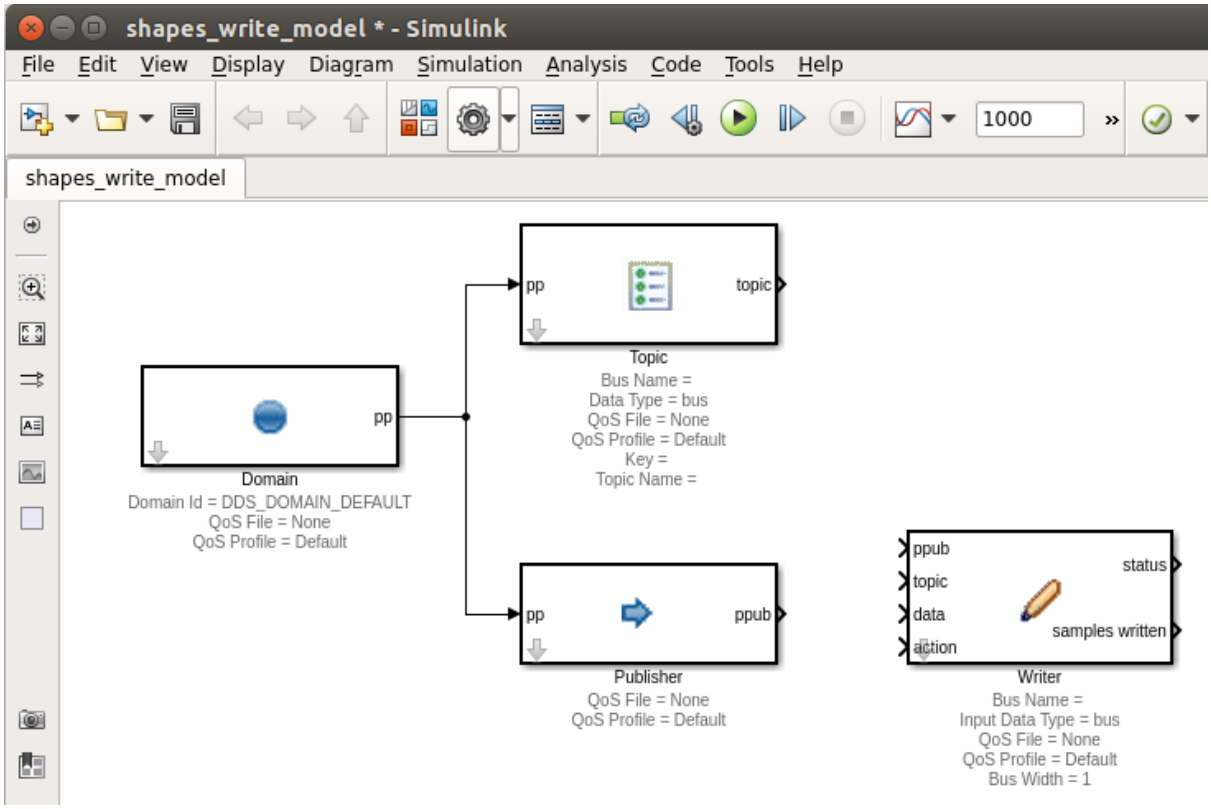
12.3.2.4 Add blocks (Topic, Publisher, and Writer)

Using the Simulink Library Browser drag the following block types onto your diagram:

- 1 Topic
- 1 Publisher
- 1 Writer

12.3.2.5 Connect Domain to Topic and Publisher

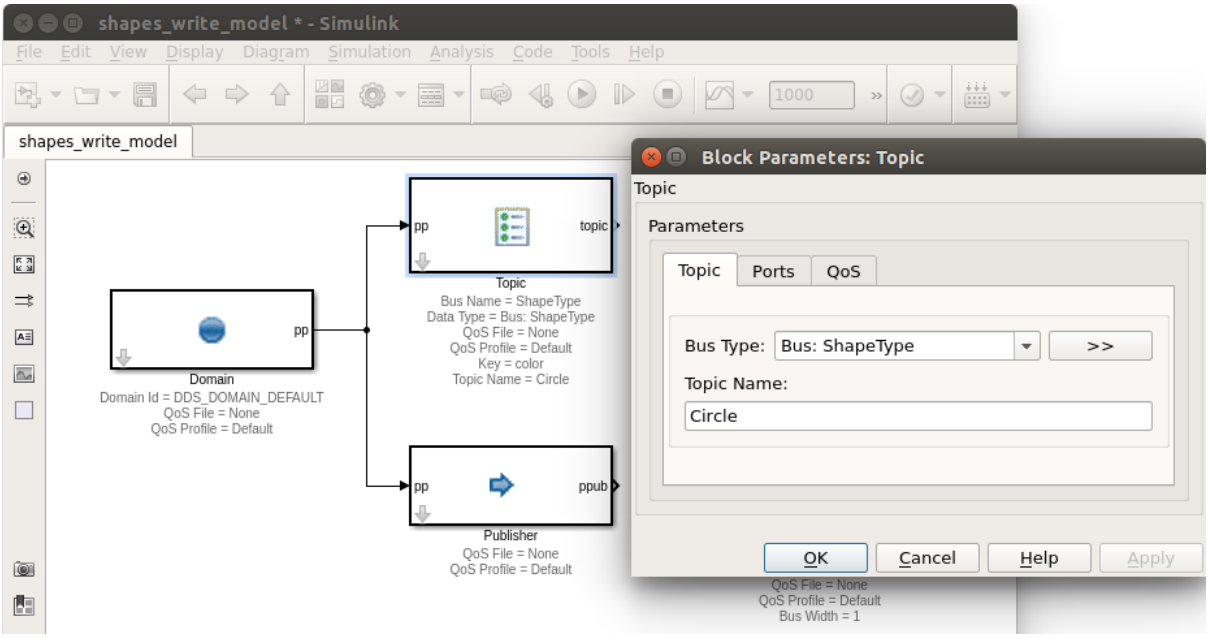
- Connect the Domain **pp** output to Topic **pp** input.
- Connect the Domain **pp** output to Publisher **pp** input.



12.3.2.6 Set Topic Block Parameters

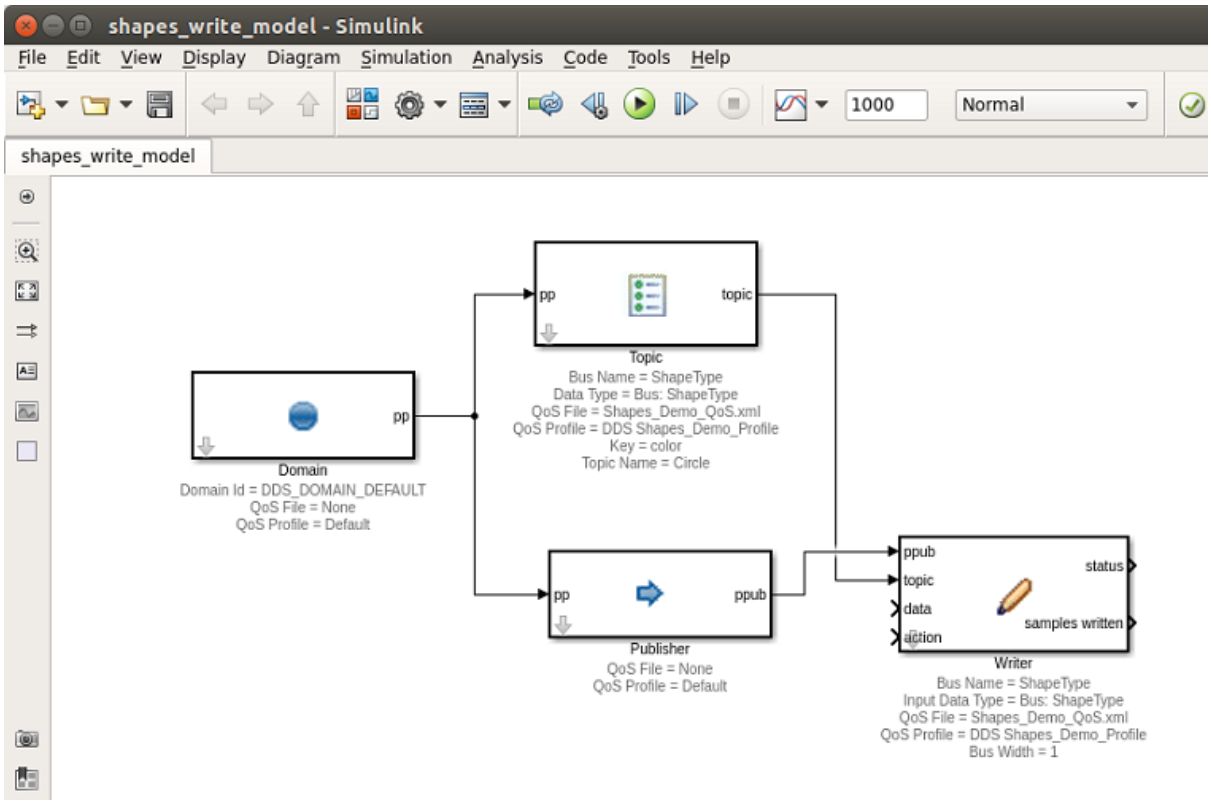
- Double click on the Topic to bring up the **Block Parameters** dialog.
- **In the Topic tab:**
  - Set **Bus Type** to: ShapeType
  - Set **Topic Name** to: Circle
- The **Ports** tab allows for the setting of optional ports. For this model, we will not change the defaults.
- Select the **QoS** tab.
- **Set the QoS file to: Shapes_Demo_QoS.xml.**

  /INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux/tools/matlab/
  /examples/simulink/dds_reader_writer_model/Shapes_Demo_QoS.xml



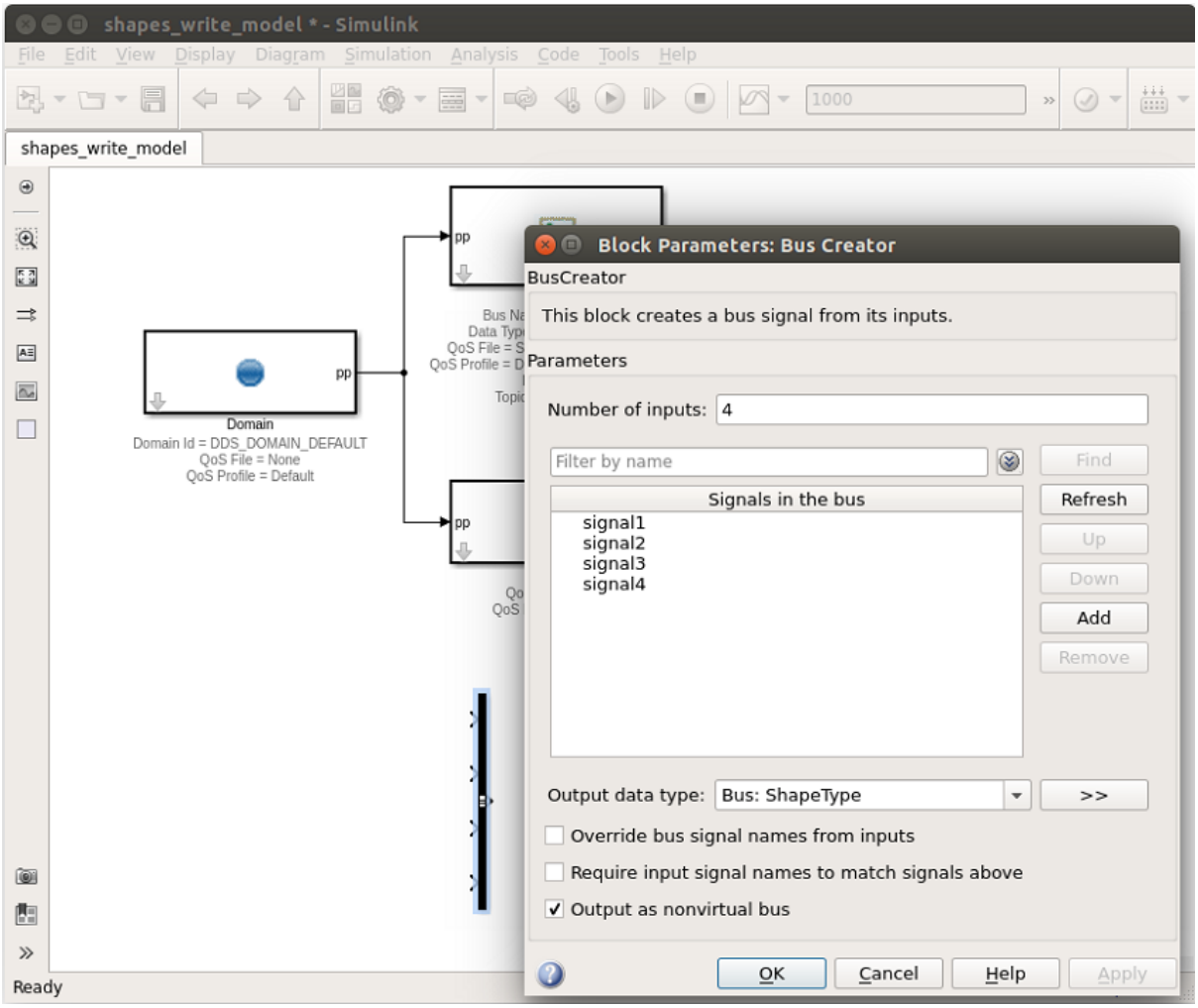
12.3.2.7 Connect Topic, Publisher and Writer Blocks & Set Writer Block Parameters

- Connect the Topic **topic** output to the Writer **topic** input.
- Connect the Publisher **ppub** output to the Writer **ppub** input.
- Double click on the Writer block to edit the **Block Parameters**. Set the **Input Data Type** to the bus: ShapeType.
- Select the Writer **QoS** tab.
- Set the QoS file to: Shapes_Demo_QoS.xml.
  - **/INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux/tools/matlab/**
    /examples/simulink/dds_reader_writer_model/Shapes_Demo_QoS.xml



12.3.2.8 Add a Bus Creator to Set Sample Data

To generate sample data, we will add a **Simulink / Signal Routing / BusCreator** block to our diagram.

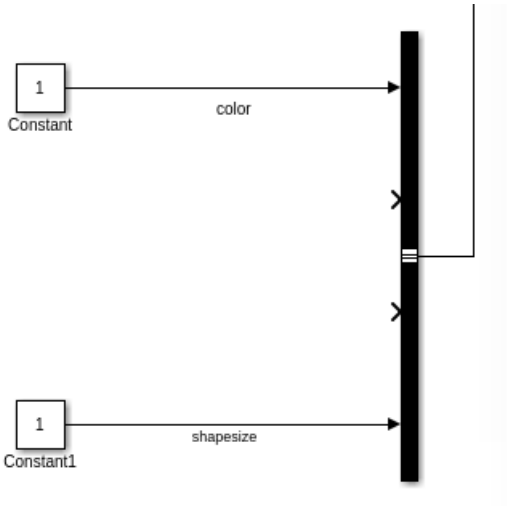


12.3.2.9 Add Bus Creator Inputs

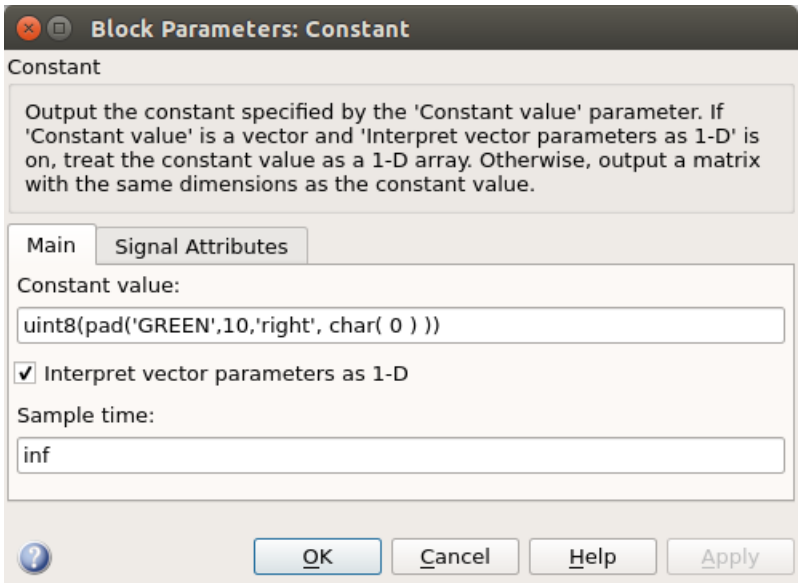
For demonstration purposes, we will input to the **BusCreator** signals using **Constant**, **Clock** and **Sine Wave** blocks.

Note: To change the positioning of block ports, you can use the **Rotate & Flip** block menu item, accessible by right clicking on a block.

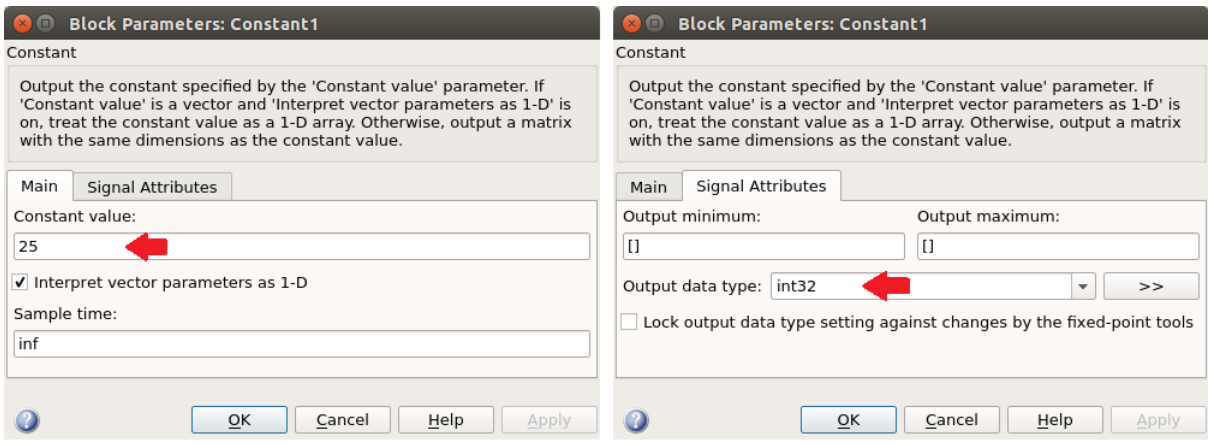
- Drag 2 **Simulink / Sources / Constant** blocks onto the diagram
- Connect one **Constant** block to color input signal
- Connect one **Constant** block to shapesize input signal



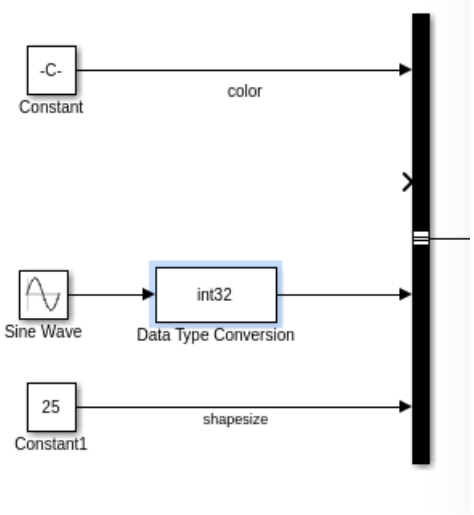
- Set the **Block Parameters** for color signal **Constant** block
- Set **Constant value** to: uint8(pad('GREEN',10,'right', char(0)))
- Select **OK**



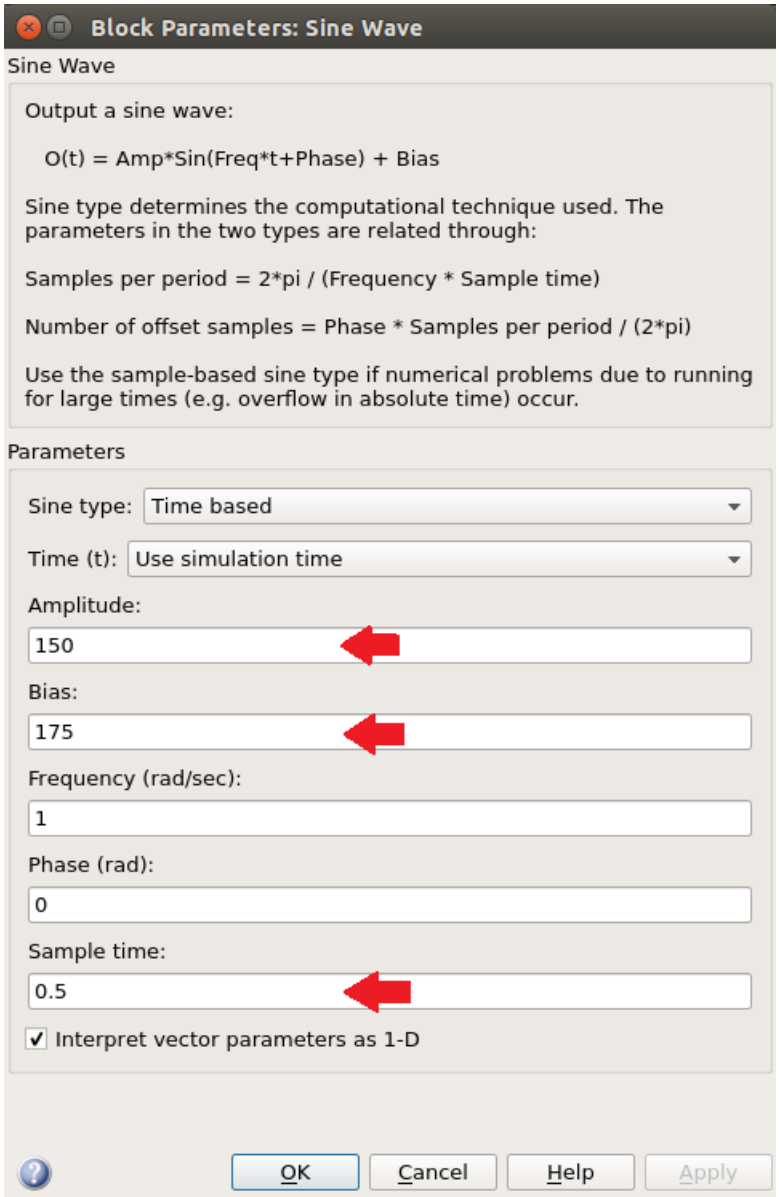
- Set the **Block Parameters** for shapesize signal **Constant** block
- Set **Constant value** to: 25
- Set **Output data type** to: int32



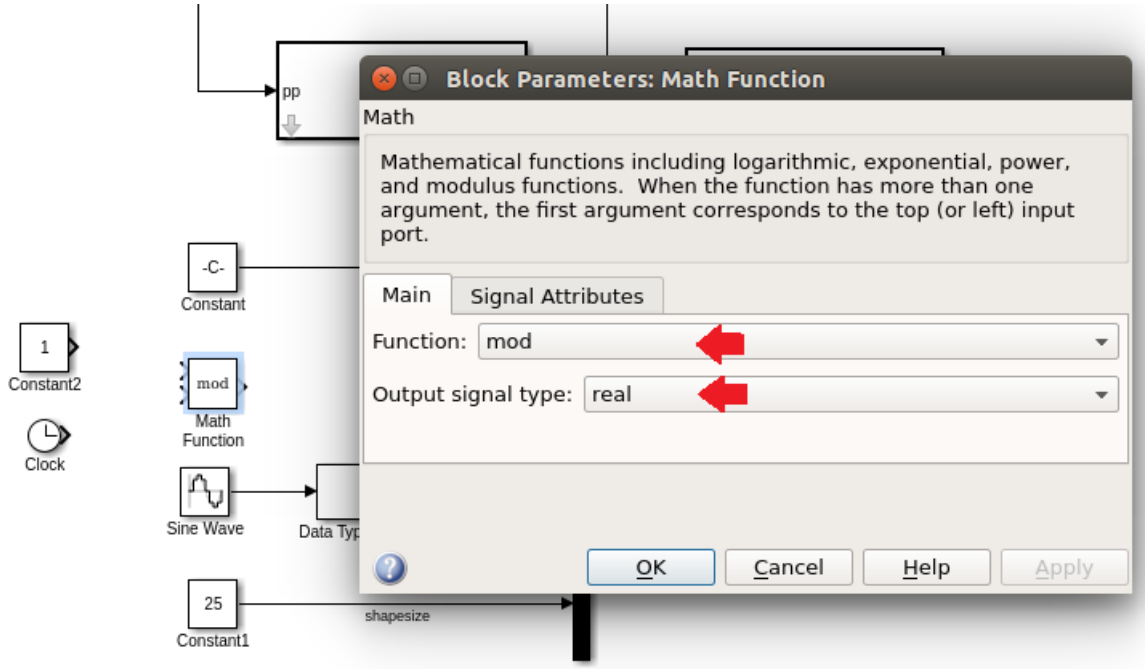
- Drag a **Simulink / Sources / Sine Wave** block onto diagram
- Connect **Sine Wave** block to y input signal
- Drag a **Simulink / Signal Attributes / Data Type Conversion** block onto **Sine Wave** connector.
- Set **Output data type** to: int32



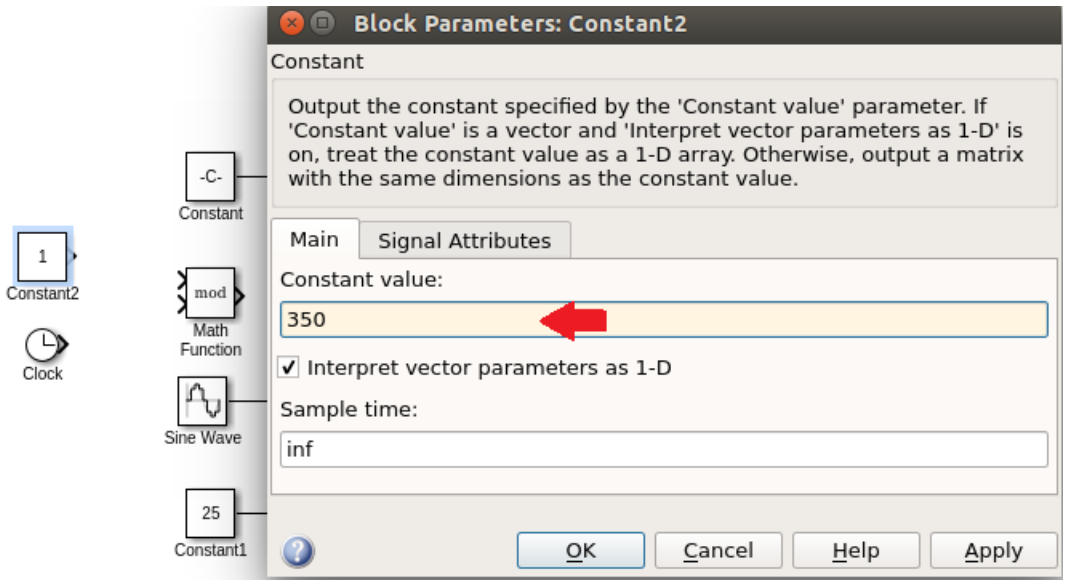
- Set **Block Parameters** for **Sine Wave** block.
- Set **Amplitude** to: 150
- Set **Bias** to: 175
- Set **Sample time** to: 0.5



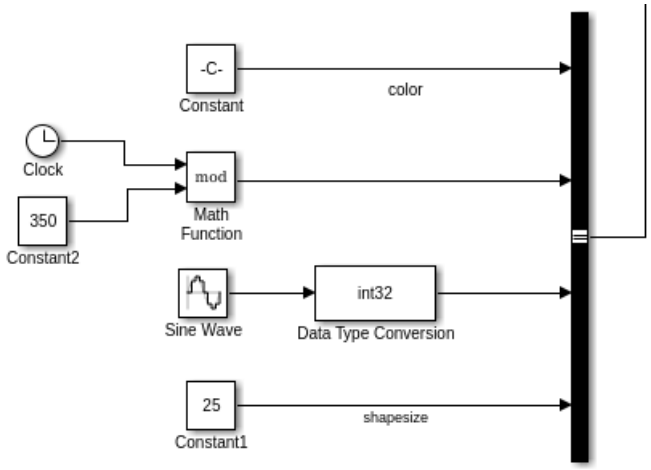
- **For the x input signal, drag 3 new blocks onto diagram:**
  - **Simulink / Sources / Constant**
  - **Simulink / Sources / Clock**
  - **Simulink / Math Operations / Math Function**
- Set **Block Parameters** for **Math Function** block.
- Set **Function** to: mod
- Set **Output signal type** to: real



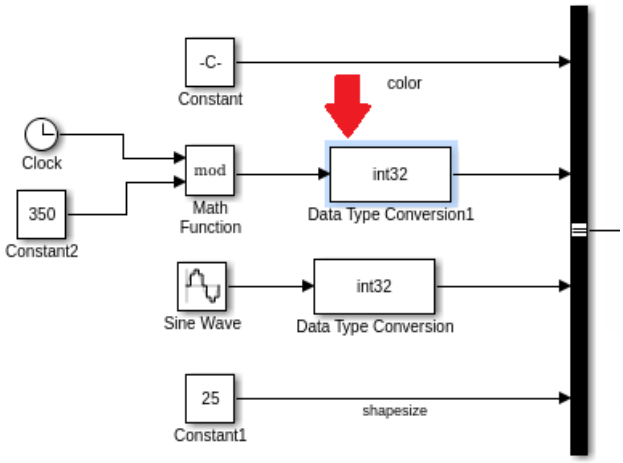
- Set **Block Parameters** for **Constant** block.
- Set **Constant value** to: 350



- Connect the **Clock** and **Constant** blocks to the mod **Math Function**
- Connect the mod **Math Function** to the **BusCreator** x input signal



- Drag a **Simulink / Signal Attributes / Data Type Conversion** block onto **Math Function** connector.
- Set **Output data type** to: int32



Save your model. Your model is now complete!

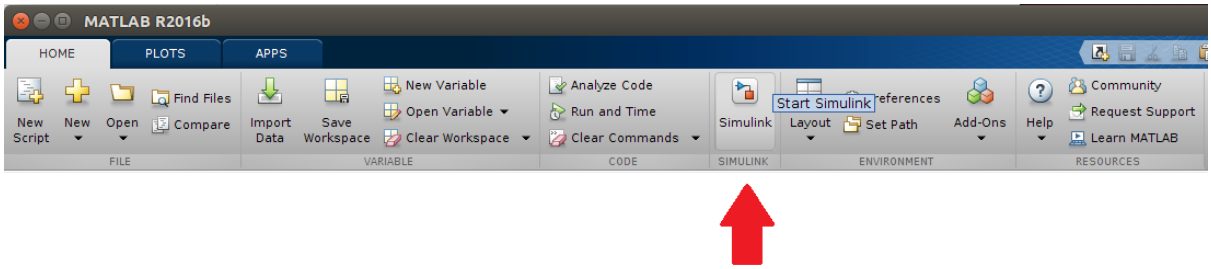
12.4 Shapes Read Model

12.4.1 Create a new Simulink model

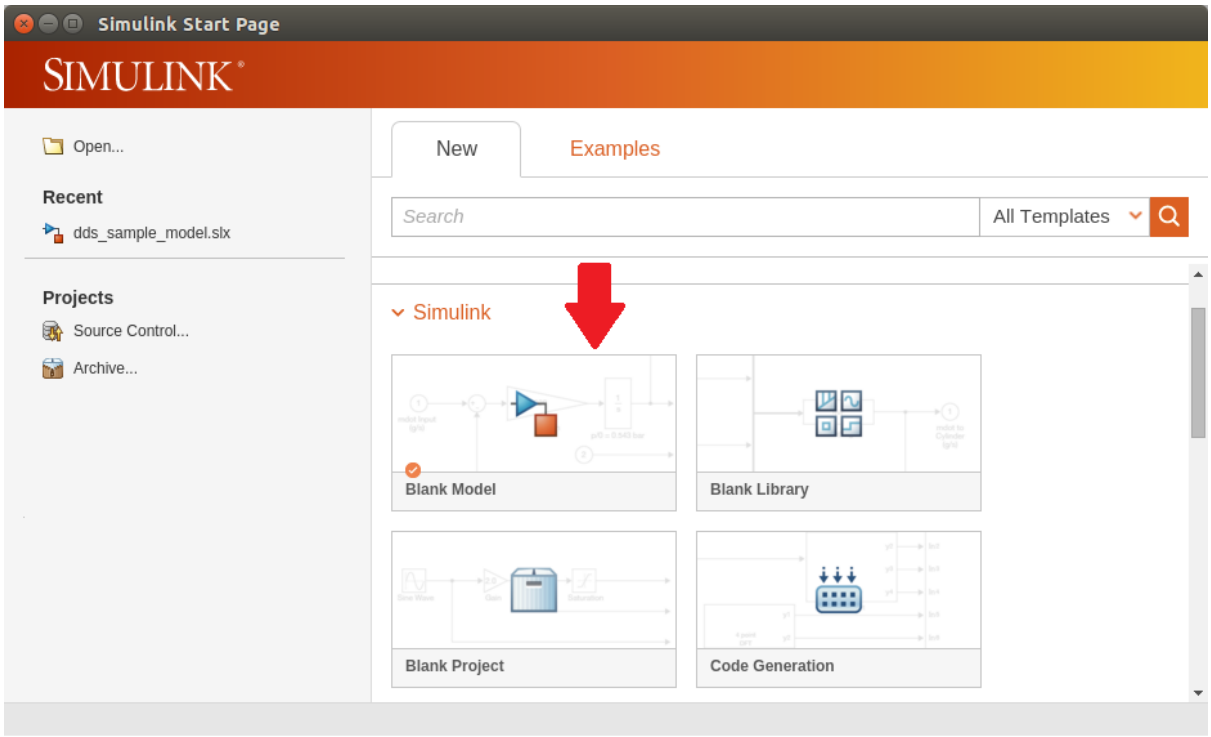
This section outlines how to create a new DDS Simulink model that will read and display sample data for the topic type **ShapeType**.

In this model example, we will be making use of many of the defaults, so the optional blocks will not be included in this model.

12.4.1.1 Start Simulink



12.4.1.2 Add a new blank model

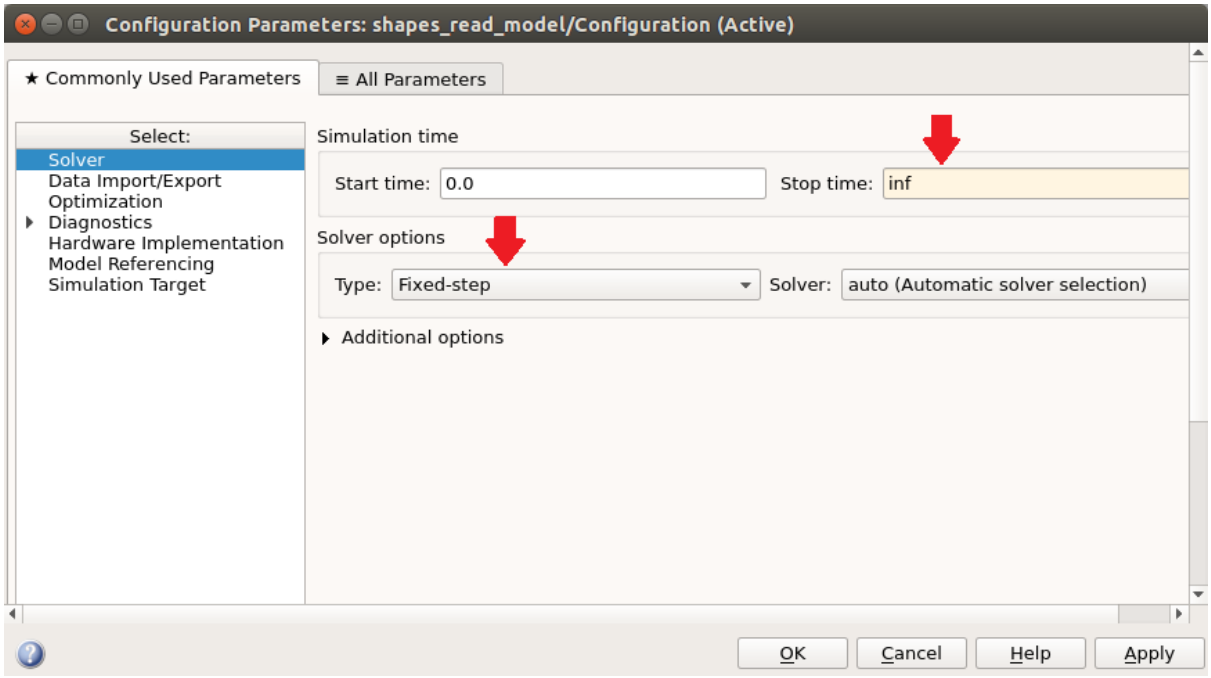


12.4.1.3 Save As...

- Save the model as *"shapes_read_model.slx"*.

12.4.1.4 Model Settings

- Open Model Configuration Parameters dialog, by selecting menu **Simulation / Model Configuration Parameters**.
- Set the simulation stop time to **inf**. (Note: "Specify inf to run a simulation or generated program until you explicitly pause or stop it." Simulink Help documentation)
- Set the Solver Type to **Fixed-step**.



12.4.2 Add Simulink DDS Blocks

12.4.2.1 Open the Simulink Library Browser

12.4.2.2 Add all required blocks (Topic and Reader)

Using the Simulink Library Browser drag the following block types onto your diagram:

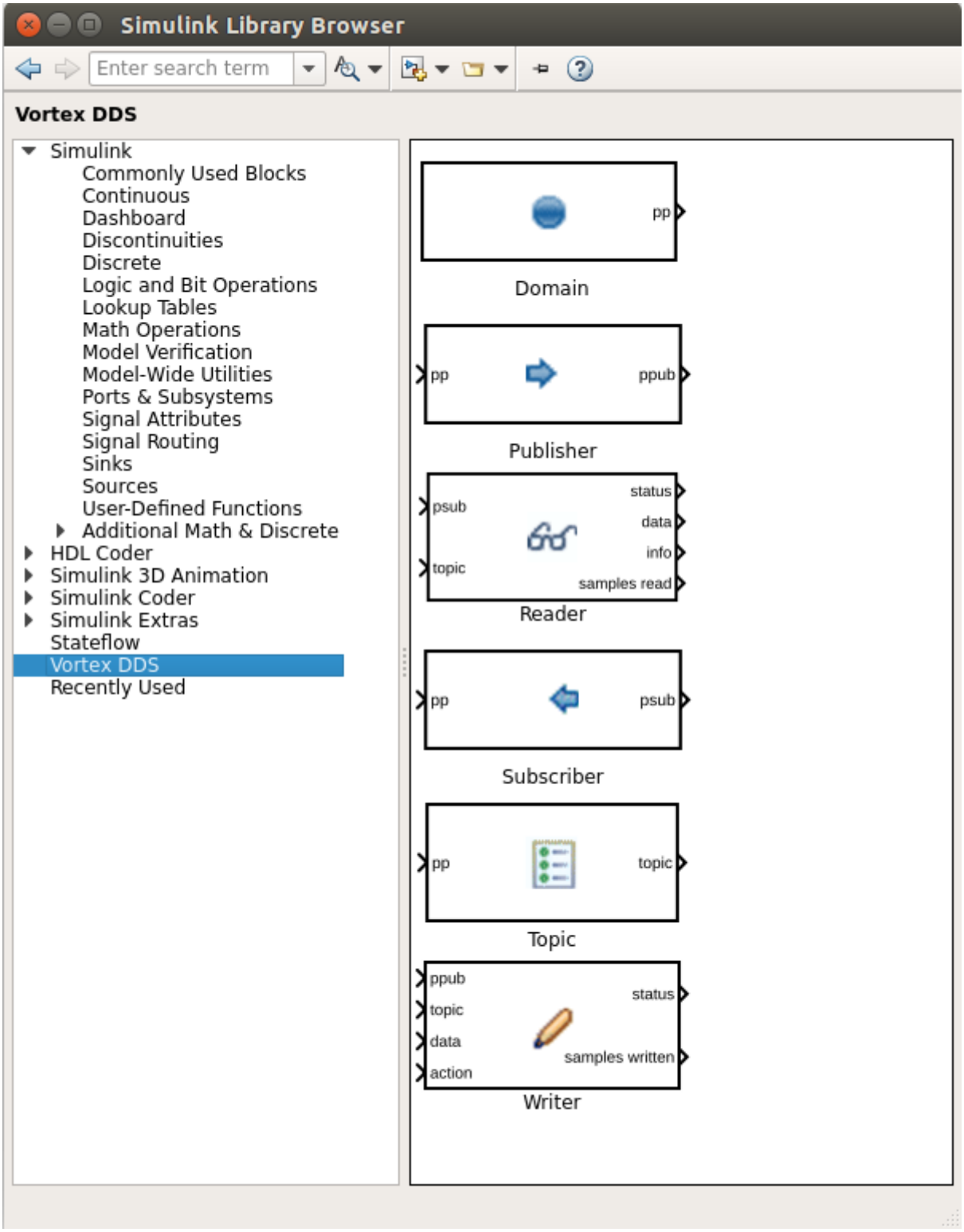
- 1 Topic
- 1 Reader

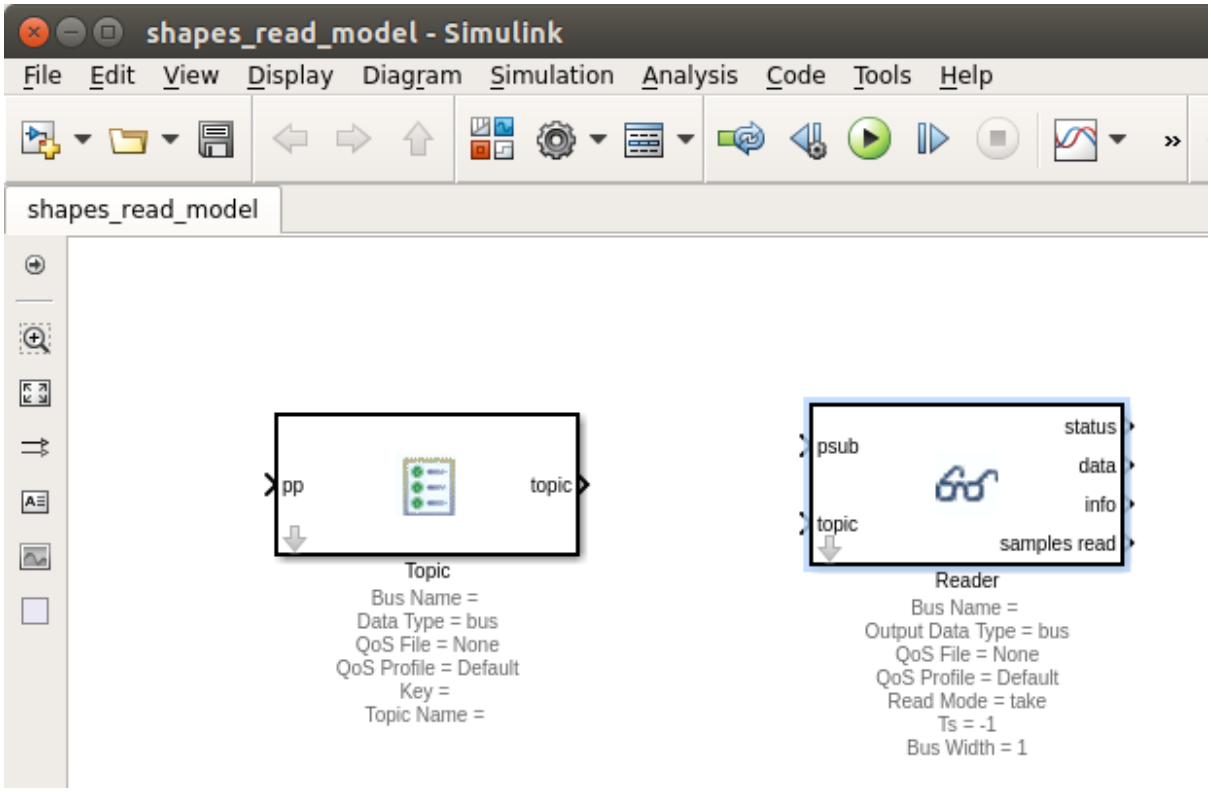
Note: For this example model, we will be using the block defaults for the Domain and Subscriber, therefore they will not be included on the model.

To set a block's parameters, double click on the block to bring up the **Block Parameters** dialog.

12.4.2.3 Toggle off optional ports

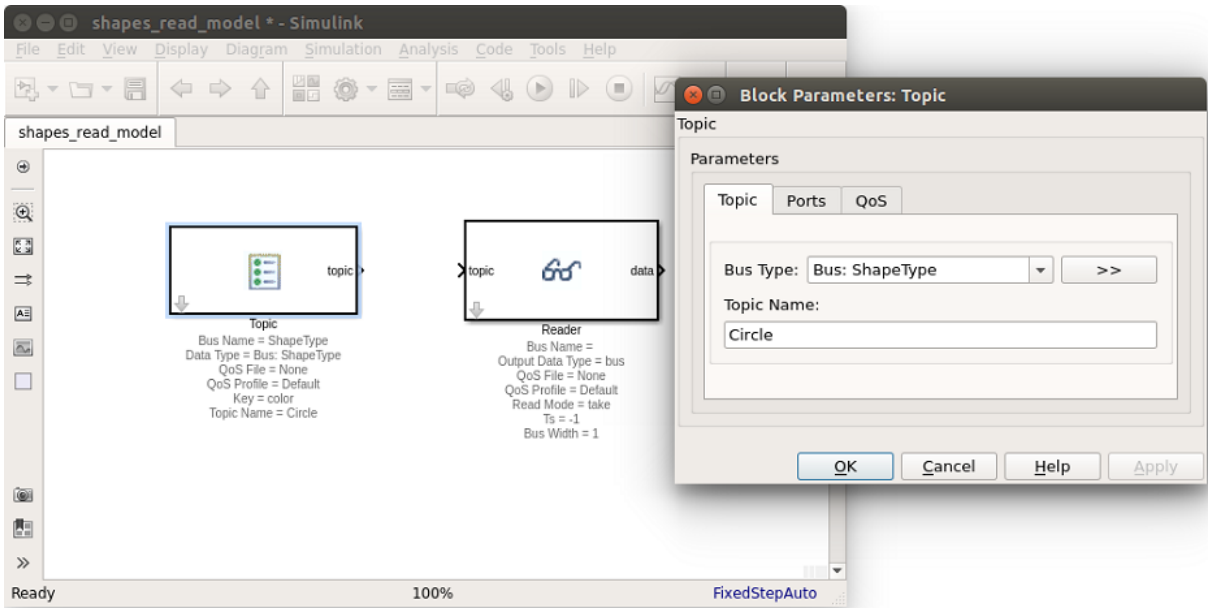
- Double click on the Topic to bring up the **Block Parameters** dialog.
- In the Topic **Ports** tab deselect the Participant port.
- Double click on the Reader to bring up the **Block Parameters** dialog.
- In the Reader **Ports** tab deselect the Subscriber, Reader, Status, Info and Samples Read ports.


Simulink Library Browser
Enter search term
Vortex DDS
Simulink
Commonly Used Blocks
Continuous
Dashboard
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functions
Additional Math & Discrete
HDL Coder
Simulink 3D Animation
Simulink Coder
Simulink Extras
Stateflow
Vortex DDS
Recently Used
pp
Domain
pp
ppub
Publisher
psub
topic
status
data
info
samples read
Reader
pp
psub
Subscriber
pp
topic
Topic
ppub
topic
data
action
status
samples written
Writer



12.4.2.4 Set Topic Block Parameters

- Double click on the Topic to bring up the **Block Parameters** dialog, select **Topic** tab.
- **Set the Bus Type: ShapeType bus** Note: If the ShapeType bus is not displayed, select Refresh data types from dropdown list.
- Set the **Topic Name** to: Circle.



- Select the **QoS** tab.
- **Set the QoS file to** [Shapes_Demo_QoS.xml.]

  INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux/tools/matlab/

examples/simulink/dds_reader_writer_model/Shapes_Demo_QoS.xml

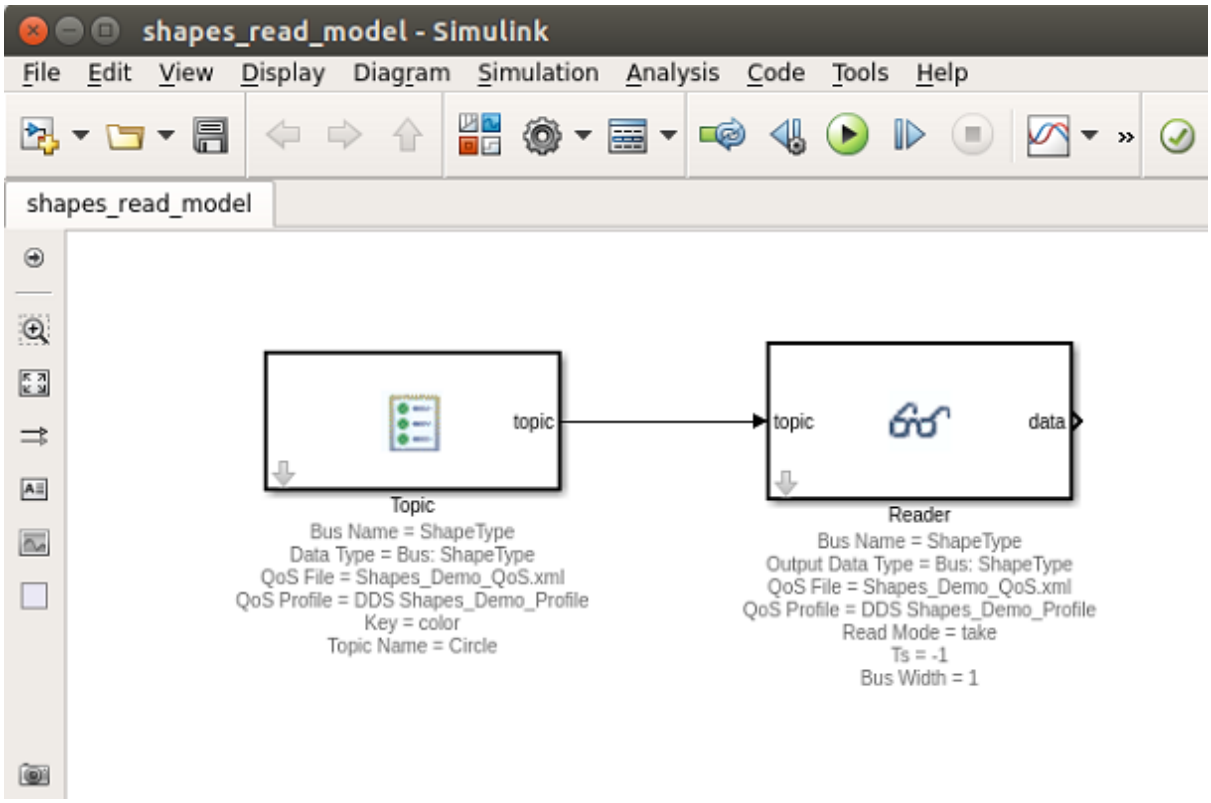
12.4.2.5 Set Reader Block Parameters

- Double click on the Reader block to edit the **Block Parameters**. Set the **Input Data Type** to the bus: ShapeType.
- Select the **QoS** tab.
- **Set the QoS file to** [Shapes_Demo_QoS.xml.]

  INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux/tools/matlab/
  examples/simulink/dds_reader_writer_model/Shapes_Demo_QoS.xml

12.4.2.6 Connect Topic and Reader

- Connect the Topic block **topic** output to the Reader block **topic** input.



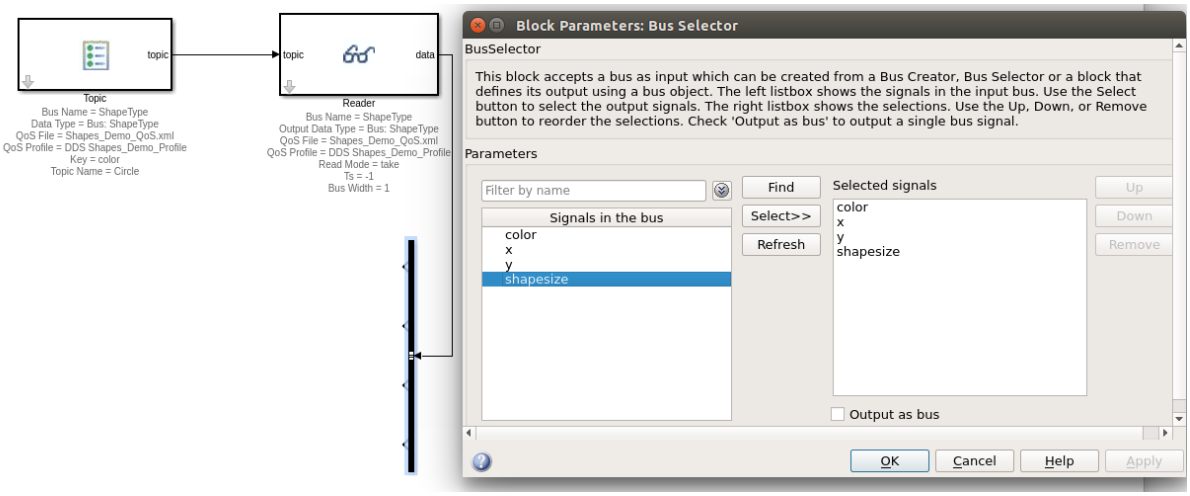
12.4.2.7 Add a Bus Selector to read and display sample data

To read and display sample data, we will add a **Simulink / Signal Routing / Bus Selector** block to our diagram.

12.4.2.8 Set Bus Selector Block Parameters

Specify the output signals we would like to display in our simulation. For this example, we will display all the ShapeType BUS signals in the running simulation.

- Connect the Reader **data** output to the **Bus Selector**.
- Double click on the **Bus Selector** block to edit the **Block Parameters**.
- Add all the signals in the bus to the **Selected signals**.



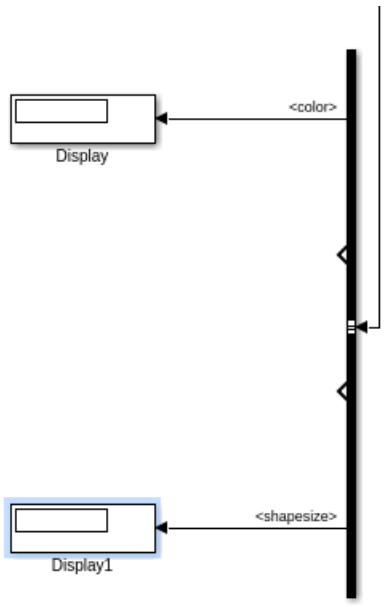
12.4.2.9 Add Bus Selector outputs

For demonstration purposes, we will output the bus signals using 2 Simulink **Display** blocks and an **XY Graph**.

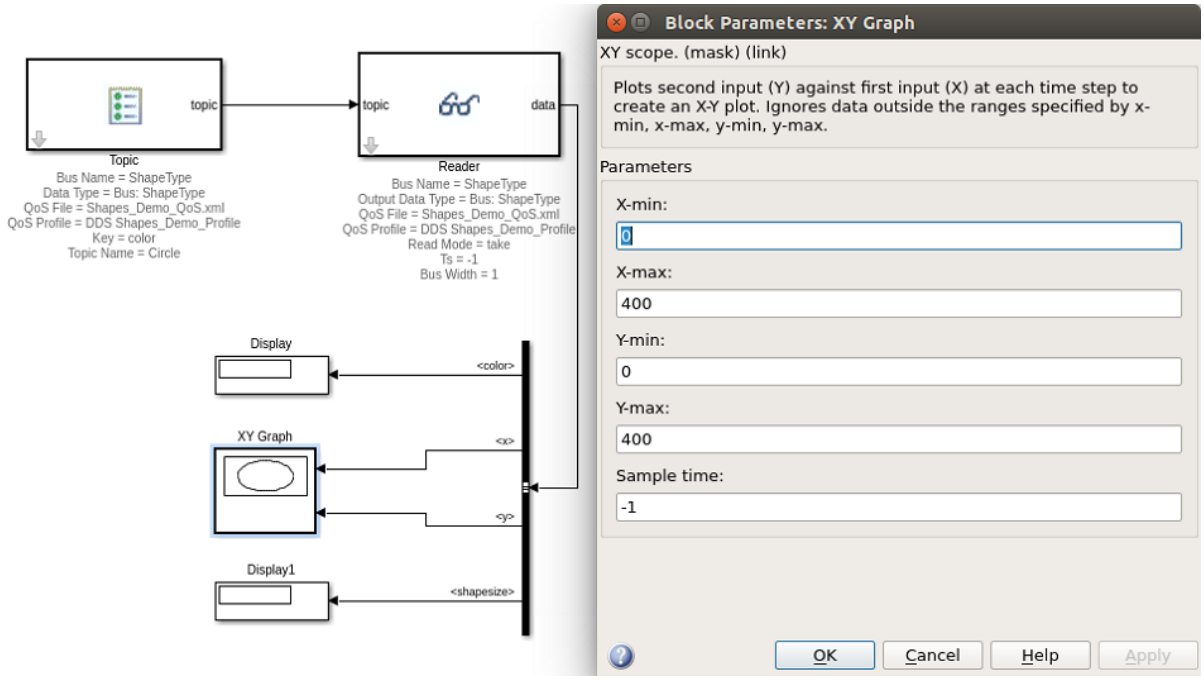
Note: To change the positioning of block ports, you can use the **Rotate & Flip** block menu item, accessible by right clicking on a block.

- Drag 2 **Simulink / Sinks / Display** blocks onto the diagram.
- **Connect the Display blocks to the Bus Selector output signals.**
  - Connect the Bus Selector color output signal to a Display block.
  - Connect the Bus Selector shapesize output signal to a Display block.

Note: Default Display block settings used.



- Drag **Simulink / Sinks / XY Graph** block onto diagram
- Connect the **BusSelector** x and y outputs to the **XY Graph** block.
- **Set the Block Parameters on the XY Graph:** X-min: 0 X-max: 400 Y-min: 0 Y-max: 400



Save your model!!! The model is now complete!

12.5 Running Simulations

We now have two Simulink models. We will run both models and see that data samples are being written by one model and read be the second model.

12.5.1 Setup Write Model

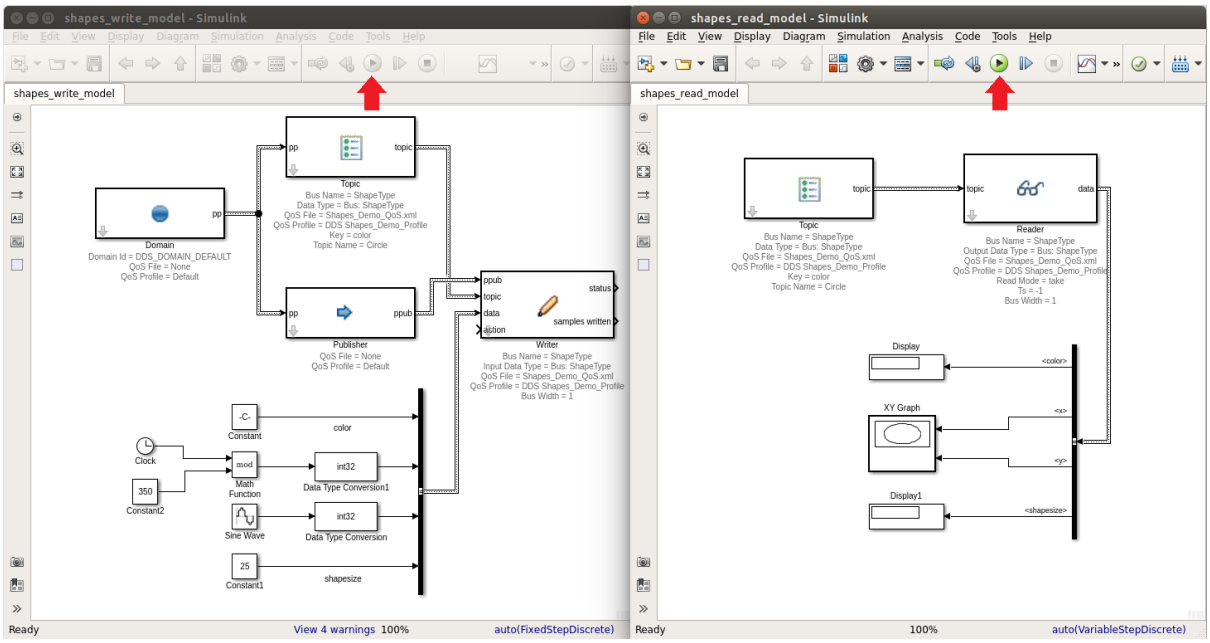
1. Open shapes_write_model.slx.
2. Select menu item **Simulation / Update Diagram** to diagnose any possible model problems.
3. Fix any issues.

12.5.2 Setup Read Model

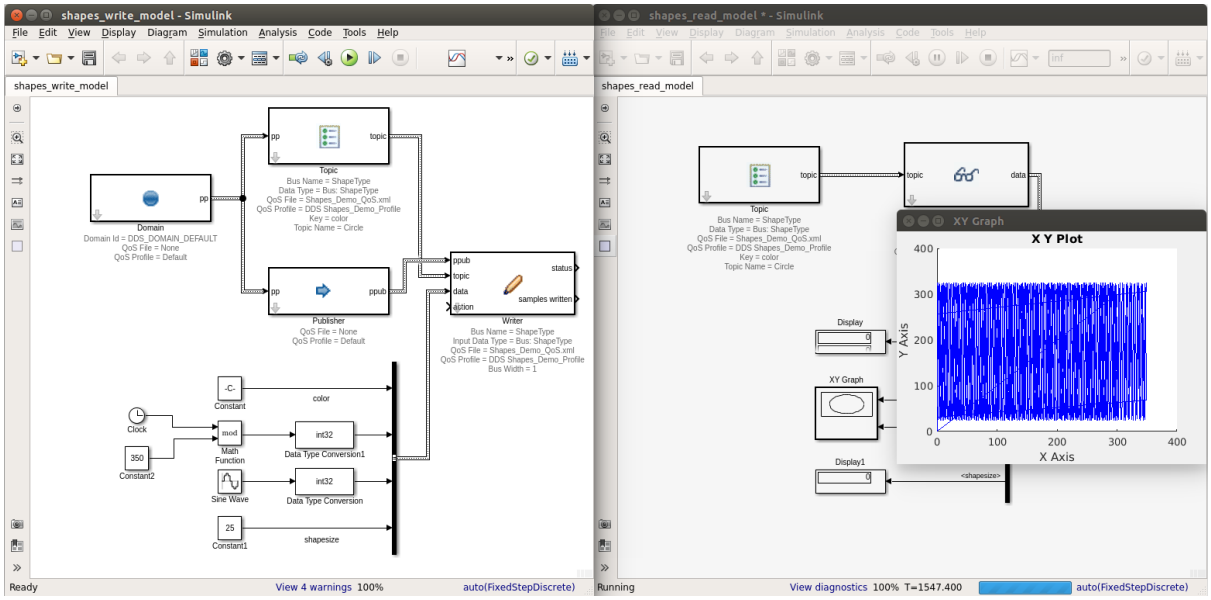
1. Open shapes_read_model.slx.
2. Select menu item **Simulation / Update Diagram** to diagnose any possible model problems.
3. Fix any issues.

12.5.3 Run Simulations

1. Position models side by side.
2. Start the read model simulation.
3. Start the write model simulation.
4. Expected: The write model will write samples, that are received by the read model and displayed in that model's **XY Graph** and **Display** blocks.
5. The write model will run to completion. The read model needs to be stopped manually.



Run Results


shapes_write_model - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
shapes_write_model
Domain
Topic
Publisher
Writer
Constant
Clock
Math Function
Constant2
Sine Wave
Data Type Conversion1
Data Type Conversion
Constant1
color
shapesize
Ready
View 4 warnings 100%
auto(FixedStepDiscrete)
shapes_read_model * - Simulink
shapes_read_model
XY Graph
X Y Plot
Y Axis
X Axis
Display
Display1
Running
View diagnostics 100% T=1547.400
auto(FixedStepDiscrete)

13

Generating C code with Simulink Coder

The Vortex DDS Blockset for Simulink supports Simulink Coder generation of C code, if you have a Simulink Coder license from MathWorks.

13.1 Prerequisites for C generation

In order to generate and compile C code containing DDS blocks, you must:

- Have Simulink Coder and MATLAB Coder installed and licensed from MathWorks.
- You have an appropriate C compile installed, as described by the MATLAB documentation.
- Vortex OpenSplice must be installed, and the appropriate `release.com` (Linux) or `release.bat` (Windows) script must have been executed in a command window.
- MATLAB must have been started from the same command window. You can check this by running the MATLAB command `getenv('OSPL_HOME')`. It should return a non-empty value.
- Your Simulink mode should execute correctly in simulation mode.

13.2 Preparing for C generation

Once your model has been validated via simulation mode, you are ready to generate and compile code. Because of an issue with the OpenSplice C99 language headers, you must manualy change the code generation options for your model. Follow these steps:

- From the model's menu, choose **Code > C/C++ Code > Code Generation Options**.
- Click on the **Code Generation** tab in the left-hand pane.
- In the **Build Configurations** drop-down, choose **Specify**.
- In the table that appears below this, edit the **Options** value in the **C Compiler** row to remove the text **$(ANSI_OPTS)**.
- Click **OK** or **Apply** to save your changes, then close the Code Generation Options dialog.

See the image, below, for an example of the code generation dialog.

13.3 Generating code

At least from the Vortex DDS Blockset point of view, you are ready to generate code. Follow these steps:

- From the model's menu, choose **Code > C/C++ Code > Build Model**.
- Simulink will get busy. You may see the following warnings in the Diagnostic View. These are OK, but are explained below.

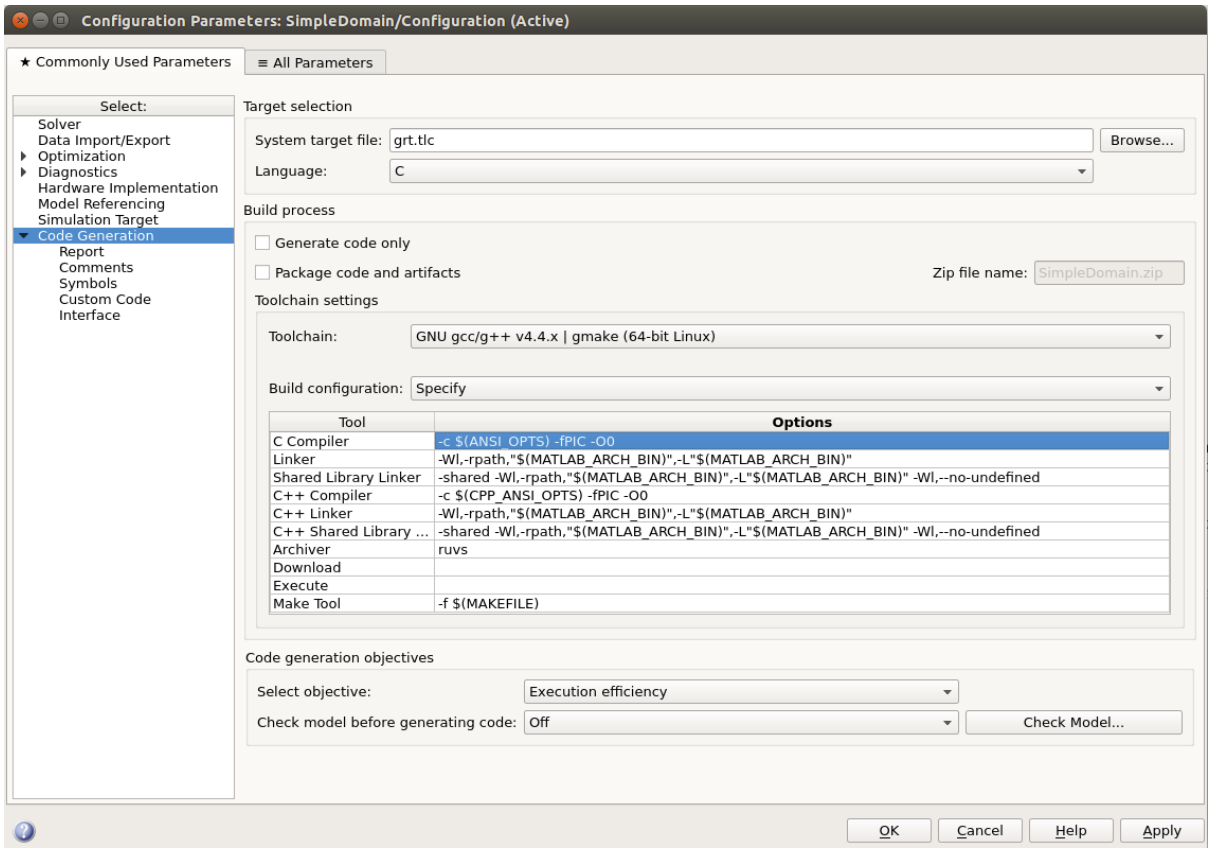


Figure 13.1: C/C++ Code Generation Options. Remove the text $(ANSI_OPTS) for C Compiler to avoid compile errors.

Domain Participant Warning

A warning may appear about the domain participant block:

```
Source 'SimpleDomain/Domain/Participant_Entity ' specifies that its sample time (-1)
is back-inherited. You should explicitly specify the sample time of sources. You can
disable this diagnostic by setting the 'Source block specifies -1 sample time'
diagnostic to 'none' in the Sample Time group on the Diagnostics pane of the
Configuration Parameters dialog box.

Component:Simulink | Category:Blockwarning
```

As the message states, this is because the block specifies a sample time of -1. The block only creates meaningful output on initialization (it connects to DDS), so any inherited sample time is sufficient. Specifying a sample time of -1 allows the block to be place into a function-call subblock.

Full header search warning

The following warning about reverting to full header searches may appear:

```
The following error occurred while attempting to run the preprocessor to find the
minimum needed set of include files:

While parsing the source file '<path-to>/source/debug_utils.c' the following error
occurred

<path-to>/source/debug_utils.c:14: cannot open source file "os_stdlib.h"
|  #include "os_stdlib.h"
|                       ^

Reverting to full header search.
```

This may occur as you are trying to package code from compilation on another platform. The referenced header file is part of the OpenSplice distrubution. When you compile on another platform, you will need to have that platform's OpenSplice distribution installed, and `release` variables set. The warning may be ignored.

Copy File information messages

If you are creating a source distrubution, you may see information messages such as the following:

```
cp: cannot stat '/libdcpsc99': No such file or directory
```

The build is attempting to copy OpenSplice shared libraries (which are refered to via environment variables). These should not be copied by the build. Instead, when you compile the source on a target platform, these libraries will be found in the local OpenSplice installation.

13.4 Cross-compilation of models

If you are using Simulink Coder to cross-compile a model using the VortexDDS Block Set to a target other than environment in which you are running MATLAB, then the cross compilation needs access to:

- the OpenSplice HDE environment on the host, in order to obtain include files as well as libraries and executables required to run the VortexDDS block set. This is specified by the *OSPL_HOME* environment variable.
- the OpenSplice RTS environment for the target environment, in order to obtain libraries necessary to link your executables. This is specified by the *LINK_OSPL_HOME* environment variable.

During a Simulink Coder build, the VortexDDS Block Set will print a warning if the LINK_OSPL_HOME environment variable is set, indicating that it is appropriate only for cross-compilation. If you are performing both cross-compilations and host compilations, you must take care to set LINK_OSPL_HOME appropriately. For host compilations, you should clear LINK_OSPL_HOME.

13.5 Running built models

When you run a compiled Simulink executable, you will need:

- An appropriate OpenSplice runtime installation on the machine executing the model
- The correct OpenSplice environment variables, which are set by the `release` script in the installation root directory.

14

Troubleshooting

When double clicking on a DDS block to view the block parameters, an error dialog is shown with the message: Error evaluating 'MaskDialog' callback of SubSystem block (mask).

Cause: The OSPL environment variables have not been setup correctly.

Solution: Open a command shell and run the script to setup OSPL environment variables.

Linux

- Open a Linux terminal.
- Navigate to directory containing release.com file.

  /INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.linux
- Run release.com. (Type in ". release.com" at command line.)

Windows

- Open a command prompt.
- Navigate to directory containing release.bat file.

  INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.8.x/HDE/x86_64.win64
- Run release.bat. (Type in "release.bat" at command line.)

15

Appendix A

Appendix A provides a description of the Simulink Bus to DDS Mapping implementation.

15.1 Simulink Bus to DDS Mapping

Simulink data is represented in buses whose types are not compatable with DDS types. Therfore sending Simulink data to DDS requires a conversion from Simulink types to DDS types. Conversly sending DDS data to Simulink requires a conversion from DDS types to Simulink types. This document describes the mapping between these types, the type descriptors generated for Topic registration and any annotations needed to describe keys, namespaces, ... for two different workflows.

15.1.1 Workflow 1: Using idlpp to Generate Simulink Bus and Type Descriptor

DDS Topic Types are described in IDL. To be compatible with DDS, these types must be represented in Sumlink as Bus Types. On writes, these Simulink Bus types are converted to DDS types and on read DDS types are converted to Simulink Bus types.

IDL can be defined in an IDL file and the corresponding Simulink Bus Types can be created by `Vortex.idlImportSl` at the MATLAB command line. This will create the necessary Simulink Bus Types and Elements in a Simulink dictionary. It also creates the XML type descriptor for the Simulink Bus types that is used to create the Topic. The type descriptor is a Simulink variable of the form <bus-name>TypeDescriptor. The type descriptor is necessary for the block set to work correctly, but you will not directly address this when creating Simulink blocks.

DDS IDL to Simulink Bus Mapping

The table below describes the generated Simulink artifacts when `Vortex.idlImportSl` is invoked from MATLAB.

DDS IDL	Simulink Type	Annotation	Comments
struct B	Simulink.Bus B		Creates Simulink Bus
enum E	Simulink Enum E		Creates Simulink Enum type
module A		@Scope(A)	Added to each enum or struct contained in the module.
#pragma keylist		@Key	Every topic bus has @Key in its description.
boolean	boolean		
char	int8		
octet	uint8		
short	int16		
unsigned short	uint16		
long	int32		
unsigned long	uint32		
long long	double		No int64 in Simulink. Copied into the memory allocated for a double.
unsigned long long	double		No int64 in Simulink. Copied into the memory allocated for a double.
float	single		
double	double		
string	int8	@String	Default dimension is 256
string<N>	int8 Dimension N	@BString	
T field[N]	Mapped type for T. Dimension N		Multidimensional arrays are supported.
sequence<T,N>	Bus: seqN_T. Dimension: N		E.g. sequence<long,3> becomes “Bus: seq3_int32”
sequence<T>	Bus: seq_T Dimension: 16		E.g. sequence<long> becomes “Bus: seq_int32” with default dimension of 16
typedef			expanded in place
Unsupported DDS data types			
wchar	*not supported*		
wstring	*not supported*		
any	*not supported*		
long double	*not supported*		
union	*not supported*		
inheritance	*not supported*		

15.1.2 Workflow 2: Manually Modeling DDS data in the Simulink Bus Editor

DDS IDL is not necessary to interact with DDS applicatoins. You can also model the Simulink buses directly. In this case, the block set will infer the DDS data types from the Simulink types.

Defining Simulink busses without first defining the IDL is not recommended; it has the following limitations:

- fewer IDL concepts are supported. In particular, sequences are unsupported.
- it will be difficult for your Simulink application to interact with applications written in other languages, as those languages will required IDL to define the topic data.

IDL-less mapping of Simulink bus IDL concepts

The table below describes how a Simulink bus that was not created from an IDL file is mapped to IDL concepts.

Simulink Type	IDL equivalent	Description
Simulink.Bus B	struct B	Defines DDS topic type B
bus annotation: @Scope(A::B)	module	Creates DDS namespace for struct
bus annotation: @Key(f1,f2)	#pragma keylist	Defines topic key field(s)
boolean	boolean	IDL array if Dimensions > 1
int8	char	IDL array if Dimensions > 1
uint8	octet	IDL array if Dimensions > 1
int16	short	IDL array if Dimensions > 1
uint16	unsigned short	IDL array if Dimensions > 1
int32	long	IDL array if Dimensions > 1
uint32	unsigned long	IDL array if Dimensions > 1
single	float	IDL array if Dimensions > 1
double	double	IDL array if Dimensions > 1
int8, annotated @String uint8, annotated @String	string	max length of read strings is determined by field dimension
int8, annotated @BString uint8, annotated @BString Dimension N	string<N>	
Simulink enumeration, E	enum E	IDL array if Dimensions > 1

16

Contacts & Notices

16.1 Contacts

ADLINK Technology Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA
Tel: +1 781 569 5819

ADLINK Technology Limited
The Edge
5th Avenue
Team Valley
Gateshead
NE11 0XA
UK
Tel: +44 (0)191 497 9900

ADLINK Technology SARL
28 rue Jean Rostand
91400 Orsay
France
Tel: +33 (1) 69 015354

Web: http://ist.adlinktech.com/

Contact: http://ist.adlinktech.com

E-mail: ist_info@adlinktech.com

LinkedIn: https://www.linkedin.com/company/79111/

Twitter: https://twitter.com/ADLINKTech_usa

Facebook: https://www.facebook.com/ADLINKTECH

16.2 Notices

Copyright © 2019 ADLINK Technology Limited. All rights reserved.

This document may be reproduced in whole but not in part. The information contained in this document is subject to change without notice and is made available in good faith without liability on the part of ADLINK Technology Limited. All trademarks acknowledged.