VORTEX

Python DCPS API Guide

Release 6.x

CONTENTS

CHAPTER

ONE

PREFACE

1.1 About the Python DCPS API Guide

The Python DCPS API Guide is a starting point for anyone using, developing or running Python applications with Vortex OpenSplice.

This guide contains:

- Python DCSP API installation instructions
- location of Python DCPS API dds module documentation
- overview of general DDS concepts and Python API for Vortex OpenSplice
- a listing of examples, and how to run them
- detailed information on how to specify DDS entity Quality of Service (QoS)
- how to discover and register DDS topics

This reference guide is based on the OMG's Data Distribution Service Specification and Python Language Mapping Specification.

Please note that this guide is not intended to provide a detailed explanation of the aforementioned OMG specifications or the Vortex OpenSplice product. It provides an introduction to the essential concepts and enables users to begin using the Python DCPS API as quickly as possible.

1.2 Intended Audience

The Python Reference Guide is intended to be used by Python programmers who are using Vortex OpenSplice to develop applications.

CHAPTER

TWO

INTRODUCTION

The Python DCPS API provides users with Python classes to model DDS communication using Python and pure DDS applications.

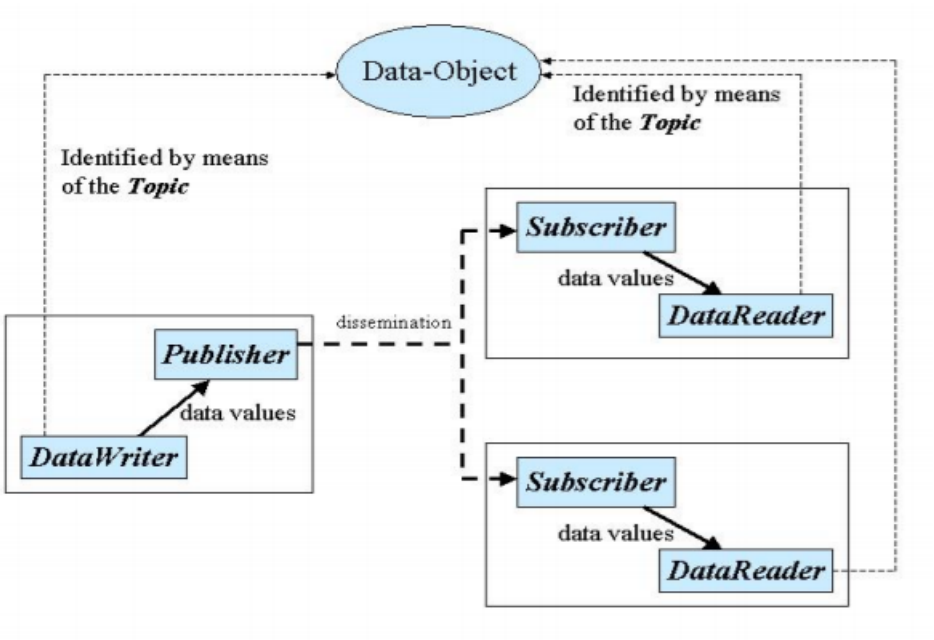
The Python DCPS API is a native Python binding that supports full DDS functionality. The language binding consists of a Python interface and a C wrapper implementation of the C99 API (C API for DDS). It makes use of Python language features and leverages ease of use by providing a higher level of abstraction.

2.1 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/

https://www.adlinktech.com/en/data-distribution-service

CHAPTER

THREE

INSTALLATION

This section describes the procedure to install the Python DCPS API on a Linux or Windows platform.

3.1 System Requirements

Operating System: Linux or Windows

3.2 Dependencies

Linux

- Python3 version 3.4.0 or later
- pip3 (package "python3-pip" on most linux package managers)
- GCC 4.8 or later

Windows

- Python version 3.5.0 or 3.6.0
- pip (usually already included in Python for Windows)
- Visual Studio 14 (2015) C compiler or later

Python modules

- Cython version 0.27 or later (install using "pip3 install Cython")

Development machines use the HDE (Host Devevelopment Environment) installation and license key. Deployment machines use the RTS (RunTime System) installation and license key.

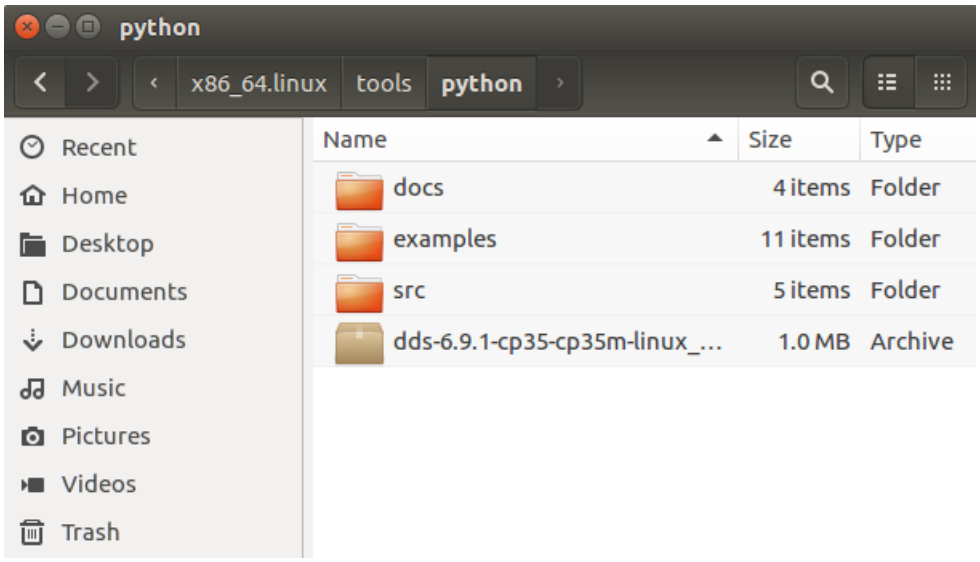
3.3 OpenSplice (OSPL) and Python DCPS API Installation

Steps:

1. Install OSPL. The Python DCPS API is included in this installer.
2. Setup OSPL license. Copy the license.lic file into the appropriate license directory.

   /INSTALLDIR/Vortex_v2/license
3. Python DCPS API files are contained in a tools/python folder

Example: *$OSPL_HOME/tools/python*



3.4 Python DCPS API Setup

3.4.1 Install Python DCPS API package from python wheel

In certain Vortex OpenSplice installs, the python api is bundled as a python wheel file in the $OSPL_HOME/tools/python directory. This way it avoids needing a native compiler and Cython prerequisites before installation.

Simply install using pip. For example, executing the command will install the wheel package, which is targeted for python 3.4 64-bit Linux:

```
$pip3 install dds-6.9.0-cp34-cp34m-linux_x86_64.whl
```

Or similarly, for python 3.5 64-bit Windows:

```
>pip install dds-6.9.0-cp35-cp35m-win_amd64.whl
```

If the binary wheel package is not present or is not compatible with your python version, then it is preferable to install the api from source.

3.4.2 Install Python DCPS API package from source

If the dependencies are satisfied, and the Vortex OpenSplice environment is set (OSPL_HOME and associated environment variables), then installation is just one command to execute in the $OSPL_HOME/python/src directory:

```
$python3 setup.py install
```

Note:

- This option requires the idlpp application and so will **not** work with a RTS installation.
- In either case, installing from wheel or from source may require administrator or superuser privileges if python was installed for all users.

- To check modules are installed correctly try importing the installed modules.

```
>>> import dds
>>> import ddsutil
```

3.5 Examples and Documentation

1. Examples

   The examples can be found in the following directory:

   - *$OSPL_HOME/tools/python/examples*

2. Python DCPS API Documentation

   The Python DCPS API can be found in the following directory:

   - *$OSPL_HOME/tools/python/docs/html*

3. Python DCPS User Guide (HTML and PDF)

   The user guide can be found in the following directories:

   - *$OSPL_HOME/docs/html*
   - *$OSPL_HOME/docs/pdf*

CHAPTER

FOUR

EXAMPLES

Several examples are provided to demonstrate the Python DCPS features.

The examples can be found in the following directory:

- *$OSPL_HOME/tools/python/examples*

4.1 Example Files

4.1.1 File Types

The examples directory contains files of different types. The runnable python script files reference the xml and idl files.

File Type	Description
py	A program file or script written in Python
xml	An XML file that contains one or more Quality of Service (QoS) profiles for DDS entities.
idl	An interface description language file used to define topic(s).

4.1.2 Examples

Example	Description
qos_example.py	An example that demonstrates how to specify QoS settings using Python DCPS APIs. Also shows usage of a waitset.
example1.py	Dynamic generation of Python Topic classes using IDL file and name. Uses a topic with a sequence.
example2.py	Dynamic generation of Python Topic classes using Enumeration and nested modules.
example3.py	Static generation of Python Topic classes as in example1 using the same topics.
example4.py	Demonstrate finding DDS topics over-the-wire. Find topics registered by other processes, and read and write samples to those topics.

4.2 Running Examples

To run an example python script:

1. Setup OSPL environment variables.

   Linux

   - source release.com

   Windows

   - release.bat

2. Ensure you are in the tools/python/examples folder
3. Run example scripts 1,2 and 4
   - > python3 example1.py
4. Run example script 3
   - > idlpp -l python example3.idl
   - > python3 example3.py

CHAPTER

FIVE

PYTHON API FOR VORTEX OPENSPLICE

The Python DCPS API provides users with Python classes to model DDS communication using Python and pure DDS applications.

The Python DCPS API consists of 2 modules.

- dds module
- ddsutil module

This section provides an overview of the main DDS concepts and Python API examples for these DDS concepts.

Note:

- The Python DCPS API can be found in the following directory:

  $OSPL_HOME/tools/python/docs/html

5.1 API Usage Patterns

The typical usage pattern for the Python DCPS API for Vortex OpenSplice is the following:

- option 1 - Recommended use. Statically generate Python topic classes using `idlpp -l python`. This must be used for RTS deployments.
- option 2 - If using IDL, the user also has the option to dynamically generate Python topic classes in python scripts. This will only work with HDE installations that include the idlpp application.
- option 3 - Model your DDS topics using IDL. Alternatively, the python scripts can use topics that already exist in the DDS system.

See *Python Generation from IDL*.

- Start writing your Python program using the Python API for Vortex OpenSplice.

The core classes you must use are `dds.Topic`, `dds.DomainParticipant` and either `dds.DataReader` or `dds.DataWriter`. 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 *Quality of Service (QoS)*.

The following list shows the sequence in which you would use the Vortex classes:

- Create a `dds.DomainParticipant` instance.
- Create one or more `dds.Topic` instances for the IDL topics your program will read or write.

- If you require publisher or subscriber level non-default QoS settings, create `dds.Publisher` and/or `dds.Subscriber` instances. (The most common reason for changing publisher/subscriber QoS is to define non-default partitions.)
- Create `dds.DataReader` and/or `dds.DataWriter` classes using the `dds.Topic` instances that you created.
- If you required data filtering, create `dds.QueryCondition` objects.
- Create the core of program, creating instances of your topic classes and writing them; or, reading data and processing it.

5.2 dds.Topic

The Python 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 Python class must be defined. These topic classes are either a class created statically using `idlpp`, dynamically using an idl file or dynamically for an existing topic discovered in the system. (see *Topic Generation and Discovery*)

API Examples

Create a Vortex OpenSplice topic named 'Msg1' based on the DDS Topic type `Msg` using dynamically generated Python topic classes.

```
TOPIC_NAME = 'Msg1'
TOPIC_TYPE = 'HelloWorldData::Msg'
IDL_FILE = 'idl/HelloWorldData.idl'

# Create domain participant
dp = DomainParticipant()

# Generate python classes from IDL file
gen_info = ddsutil.get_dds_classes_from_idl(IDL_FILE, TOPIC_TYPE)

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = gen_info.register_topic(dp, TOPIC_NAME, qos)
```

Create a Vortex OpenSplice topic named 'Msg1' based on the DDS Topic type `Msg` using statically generated Python topic classes.

```
TOPIC_NAME = 'Msg1'
TOPIC_TYPE = 'HelloWorldData::Msg'
IDL_FILE = 'idl/HelloWorldData.idl'

# Create domain participant
dp = DomainParticipant()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic(TOPIC_NAME, TOPIC_TYPE, qos)
```

5.3 dds.DomainParticipant

The Python``dds.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."

The dds.DomainParticipant has two optional parameters on creation. If these parameters are not provided, defaults are used.

Parameters:

- qos (Qos) – Participant QoS. Default: None
- listener (Listener) – Participant listener Default: None

API Examples

Create a Vortex OpenSplice domain participant. Returns participant or throws a `dds.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).

```
# Create domain participant
dp = DomainParticipant()
```

Create a participant on default domain with QoS profile.

```
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')

# Create participant
dp = DomainParticipant(qos = qp.get_participant_qos())
```

Create a participant on domain with QoS profile and listener. TODO

```
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')

# Create participant
dp = DomainParticipant(qos = qp.get_participant_qos())
```

5.4 dds.Publisher

The Python `dds.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 `dds.Publisher` class is optional. In it's place, you can use a `dds.DomainParticipant` instance. Reasons for explicitly creating a `dds.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 `dds.DDSException` if the publisher cannot be created.

Create a publisher with participant.

```
# Create participant
dp = DomainParticipant()

# Create publisher
pub = dp.create_publisher()
```

Create a publisher with participant and QoS profile.

```
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')

# Create participant
dp = DomainParticipant(qos = qp.get_participant_qos())

# Create publisher
pub = dp.create_publisher(qos = qp.get_publisher_qos())
```

5.5 dds.DataWriter

The Python `dds.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 `dds.DataWriter` class is required in order to write data to a DDS domain. It is attached to a DDS publisher or a DDS domain participant.

A `dds.DataWriter` class instance references an existing `dds.Topic` instance.

API Examples

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

(NB topic = dp.create_topic('Msg1', Msg, qos) can be replaced by topic = gen_info.register_topic(dp, Msg, qos) if dynamically generated)

Create a writer within a domain participant, and with default QoS.

```
# Create domain participant
dp = DomainParticipant()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a writer
writer = dp.create_datawriter(topic)
```

Create a writer within a publisher, and with default QoS.

```
# Create domain participant
dp = DomainParticipant()
```

(continues on next page)

(continued from previous page)

```
# Create publisher
pub = dp.create_publisher()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a writer
writer = pub.create_datawriter(topic)
```

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

```
# Create domain participant
dp = DomainParticipant()

# Create publisher
pub = dp.create_publisher()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a writer
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')
writer = pub.create_datawriter(topic)
writer = pub.create_datawriter(topic, qp.get_writer_qos())
```

Write a Msg topic class instance to a writer (statically generated).

```
# Topic data class
idMessage = 1
message1 = 'Hello World'
s = Msg(userID = idMessage, message = message1)

# Write data
writer.write(s)
```

Write a Msg topic class instance to a writer (dynamically generated).

```
# Topic data class
idMessage = 1
message1 = 'Hello World'
s = gen_info.topic_data_class(userID = idMessage, message = message1)

# Write data
writer.write(s)
```

Dispose a DDS topic instance.

```
# dispose instance
writer.dispose_instance(s)
```

Unregister a DDS topic instance.

```
# unregister instance
writer.unregister_instance(s)
```

5.6 dds.Subscriber

The Python `dds.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 `dds.Subscriber` class is optional. In it's place, you can use a `dds.DomainParticipant` instance. Reasons for explicitly creating a `dds.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 OpenSplice domain subscriber. Returns subscriber or throw a `dds.DDSException` if the subscriber cannot be created.

Create a subscriber with participant.

```
# Create participant
dp = DomainParticipant()

# Create Subscriber
sub = dp.create_subscriber()
```

Create a subscriber with participant and QoS profile.

```
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')

# Create participant
dp = DomainParticipant(qos = qp.get_participant_qos())

# Create Subscriber
sub = dp.create_subscriber(qos = qp.get_subscriber_qos())
```

5.7 dds.DataReader

The Python `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 `dds.DataReader` class is required in order to write data to a DDS domain. It is attached to a DDS subscriber or a DDS domain participant.

A `dds.DataReader` class instance references an existing `dds.Topic` instance.

API Examples

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

(NB topic = dp.create_topic('Msg1', Msg, qos) can be replaced by topic = gen_info.register_topic(dp, Msg1, qos) if dynamically generated)

Create a reader for a topic within a participant, and with default QoS.

```
# Create domain participant
dp = DomainParticipant()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a reader
reader = dp.create_datareader(topic)
```

Create a reader for a topic within a subscriber, and with default QoS.

```
# Create domain participant
dp = DomainParticipant()

# Create subscriber
sub = dp.create_subscriber()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a reader
reader = sub.create_datareader(topic)
```

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

```
# Create domain participant
dp = DomainParticipant()

# Create subscriber
sub = dp.create_subscriber()

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
    ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = dp.create_topic('Msg1', Msg, qos)

# Create a reader
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')
reader = sub.create_datareader(topic, qp.get_reader_qos())
```

Take data from a data reader.

```
l = reader.take(10)
```

Read data from a data reader.

```
l = reader.read(10)
```

Specify a wait timeout, in seconds, before read or take will return without receiving data TODO change description the code is wrong

```
# Create waitset
waitset = WaitSet()
qc = QueryCondition(reader, DDSMaskUtil.all_samples(), 'long1 > 1')

waitset.attach(qc)

# Wait for data
conditions = waitset.wait()

# Print data
l = reader.take(10)
```

5.8 dds.QueryCondition

The Python `dds.QueryCondition` 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 `dds.Query` class instance references an existing `dds.DataReader` instance.

API Examples

Create a `dds.QueryCondition` with a state mask and a filter expression for a reader and take data.

```
# Create waitset
waitset = WaitSet()
qc = QueryCondition(reader, DDSMaskUtil.all_samples(), 'long1 > 1')

waitset.attach(qc)

# Wait for data
conditions = waitset.wait()

# Print data
l = reader.take(10)
for sd, si in l:
    sd.print_vars()
```

CHAPTER

SIX

QUALITY OF SERVICE (QOS)

The following section explains how to set the Quality of Service (QoS) for a DDS entity.

Users have two options available to set the QoS for an entity or entities. They can define the QoS settings using an XML file, or they can use the Python DCPS APIs. Both of these options are explained.

If a QoS setting for an entity is not set using an xml file or the Python DCPS APIs, the defaults will be used. This allows a user the ability to override only those settings that require non-default values.

The code snippets referenced are taken from the runnable examples.

Note:

- The *Examples* section provides the examples directory location, example descriptions and running instructions.

6.1 Setting QoS Using QoS Provider XML File

QoS for DDS entities can be set using XML files based on the XML schema file QoSProfile.xsd. These XML files contain one or more QoS profiles for DDS entities.

Sample QoS Profile XML files can be found in the examples directory.

6.1.1 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: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>
```

6.1.2 Applying QoS Profile

To set the QoS profile for a DDS entity using the Python DCPS API and an XML file, the user specifies the File URI and the QoS profile name as parameters.

example1.py

```
...
qp = QosProvider('file://DDS_DefaultQoS_All.xml', 'DDS DefaultQosProfile')

# Create participant
dp = DomainParticipant(qos = qp.get_participant_qos())

# Create publisher
pub = dp.create_publisher(qos = qp.get_publisher_qos())

# Create Subscriber
sub = dp.create_subscriber(qos = qp.get_subscriber_qos())
...
```

6.2 Setting QoS Using Python DCPS API Classes

QoS settings can also be set by using the python classes alone. (No XML files required.)

Below is a code snippet that demonstrates how to specify the QoS settings for a writer using the python DCPS apis. In this example, all the QoS settings for the writer are set and all of the default QoS settings are overridden. If a QoS setting for an entity is not set, the default is used.

qos_example.py

```
...
writer_qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
         DeadlineQosPolicy(DDSDuration(500)),
         LatencyBudgetQosPolicy(DDSDuration(3000)),
         LivelinessQosPolicy(DDSLivelinessKind.MANUAL_BY_PARTICIPANT),
         ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE, DDSDuration.infinity()),
         DestinationOrderQosPolicy(DDSDestinationOrderKind.BY_SOURCE_TIMESTAMP),
         HistoryQosPolicy(DDSHistoryKind.KEEP_ALL),
         ResourceLimitsQosPolicy(10,10,10),
```

(continues on next page)

(continued from previous page)

```
        TransportPriorityQosPolicy(700),
        LifespanQosPolicy(DDSDuration(10, 500)),
        OwnershipQosPolicy(DDSOwnershipKind.EXCLUSIVE),
        OwnershipStrengthQosPolicy(100),
        WriterDataLifecycleQosPolicy(False)
        ])
...
```

In the next example, a topic QoS is created that overrides only a subset of the QoS settings.

HelloWorldDataSubscriber.py

```
...
# Create domain participant
dp = DomainParticipant()

# Create subscriber
sub = dp.create_subscriber()

# Generate python classes from IDL file
gen_info = ddsutil.get_dds_classes_from_idl(IDL_FILE, TOPIC_TYPE)

# Create a topic QoS that overrides defaults for durability and reliability
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
       ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])
topic = gen_info.register_topic(dp, TOPIC_NAME, qos)
...
```

CHAPTER

SEVEN

TOPIC GENERATION AND DISCOVERY

A DDS Topic represents the unit for information that can be produced or consumed by a DDS application. Topics are defined by a name, a type, and a set of QoS policies.

The Python DCPS API provides several ways of generating Python classes to represent DDS topics.

- over the wire discovery
- dynamic generation of Python Topic classes using parameters IDL file and topic name
- static generation of Python Topic classes using IDL

Note:

- The *Examples* section provides the examples directory location, example descriptions and running instructions.

7.1 Over the Wire Discovery

Python topic classes can be generated for existing DDS topics in the DDS system. These topics are "discovered over the wire".

The Python classes are generated when the topic is requested by name.

A code snippet is provided from example4.py. This example finds a topic registered by another process, and reads and writes samples to that topic.

example4.py

```
...
print('Connecting to DDS domain...')
dp = dds.DomainParticipant()

print('Finding OsplTestTopic...')
found_topic = dp.find_topic('OsplTestTopic')

print('Registering OsplTestTopic locally')
local_topic = ddsutil.register_found_topic_as_local(found_topic)

print('Getting Python classes for the found topic...')
gen_info = ddsutil.get_dds_classes_for_found_topic(found_topic)
OsplTestTopic = gen_info.get_class(found_topic.type_name)
Tstate = gen_info.get_class('ospllog::Tstate')
```

(continues on next page)

(continued from previous page)

```
print('Creating sample data to write...')
data = OsplTestTopic(id=11,index=22, x=1.2, y=2.3, z= 3.4, t=9.8,
    state=Tstate.init, description='Hello from Python')

print('Creating readers and writers...')
pub = dp.create_publisher()
wr = pub.create_datawriter(local_topic, found_topic.qos)
sub = dp.create_subscriber()
rd = sub.create_datareader(local_topic, found_topic.qos)

print('Writing sample data...')
wr.write(data)
print('Wrote: %s' % (str(data)))
...
```

7.2 Static Generation of Python Topic Classes Using IDL

The Python topic classes can be generated statically using an IDL file. Please see *Static Generation* for more information.

7.3 Dynamic Generation of Python Topic Classes Using IDL and Name

The Python topic classes can be generated dynamically using an IDL file, and the topic name. Please see *Dynamic Generation* for more information.

CHAPTER

EIGHT

PYTHON GENERATION FROM IDL

The Python DCPS API supports generation of Python topic classes from IDL. This chapter describes the details of the IDL-Python binding.

8.1 Running IDLPP

8.1.1 Static Generation

The Python topic classes can be generated statically using an IDL file.

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

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

Note:

- A Python package with the same name as the idl file (without the .idl extension) is always created.
- It defines types not included in an IDL module. IDL modules become Python packages within this base package.

8.1.2 Dynamic Generation

The Python topic classes can be generated dynamically using an IDL file, and the topic name. An api, `ddsutil.get_dds_classes_from_idl`, is provided to generate the topic classes at runtime from within a Python script.

Please note that dynamic generation will **not** work with a deployment license as it requires the idlpp application and should only be used during development of Topics.

HelloWorldDataPublisher.py

```
...
TOPIC_NAME = 'Msg1'
TOPIC_TYPE = 'HelloWorldData::Msg'
IDL_FILE = 'idl/HelloWorldData.idl'

# Create domain participant
dp = DomainParticipant()

# Create publisher
```

(continues on next page)

(continued from previous page)

```
pub = dp.create_publisher()

# Generate python topic classes from IDL file
gen_info = ddsutil.get_dds_classes_from_idl(IDL_FILE, TOPIC_TYPE)

# Type support class
qos = Qos([DurabilityQosPolicy(DDSDurabilityKind.TRANSIENT),
       ReliabilityQosPolicy(DDSReliabilityKind.RELIABLE)])

# Register topic
topic = gen_info.register_topic(dp, TOPIC_NAME, qos)

# Create a writer
writer = pub.create_datawriter(topic)

# Topic data class
idMessage = 1
message1 = 'Hello World'
s = gen_info.topic_data_class(userID = idMessage, message = message1)

# Write data
writer.write(s)

#output to console
print('=== [Publisher] writing a message containing : ')
print('userID  :',  idMessage)
print('message :', message1)
...
```

8.1.3 Generated Artifacts

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

IDL Concept	Python Concept	Comment
module	package	Folder with module name, plus __init__.py that defines types defined within the module.
enum	class	Defined in __init__.py files.
enum value	enum value	Defined in __init__.py files.
struct	class	Defined in __init__.py files.
field	class property	Defined in __init__.py files.
union	union	Defined in __init__.py files. (Only statically generated supported)

Datatype mappings

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

IDL Type	Python Type
boolean	bool
char	str, length==1
octet	int
short	int
ushort	int
long	int
ulong	int
long long	int
ulong long	int
float	float
double	float
string	str
wchar	Unsupported
wstring	Unsupported
any	Unsupported
long double	Unsupported

Implementing Arrays and Sequences in Python

Both IDL arrays and IDL sequences are mapped to Python lists.

The constructors for generated classes always fully allocate any array fields. Sequences are always initialized to the empty list.

8.2 Limitations of Python Support

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

- IDL unions are Supported by statically generated Python, but not by dynamic or over-the-wire.
- The following IDL data types are not supported: wchar, wstring, any and long double .

CHAPTER

NINE

CONTACTS & NOTICES

9.1 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: https://www.adlinktech.com/en/data-distribution-service

Contact: https://www.adlinktech.com/en/data-distribution-service

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

9.2 Notices