ADLINK

VORTEX

DDS MATLAB Guide

Release 6.x

Contents

1

Introduction

The DDS MATLAB Integration provides users with DDS MATLAB classes to model DDS communication using MATLAB and pure DDS applications.

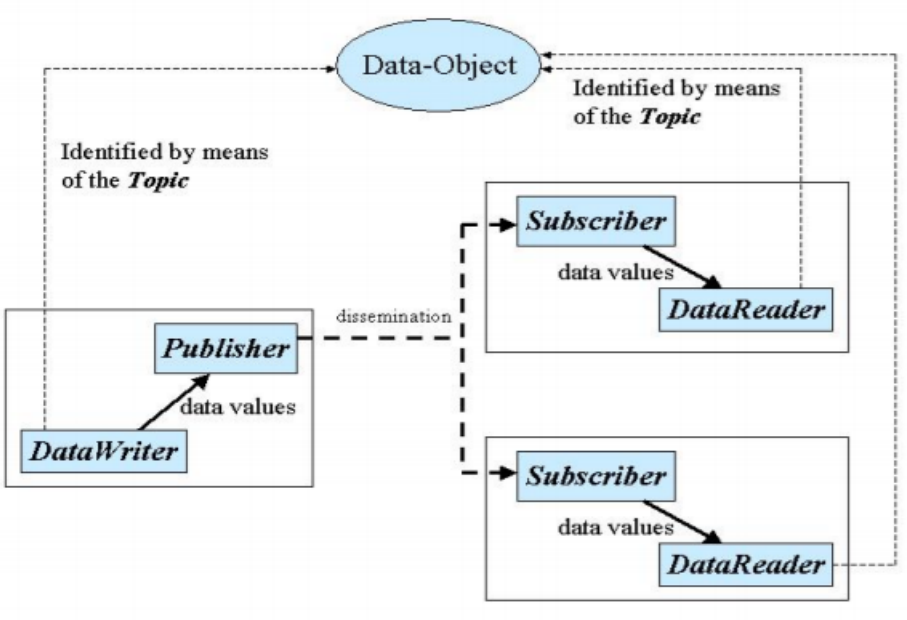
Please refer to the DDS and MATLAB documentation for detailed information.

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 MATLAB

What is MATLAB?

"The Language of Technical Computing

Millions of engineers and scientists worldwide use MATLAB® to analyze and design the systems and products transforming our world. MATLAB is in automobile active safety systems, interplanetary spacecraft, health monitoring devices, smart power grids, and LTE cellular networks. It is used for machine learning, signal processing, image processing, computer vision, communications, computational finance, control design, robotics, and much more.

Math. Graphics. Programming.

The MATLAB platform is optimized for solving engineering and scientific problems. The matrix-based MATLAB language is the world's most natural way to express computational mathematics. Built-in graphics make it easy to visualize and gain insights from data. A vast library of prebuilt toolboxes lets you get started right away with algorithms essential to your domain. The desktop environment invites experimentation, exploration, and discovery. These MATLAB tools and capabilities are all rigorously tested and designed to work together.

Scale. Integrate. Deploy.

MATLAB helps you take your ideas beyond the desktop. You can run your analyses on larger data sets and scale up to clusters and clouds. MATLAB code can be integrated with other languages, enabling you to deploy algorithms and applications within web, enterprise, and production systems."

https://www.mathworks.com/products/matlab.html

2

Installation

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

2.1 System Requirements

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

2.2 OpenSplice (OSPL) and DDS MATLAB Installation

Steps:

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

   /INSTALLDIR/Vortex_v2/license
3. DDS MATLAB files are contained in a tools/matlab folder

   Example: */INSTALLDIR/Vortex_v2/Device/VortexOpenSplice/6.8.1/HDE/x86_64.linux/tools/matlab*

2.3 MATLAB and DDS 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/Vortex_v2/Device/VortexOpenSplice/6.8.1/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/Vortex_v2/Device/VortexOpenSplice/6.8.1/HDE/x86_64.win64
   - Run release.bat. (Type in "release.bat" at command line.)

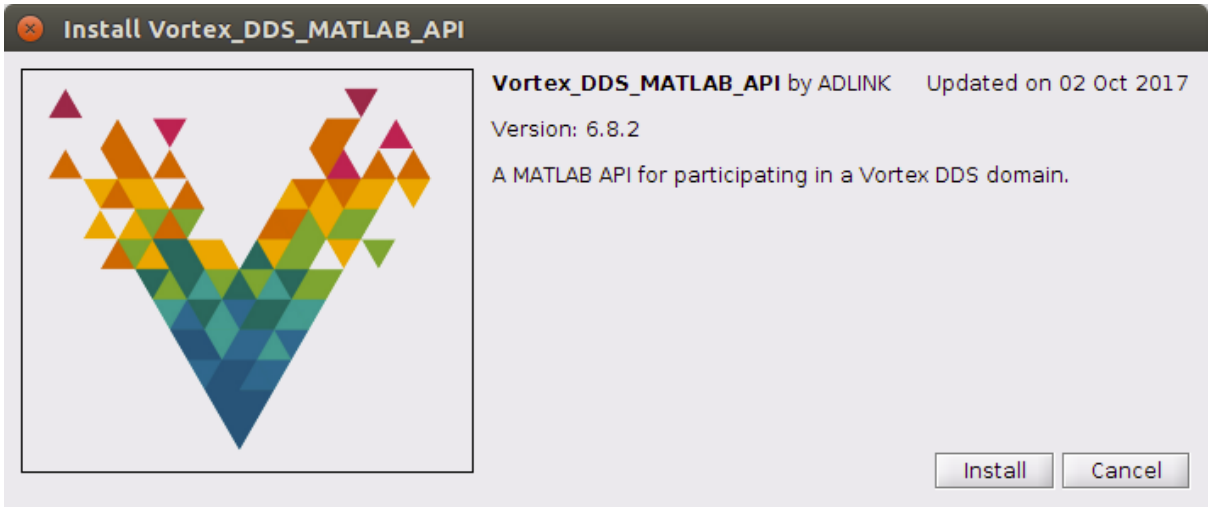
2. Start MATLAB using the **SAME** command shell used in Step 1.

   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 MATLAB classes.

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

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

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



2.4 Examples

Example models have been provided in the examples folder.

Example: */INSTALLDIR/Vortex_v2/Device/VortexOpenSplice/6.8.1/HDE/x86_64.linux/tools/matlab/examples/matlab*

3

MATLAB API for Vortex DDS Classes

The DDS MATLAB Integration provides a class library with custom classes to read and write data with DDS.

The MATLAB DDS Classes are included in a **Vortex** package.

3.1 API Usage patterns

The typical usage pattern for the MATLAB API for Vortex DDS is the following:

- model your DDS topics using IDL
- using `idlpp -l matlab` to compile your IDL into MATLAB topic classes. See *MATLAB Generation from IDL*.
- start writting your MATLAB program using the MATLAB API for Vortex DDS.

The core classes you must use are `Vortex.Topic` and either `Vortex.Reader` or `Vortex.Writer`. Other classes may be required, especially if you need to adjust the Quality of Service (QoS) defaults. For details on setting QoS values with the API, see *QoS Provider*. The following list shows the sequence in which you would use the Vortex classes:

- If you require participant-level non-default QoS settings, create a `Vortex.Participant` instance. Pass the participant to subsequently created Vortex entities.
- Create one or more `Vortex.Topic` instances for the IDL topics your program will read or write.
- If you require publisher or subscriber level non-default QoS settings, create `Vortex.Publisher` and/or `Vortex.Subscriber` instances. Pass these to any created reader or writers. (The most common reason for changing publisher/subscriber QoS is to define non-default partitions.)
- Create `Vortex.Reader` and/or `Vortex.Writer` classes from the `Vortex.Topic` instances that you created.
- If you required data filtering, create `Vortex.Query` objects.
- Create the core of program, creating instances of your topic classes and writing them; or, reading data and processing it.

3.2 Vortex.Topic

The MATLAB Topic class 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 MATLAB class must be defined in the MATLAB workspace. This is either a class created by `idlpp` (see *MATLAB Generation from IDL*) or a manually created MATLAB class (see *MATLAB without IDL*). It is recommend that you create DDS Topic type definitions via IDL and *idlpp*.

API Examples

Create a Vortex DDS domain topic. Returns a topic instance, or throws a `Vortex.DDSException` if the topic cannot be created.

Create a topic named 'Circle' based on the DDS Topic type `ShapeType` with default participant and QoS:

```
topic = Vortex.Topic('Circle', ?ShapeType);
```

Note: In MATLAB, references to classes such as ShapeType are created by prefixing them with a question mark (?). If the class is in a MATLAB package, then the fully qualified name must be used. For example: `?ShapesDemo.Topics.ShapeType`.

Create the 'Circle' topic with an explicitly created participant:

```
% dp: a Vortex.DomainParticipant instance
topic = Vortex.Topic(dp, 'Circle', ?ShapeType);
```

Create the 'Circle' topic with default participant and QoS profile:

```
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
topic = Vortex.Topic('Circle', ?ShapeType, qosFileURI, qosProfile);
```

Create the 'Circle' topic with explicit participant and QoS profile:

```
% dp: a Vortex.DomainParticipant instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
topic = Vortex.Topic(dp, 'Circle', ?ShapeType, qosFileURI, qosProfile);
```

3.3 Vortex.DomainParticipant

The `Vortex.DomainParticipant` class 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."

Use of the `Vortex.DomainParticipant` class is optional. The API provides a 'default participant', which is used if no explicit domain participant is provided. The default participant is created on first usage, and is disposed when MATLAB exits. Reasons for using an explicitly created domain participant are:

- to provide non-default QoS settings.
- to control the timing of participant creation and destruction.

API Examples

Create a Vortex DDS domain participant. Returns participant or throws a `Vortex.DDSException` if the participant cannot be created.

Create a domain participant in the default DDS domain (the one specified by the OSLP_URI environment variable):

```
dp = Vortex.DomainParticipant();
```

Create a domain participant on domain, specifying a domain id:

```
% domainId: an integer value
dp = Vortex.DomainParticipant(domainId);
```

Note: The underlying DCPS C99 API used by *Vortex.DomainParticipant* does not currently support this operation, and will result in a `Vortex.DDSException` being raised.

Create a participant on default domain with QoS profile:

```
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
dp = Vortex.DomainParticipant(qosFileURI, qosProfile);
```

Create a participant on domain with QoS profile:

```
% domainId: an integer value
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
dp = Vortex.DomainParticipant(domainId, qosFileURI, qosProfile);
```

3.4 Vortex.Publisher

The MATLAB `Vortex.Publisher` class represents a DDS publisher entity.

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

Use of the `Vortex.Publisher` class is optional. In it's place, you can use a `Vortex.DomainParticipant` instance, or default to the *default domain participant*. Reasons for explicitly creating a `Vortex.Publisher` instance are:

- to specify non-default QoS settings, including specifying the DDS *partition* upon which samples are written.
- to control the timing of publisher creation and deletion.

API Examples

Create a DDS Publisher entity. Returns publisher or throws a `Vortex.DDSException` if the publisher cannot be created.

Create a default publisher with default participant:

```
pub = Vortex.Publisher();
```

Create a publisher with an explicit participant:

```
% dp: a Vortex.DomainParticipant instance
pub = Vortex.Publisher(dp);
```

Create default publisher with default participant and QoS profile:

```
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
pub = Vortex.Publisher(qosFileURI, qosProfile);
```

Create a publisher with participant and QoS profile:

```
% dp: a Vortex.DomainParticipant instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
pub = Vortex.Publisher(dp, qosFileURI, qosProfile);
```

3.5 Vortex.Writer

The MATLAB `Vortex.Writer` class 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."

A `Vortex.Writer` class is required in order to write data to a DDS domain. It may be explicitly attached to a DDS publisher or a DDS domain participant; or, it is implicitly attached to the *default domain participant*.

A `Vortex.Writer` class instance references an existing `Vortex.Topic` instance.

API Examples

Create a Vortex DDS domain writer. Returns writer or throws a `Vortex.DDSException` if the writer cannot be created.

Create a writer for a topic, in the default domain participant and default QoS settings:

```
% topic: a Vortex.Topic instance
writer = Vortex.Writer(topic);
```

Create a writer within an explicitly specified publisher or domain participant:

```
% pubOrDp: a Vortex.Publisher or Vortex.DomainParticipant instance
% topic: a Vortex.Topic instance
writer = Vortex.Writer(pubOrDp, topic);
```

Create writer for a topic with explicit QoS profile:

```
% topic: a Vortex.Topic instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
writer = Vortex.Writer(topic, qosFileURI, qosProfile);
```

Create a writer with publisher or participant, topic and QoS profile:

```
% pubOrDp: a Vortex.Publisher or Vortex.DomainParticipant instance
% topic: a Vortex.Topic instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
writer = Vortex.Writer(pubOrDp, topic, qosFileURI, qosProfile);
```

Write a ShapeType topic class instance to a writer:

```
% writer: a Vortex.Writer instance
% ShapeType: a 'topic class' created manually or via IDLPP
data = ShapeType(); % create an object instance
% set data values...
data.color = 'RED';
% ... set other values ...

ddsStatus = writer.write(data);
```

Note: the returned status value is 0 for success, and negative for failure. Use the `Vortex.DDSException` class to decode an failure status.

Dispose a DDS topic instance:

```
% writer: a Vortex.Writer instance
% ShapeType: a 'topic class' created manually or via IDLPP
data = ShapeType(); % create an object instance
% set data key values...
data.color = 'RED';

ddsStatus = writer.dispose(data);
```

Note: the returned status value is 0 for success, and negative for failure. Use the `Vortex.DDSException` class to decode an failure status.

Unregister a DDS topic instance:

```
% writer: a Vortex.Writer instance
% ShapeType: a 'topic class' created manually or via IDLPP
data = ShapeType(); % create an object instance
% set data key values...
data.color = 'RED';

ddsStatus = writer.unregister(data);
```

Note: the returned status value is 0 for success, and negative for failure. Use the `Vortex.DDSException` class to decode an failure status.

3.6 Vortex.Subscriber

The MATLAB `Vortex.Subscriber` class 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."

Use of the `Vortex.Subscriber` class is optional. In it's place, you can use a `Vortex.DomainParticipant` instance, or default to the *default domain participant*. Reasons for explicitly creating a `Vortex.Subscriber` instance are:

- to specify non-default QoS settings, including specifying the DDS *partition* upon which samples are written.
- to control the timing of subscriber creation and deletion.

API Examples

Create a Vortex DDS domain subscriber. Returns subscriber or throw a `Vortex.DDSException` if the subscriber cannot be created.

Create a subscriber within the default domain participant:

```
sub = Vortex.Subscriber();
```

Create a subscriber within an explicit participant:

```
% dp: a Vortex.DomainParticipant instance
sub = Vortex.Subscriber(dp);
```

Create subscriber within the default domain participant and with a QoS profile:

```
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
sub = Vortex.Subscriber(qosFileURI, qosProfile);
```

Create a subscriber with participant and QoS profile:

```
% dp: a Vortex.DomainParticipant instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
sub = Vortex.Subscriber(dp, qosFileURI, qosProfile);
```

3.7 Vortex.Reader

The MATLAB `Vortex.Reader` class represents a DDS data reader entity.

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

A `Vortex.Reader` class is required in order to write data to a DDS domain. It may be explicitly attached to a DDS subscriber or a DDS domain participant; or, it is implicitly attached to the *default domain participant*.

A `Vortex.Reader` class instance references an existing `Vortex.Topic` instance.

API Examples

Create a Vortex DDS domain reader. Returns reader or throw a `Vortex.DDSException` instance if the reader cannot be created.

Create a reader for a topic within the default domain participant, and with default QoS:

```
% topic: a Vortex.Topic instance
reader = Vortex.Reader(topic);
```

Create a reader for a topic within a subscriber or participant, and with default QoS:

```
% subOrDp: a Vortex.Subscriber or Vortex.DomainParticipant instance
% topic: a Vortex.Topic instance
reader = Vortex.Reader(subOrDp, topic);
```

Create reader for a topic within the default domain participant and with a QoS profile:

```
% topic: a Vortex.Topic instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
reader = Vortex.Reader(topic, qosFileURI, qosProfile);
```

Create a reader for a topic within a subscriber or participant, with with a QoS profile:

```
% subOrDp: a Vortex.Subscriber or Vortex.DomainParticipant instance
% topic: a Vortex.Topic instance
% qosFileURI: a char array representing a file: URI
% qosProfile: a char array containing the name of a profile defined in the QoS File
reader = Vortex.Reader(subOrDp, topic, qosFileURI, qosProfile);
```

Take data from a data reader:

```
% reader: a Vortex.Reader
[data, dataState] = reader.take;
% data: an array of topic class instances (e.g. ShapeType); possibly empty
% dataState: an struct array; each entry describes the
%    state of the corresponding data entry
```

Read data from a data reader:

```
% reader: a Vortex.Reader
[data, dataState] = reader.read;
% data: an array of topic class instances (e.g. ShapeType); possibly empty
% dataState: an struct array; each entry describes the
%    state of the corresponding data entry
```

Specify a wait timeout, in seconds, before read or take will return without receiving data:

```
% reader: a Vortex.Reader
reader.waitsetTimeout(2.0);
```

3.8 Vortex.Query

The MATLAB `Vortex.Query` class represents a DDS query entity.

A query is a data reader, restricted to accessing data that matches specific status conditions and/or a filter expression.

A `Vortex.Query` class instance references an existing `Vortex.Reader` instance.

API Examples

Create a `Vortex.Query` instance or throw a `Vortex.DDSException` if an error occurs.

Create a query based on a state mask only:

```
% reader: a Vortex.Reader
% only receive samples that:
%   * have not been read by this application
%   * AND for instances that previously seen by this application
%   * AND for which there is a live writer
mask = Vortex.DataState.withNew().withNotRead().withAlive();
query = Vortex.Query(reader, mask);
```

Create a query based on a state mask and a filter expression:

```
% reader: a Vortex.Reader
mask = Vortex.DataState.withAnyState();
filter = 'color = %0 and x > %1';
% filter for 'RED' shapes with x > 10...
query = Vortex.Query(reader, mask, filter, {'RED', 10'});
```

Take data from a query:

```
% query: a Vortex.Query
[data, dataState] = query.take;
% data: an array of topic class instances (e.g. ShapeType); possibly empty
% dataState: an struct array; each entry describes the
%    state of the corresponding data entry
```

Read data from a query:

```
% query: a Vortex.Query
[data, dataState] = query.read;
% data: an array of topic class instances (e.g. ShapeType); possibly empty
% dataState: an struct array; each entry describes the
%    state of the corresponding data entry
```

Specify a wait timeout, in seconds, before read or take will return without receiving data:

```
% query: a Vortex.Query

% specify the waitset timeout on the reader
query.waitsetTimeout(2.0);

% now, read or take 'query'
```

3.9 Vortex.DDSException

The `Vortex.DDSException` class is thrown in the case of a DDS error arising. The class can also be used to decode an error status code returned by methods such as `Vortex.Writer.write`.

API Examples

Catch a DDS error while creating a DDS entity:

```
% dp: a Vortex.DomainParticipant
try
  topic = Vortex.topic('Circle', ?SomeAlternateDef.ShapeType);
catch ex
  switch ex.identifier
    case 'Vortex:DDSError'
      % it's a Vortex Error
      fprintf(['DDS reports error:\n' ...
         '  %s\n' ...
         '    DDS ret code: %s (%d)\n'], ...
         ex.message, char(ex.dds_ret_code), ex.dds_ret_code);
    otherwise
      rethrow ex;
  end
end
```

Decode a dds status code returned by `Vortex.Writer.write`:

```
% ddsstatus: a Vortex.Writer.write return value
ex = Vortex.DDSException('', ddsstatus);
switch ex.dds_ret_code
  case Vortex.DDSReturnCode.DDS_RETCODE_OK
```

```
    % ...
  case Vortex.DDSReturnCode.DDS_BAD_PARAMETER
    % ...
  case Vortex.DDSReturnCode.DDS_RETCODE_INCONSISTENT_POLICY
    % ...
end
```

4

MATLAB Generation from IDL

The DDS MATLAB Integration supports generation of MATLAB classes from IDL. This chapter describes the details of the IDL-MATLAB binding.

4.1 Running IDLPP

Compiling IDL into MATLAB code is done using the `-l matlab` switch on idlpp:

```
idlpp -l matlab idl-file-to-compile.idl
```

Generated Artifacts

The following table defines the MATLAB artifacts generated from IDL concepts:

IDL Concept	MATLAB Concept	Comment
module	package	a MATLAB package is a folder starting with '+'.
enum	class	a MATLAB .m file.
enum value	enum value	
struct	class	a MATLAB .m file.
field	class property	
typedef		IDL typedef's are inlined.
union	Unsupported	
inheritance	Unsupported	

Datatype mappings

The following table shows the MATLAB equivalents to IDL primitive types:

IDL Type	MATLAB Type
boolean	logical
char	int8
octet	uint8
short	int16
ushort	uint16
long	int32
ulong	uint32
long long	int64
ulong long	uint64
float	single
double	double
string	char
wchar	Unsupported
wstring	Unsupported
any	Unsupported
long double	Unsupported

Implementing Arrays and Sequences in MATLAB

Both IDL arrays and IDL sequences are mapped to MATLAB arrays. MATLAB supports both native array types, which must have homogenenous contents and *cell arrays*, which may have heterogenous content. In general, IDLPP prefers native arrays, as they support more straight forward type checking. However, some situations require cell arrays. The following table summarizes the cases where IDLPP will generate cell arrays:

Datatype	Sample Syntax	Reason for using cell array
sequence of sequence	sequence<sequence<T>> f;	Nested sequences need not have a homogeneous length
array of sequence	sequence<T> f[N];	Sequences lengths need not be homogeneous
sequence of array	sequence<A> f;	A multi-dim array makes adding elements too difficult
sequence of string	sequence<string> f;	Nested strings need not have a homogeneous length
string array	string f[N];	Nested strings need not have a homogeneous length

4.2 Limitations of MATLAB Support

The IDL-to-MATLAB binding has the following limitations:

- IDL unions are not supported
- the following IDL data types are not supported: wchar, wstring, any and long double
- arrays of sequences of structures are not supported

5

MATLAB without IDL

It is possible, but not recommended, to directly create MATLAB classes that represent DDS topics. This approach allows you to quickly start using Vortex DDS, but it has a number of disadvantages:

- without IDL, you cannot reliably define the topic in other language bindings.
- not all IDL types are available.

5.1 Creating a MATLAB Topic class

To create a Topic Class without IDL, create a MATLAB class that inherits from Vortex.AbstractType. The class will be interpreted as an IDL struct. Class properties will be interpreted as IDL struct fields. Finally, the class must define a static method *getKey* which returns a comma separated list of key fields for the topic.

The following shows a simple MATLAB class, ShapeType:

```
classdef ShapeType < Vortex.AbstractType
    properties
        color     char  % IDL: string color;
        x         int32 % IDL: long x;
        y         int32 % IDL: long y;
        shapesize int32 % IDL: long shapesize;
    end

    methods (Static)
        function keys = getKey
            keys = 'color';
        end
    end
end
```

The topic class defines four fields, and identifies the `color` field as the topic key.

You would use the topic class in the same way as one generated by IDLPP. The following example shows the creation of a DDS topic object from a topic class:

```
% define a Vortex DDS topic called 'Circle'
circleTopic = Vortex.Topic('Circle', ?ShapeType);
```

With the *circleTopic* variable, you can then create appropriate Vortex DDS readers and writers.

5.2 Mapping of MATLAB types to IDL types

When using an IDL-less Topic class, the Vortex DDS API for MATLAB makes maps property types to IDL types as follows:

MATLAB Type	IDL Type
logical	boolean
int8	char
uint8	octet
int16	short
uint16	unsigned short
int32	long
uint32	unsigned long
int64	long long
uint64	unsigned long long
single	float
double	double
char	string
class	equivalent IDL struct
enum class	equivalent IDL enum
not type	double

5.3 Creating arrays

When defining a topic class, you can make a field map to an IDL array by initializing it to an array of the appropriate dimensions. The MATLAB API for Vortex DDS recognizes arrays of most types as identifying IDL arrays. The one exception is that arrays of MATLAB `char` are still interpreted as an IDL string.

The following example shows an topic class that defines arrays:

```
classdef ArrayTopic < Vortex.AbstractType
    properties
        x = zeros([1 4])        %IDL: double x[4];
        y int32 = zeros([3 4])  %IDL: long y[3][4];
    end
    methods (Static)
        function keys = getKey
            keys = ''; %No Key
        end
    end
end
```

5.4 Unsupported Types

When creating topics with out IDL, you must accept the following restrictions:

- IDL sequences cannot be defined.
- arrays of IDL strings cannot be defined.
- bounded IDL strings cannot be defined.

6

QoS Provider

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

6.1 QoS Provider File

Quality of Service for DDS entities is set using XML files based on the XML schema file 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.

6.2 QoS Profile

A QoS profile consists of a name and optionally a base_name attribute. The base_name attribute allows a QoS or a profile to inherit values from another QoS or profile in the same file. 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:sc
    <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>
```

6.3 Setting QoS Profile in MATLAB

To set the QoS profile for a DDS entity in MATLAB the user must specify the File URI and the QoS profile name. If the file is not specified, the DDS entity will be created with the default QoS values.

```
% QOS File
PINGER_QOS_FILE = '/home/dds/matlab_examples/pingpong/DDS_PingerVolatileQoS.xml';
PINGER_QOS_PROFILE = 'DDS VolatileQosProfile';

% create the pinger participant on default domain with specified qos profile
dp = Vortex.DomainParticipant(['file://', PINGER_QOS_FILE], PINGER_QOS_PROFILE);
```

The file for the above would minimally contain the following <domainparticipant_qos> tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns="http://www.omg.org/dds/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sc
    <qos_profile name="DDS VolatileQosProfile">
        <domainparticipant_qos>
            <user_data>
                <value></value>
            </user_data>
            <entity_factory>
                <autoenable_created_entities>true</autoenable_created_entities>
            </entity_factory>
         </domainparticipant_qos>
    </qos_profile>
</dds>
```

7

Ping Pong Example

A simple ping pong example is provided to demonstrate the basic capabilities of the MATLAB DDS integration.

The ping pong example can be found in the following directory:

OSPL_HOME/tools/matlab/examples/matlab/pingpong

The ping pong example creates two participants.

1. A pinger that writes to the PING partition and reads from the PONG partition.
2. A ponger that writes to the PONG partition and reads from the PING partition.

A matlab script is provided that writes and reads sample data in the pinger and ponger participants.

7.1 Example Files

Files with the .m extension are MATLAB script files.

An explanation of what each example file does is provided below.

pingpong.idl

- Defines the PingPongType in idl
- Used to generate the MATLAB file PingPongType.m via:

idlpp -l matlab pingpong.idl

PingPongType.m

- Defines a PingPongType; generated from idlpp
- The PingPongType represents a DDS topic type.
- PingPongType specifies two properties: id, count.

DDS_PingerVolatileQoS.xml

- XML file that specifies the DDS QoS (quality of service) settings for pinger.

DDS_PongerVolatileQoS.xml

- XML file that specifies the DDS QoS (quality of service) settings for ponger.

setup_pinger.m

- Creates the pinger participant on the default DDS domain, with specified QoS profile.
- Creates the topic PingPongType.
- Creates the publisher using the domain participant on the PING partition specified in the specified QoS profile.
- Creates the writer using the publisher and the specified QoS profile.
- Creates the subscriber using the domain participant on the PONG partition specified in the QoS profile.

- Creates the reader using the subscriber and the specified QoS profile.

setup_ponger.m

- Creates the ponger participant on default domain with specified QoS profile.
- Creates the topic PingPongType.
- Creates the publisher using the domain participant on the PONG partition specified in the QoS profile.
- Creates the writer using the publisher and the specified QoS profile.
- Creates the subscriber using the domain participant on the PING partition specified in the QoS profile.
- Creates the reader using the subscriber and the specified QoS profile.

pinger_ponger.m

- MATLAB script that writes and reads sample data in the pinger and ponger participants.
- This script calls the setup_pinger.m and setup_ponger.m scripts.
- This is the main script to run the ping pong example.
- This script is run from the MATLAB Command Window.

7.2 Steps to run example

1. **In the MATLAB command window, run pinger_ponger.m.**
    - Type “pinger_ponger”.
    - Hit enter.

8

Contacts & Notices

8.1 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

8.2 Notices