ADLINK

VORTEX

Node.js DCPS API Guide

Release Beta 6.x

Contents

1

Preface

1.1 About the Node.js DCPS API Guide

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

This guide contains:

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

This reference guide is based on the OMG's Data Distribution Service 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 Node.js DCPS API as quickly as possible.

1.2 Intended Audience

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

2

Introduction

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

The Node.js DCPS API is a native JavaScript binding that supports DDS functionality. The language binding consists of Node.js classes and wrapper implementations of the C99 API (C API for DDS). It makes use of ECMAScript 2015 (ES6) JavaScript language features and leverages ease of use by providing a higher level of abstraction.

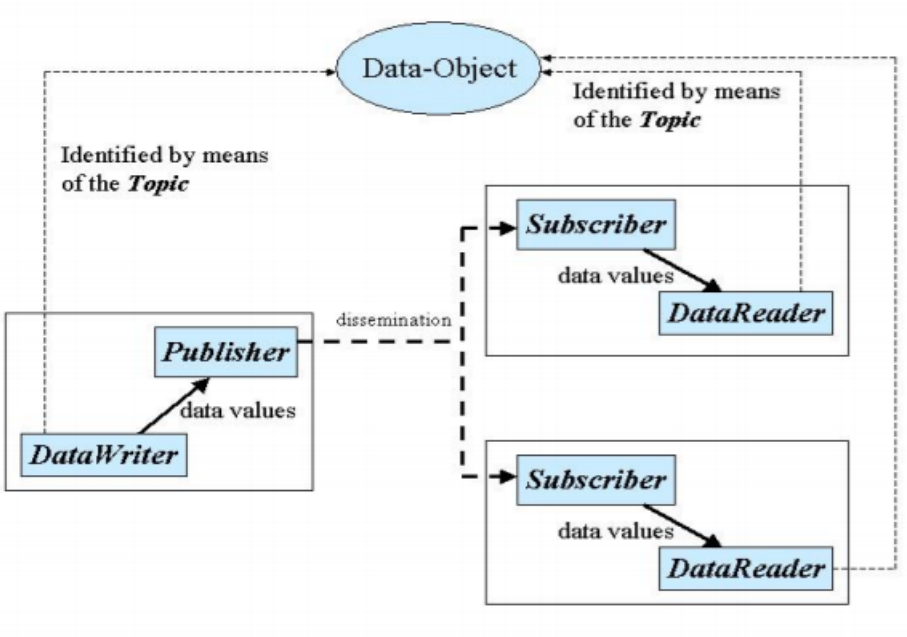
Please see *Limitations of Node.js Support* for limitations of Node.js DDS API support.

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/

http://ist.adlinktech.com/

3

Installation

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

3.1 Dependencies

The Node.js DCPS API has several dependencies that must be installed.

3.1.1 Linux 64-bit

- Node.js LTS 8.11.1 or later (preferably Node.js LTS 8.x version)
- npm (node package manager) version 5.6.0 or later (typically included with a Node.js install)
- Python 2.7 (v3.x.x is not supported)
- make
- C/C++ compiler toolchain like GCC

3.1.2 Windows 64-bit

- Node.js LTS 8.11.1 or later (preferably Node.js LTS 8.x version)
- npm (node package manager) version 5.6.0 or later (typically included with a Node.js install)
- Python 2.7 (v3.x.x is not supported)
- Visual C++ build tools (VS 2015 or VS 2017)

3.1.2.1 Python and Visual C++ build tools install

- Install all the required tools and configurations using Microsoft's windows-build-tools by running the following from a command prompt as an administrator:

  npm install - -global - -production windows-build-tools

  (or)

- Install tools and configuration manually
  - Visual C++ build tools (VS 2015 or VS 2017)
  - Python 2.7 (v3.x.x is not supported)

More detailed information on installing Python and Visual C++ build tools on Windows can be found at: https://github.com/Microsoft/nodejs-guidelines/blob/master/windows-environment.md#compiling-native-addon-modules

3.2 OpenSplice (OSPL) and Node.js DCPS API Installation

Steps:

1. Install OSPL. Choose an HDE type installer which is a Host Development Environment. This contains all of the services, libraries, header files and tools needed to develop applications using OpenSplice. The Node.js DCPS API is included in the following installers:

Platform	Platform Code
Ubuntu1404 64 bit	P704
Ubuntu1604 64 bit	P768
Windows10 64 bit	P738

   Example installer:

   P704-VortexOpenSplice-6.9.x-HDE-x86.linux-gcc4.1.2-glibc2.5-installer.run

2. Setup OSPL license. Copy the license.lic file into the appropriate license directory

   /INSTALLDIR/Vortex_v2/license

3. Node.js DCPS API files are contained in a tools/nodejs folder

   Example: *$OSPL_HOME/tools/nodejs*

3.3 Installing Node.js DCPS API in a Node.js application

1. Start a command shell. Setup OSPL environment variables by running release.com or release.bat which can be found in

   Linux

   /INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.9.x/HDE/x86_64.linux/

   Windows

   \INSTALLDIR\ADLINK\Vortex_v2\Device\VortexOpenSplice\6.9.x\HDE\x86_64.windows

2. Create a node project folder, if not created

   mkdir <project_name>

   cd <project_name>

   npm init

3. Change directory to node project folder
4. Install the Node.js DCPS API to your project by executing:

   Linux

   npm install $OSPL_HOME/tools/nodejs/vortexdds-x.y.z.tgz

   Windows

   npm install %OSPL_HOME%\tools\nodejs\vortexdds-x.y.z.tgz

3.4 Examples and Documentation

1. Examples directory:

   $OSPL_HOME/tools/nodejs/examples

2. Node.js DCPS API documentation directory:

   $OSPL_HOME/docs/nodejs/html

3. Node.js DCPS User Guide (HTML and PDF) directory:

   $OSPL_HOME/docs

4

Examples

Examples are provided to demonstrate the Node.js DCPS features.

The examples can be found in the following directory:

$OSPL_HOME/tools/nodejs/examples

WHERE:

Linux

OSPL_HOME = *INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.9.x/HDE/x86_64.linux*

Windows

OSPL_HOME = *INSTALLDIR\ADLINK\Vortex_v2\Device\VortexOpenSplice\6.9.x\HDE\x86_64.win64*

4.1 Example Files

Each subfolder consists of a Node.js example project.

4.1.1 Examples

Example	Description
HelloWorld	Demonstrates how to read and write a simple topic in DDS
jsshapes	Demonstrates how to read and write to a DDS application
IoTData	Demonstrates reading and writing an IoTData topic
PingPong	Demonstrates reading, writing, and waitsets
QoSExample	Demonstrates QoS settings
GetSetQoSExample	Demonstrates getting and setting QoS using QoS Provider and DCPS api

4.1.2 File Types

The examples directory contains files of different types.

File Type	Description
js	A program file or script written in JavaScript
package.json	Lists the packages that a project depends on
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)

4.2 Running Examples

To run a Node.js example:

1. Setup OSPL environment variables

   Linux

   - For each example javascript file to be run, open a Linux terminal
   - Navigate to directory containing release.com file (OSPL_HOME)

     /INSTALLDIR/ADLINK/Vortex_v2/Device/VortexOpenSplice/6.9.x/HDE/x86_64.linux
   - Run release.com (". release.com")

   Windows

   - For each example javascript file to be run, open a command prompt
   - Navigate to directory containing release.bat file (OSPL_HOME)

     \INSTALLDIR\ADLINK\Vortex_v2\Device\VortexOpenSplice\6.9.x\HDE\x86_64.win64
   - Run release.bat ("release.bat"), if OSPL environment variables are not set system wide
2. Change directory to the example folder

   Example:

   $OSPL_HOME/tools/nodejs/examples/HelloWorld
3. Run npm install to install the Node.js DCPS API (vortexdds-x.y.z.tgz) and all other dependencies

   npm install

Note: If you are running the examples from another directory outside the OSPL install, then you will need to manually install the Node.js DCPS API first and then run npm install for the example dependencies as follows:

npm install $OSPL_HOME/tools/nodejs/vortexdds-x.y.z.tgz

npm install

---

4. Run .js file(s) in a terminal/command shell

   Example:

   node HelloWorldSubscriber.js

5

Node.js API for Vortex DDS

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

The Node.js DCPS API consists of one module.

- vortexdds

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

Note:

- The Node.js DCPS API documentation can be found in the following directory:

  $OSPL_HOME/docs/nodejs/html

5.1 API Usage Patterns

The typical usage pattern for the Node.js DCPS API for Vortex DDS is the following:

- Model your DDS topics using IDL and generate Node.js topic classes from IDL.
- AND/OR generate Node.js topic classes for topics that already exist in the DDS system.
- Start writing your Node.js program using the Node.js API for Vortex DDS.

The core classes are `Participant`, `Topic`, `Reader` and `Writer`. `Publisher` and `Subscriber` classes can be used 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 `Participant` instance.
- Create one or more `Topic` instances.
- If you require publisher or subscriber level non-default QoS settings, create `Publisher` and/or `Subscriber` instances. (The most common reason for changing publisher/subscriber QoS is to define non-default partitions.)
- Create `Reader` and/or `Writer` classes using the `Topic` instances that you created.
- If you required data filtering, create `QueryCondition` objects.
- Create the core of program, writing and/or reading data and processing it.

5.2 Participant

The Node.js `Participant` 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."

Parameters:

- {number} domainId 0 <= domainId <= 230 (Default: DDSConstants.DDS_DOMAIN_DEFAULT)
- {QoS} qos (Default: undefined)
- {object} listener (Default: undefined)

The optional parameters have defaults if not specified. If the qos is undefined, the OSPL default QoS is used.

Examples

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

Create a domain participant in the default DDS domain (the one specified by the OSPL_URI environment variable).

```
const dds = require('vortexdds');

// create domain participant
const participant = new dds.Participant();
```

Create a participant on the default domain with a QoS profile. Consider the code snippet below taken from example: GetSetQoSExample/GetSetQoSExample.js file.

```
const QOS_PATH = 'DDS_Get_Set_QoS.xml';
const QOS_PROFILE = 'DDS GetSetQosProfile';
const DOMAIN_ID = dds.DDS_DOMAIN_DEFAULT;

async function main(){

  // create a qos provider using qos xml file
  let qp = new dds.QoSProvider(QOS_PATH, QOS_PROFILE);

  // get participant qos from qos provider
  let pqos = qp.getParticipantQos();
  let participant = new dds.Participant(DOMAIN_ID, pqos);

}
```

5.3 Topic

The Node.js `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.

Javascript `Topic` instances can be created using an IDL file.

Step 1 - Generate TypeSupport objects from IDL file

The function `getTopicTypeSupportsForIDL(idlPath)` is provided to generate a `TypeSupport` instance for every topic defined in an IDL file. This function returns a promise, therefore should be called from an async function.

Step 2 - Create Topic instance using TypeSupport

The `createTopicFor` method in the `Participant` class can then be used to create a topic instance.

```
/**
* Create a Topic on this participant given a TypeSupport object.
* @param {string} topicName
```

```
* @param {TypeSupport} typeSupport
* @param {QoS} QoS (default undefined)
* @param {object} listener (default undefined)
* @returns {Topic} topic instance
*/
createTopicFor(
        topicName,
        typeSupport,
        qos = null,
        listener = null
) {
```

Example

Create a Vortex DDS topic named 'HelloWorldData_Msg' based on the DDS Topic type `Msg` from the `HelloWorldData.idl` file. This topic is created using the getTopicTypeSupportsForIDL function, and the `Participant` createTopicFor method.

Consider the code snippet below taken from example: HelloWorld/HelloWorldTopic.js file.

```
const dds = require('vortexdds');
const path = require('path');

module.exports.create = async function(participant) {

  const topicName = 'HelloWorldData_Msg';
  const idlName = 'HelloWorldData.idl';
  const idlPath = path.resolve(idlName);

  //wait for dds.getTopicTypeSupportsForIDL to return a Map of typeSupports
  let typeSupports = await dds.getTopicTypeSupportsForIDL(idlPath);

  //HelloWorldData.idl contains 1 topic
  let typeSupport = typeSupports.get('HelloWorldData::Msg');

  //create topic qos
  let tqos = dds.QoS.topicDefault();
  tqos.durability = dds.DurabilityKind.Transient;
  tqos.reliability = dds.ReliabilityKind.Reliable;

  //create topic
  return participant.createTopicFor(topicName, typeSupport, tqos);

};
```

5.4 Publisher

The Node.js `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 `Publisher` class is optional. In its place, you can use a `Participant` instance. Reasons for explicitly creating a `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.

Examples

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

Create a publisher with participant.

```
const dds = require('vortexdds');

//create participant
const participant = new dds.Participant();

//create publisher
const pub = participant.createPublisher();
```

Create a publisher with a participant and QoS profile. Consider the code snippet below taken from example: GetSetQoSExample/GetSetQoSExample.js file.

```
const QOS_PATH = 'DDS_Get_Set_QoS.xml';
const QOS_PROFILE = 'DDS GetSetQosProfile';
const DOMAIN_ID = dds.DDS_DOMAIN_DEFAULT;

async function main(){

  // create a qos provider using qos xml file
  let qp = new dds.QoSProvider(QOS_PATH, QOS_PROFILE);

  // get participant qos from qos provider
  let pqos = qp.getParticipantQos();
  let participant = new dds.Participant(DOMAIN_ID, pqos);

  // get publisher qos from qos provider and create a publisher
  let pubqos = qp.getPublisherQos();
  let publisher = new dds.Publisher(participant, pubqos);
}
```

5.5 Writer

The Node.js `Writer` class represents a DDS data writer entity.

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

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

Examples

Create a Vortex DDS domain writer and write data. Returns writer or throws a `DDSError` if the writer cannot be created. The example below uses the 'HelloWorldData_Msg' topic from the 'HelloWorldTopic.js' file as shown in *Topic* example. Consider the code snippet below taken from example: HelloWorld/HelloWorldPublisher.js file.

```
const dds = require('vortexdds');
const HelloWorldTopic = require('./HelloWorldTopic');

main();

async function main(){

    //create domain participant
    let participant = new dds.Participant();

    //wait for topic to be created
    let topic = await HelloWorldTopic.create(participant);

    //create a publisher with publisher QoS
    let pqos = dds.QoS.publisherDefault();
```

```
    pqos.partition = 'HelloWorld example';
    let pub = participant.createPublisher(pqos, null);

    //create a writer with writer QoS
    let wqos = dds.QoS.writerDefault();
    wqos.durability = dds.DurabilityKind.Transient;
    wqos.reliability = dds.ReliabilityKind.Reliable;
    let writer = pub.createWriter(topic, wqos, null);

    //write one sample
    let msg = {userID: 1, message: 'Hello World'};

    console.log('=== HelloWorldPublisher');
    console.log('=== [Publisher] writing a message containing :');
    console.log('    userID  : ' + msg.userID);
    console.log('    Message : ' + msg.message);

    writer.write(msg);

}
```

5.6 Subscriber

The Node.js `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 `Subscriber` class is optional. In its place, you can use a `Participant` instance. Reasons for explicitly creating a `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.

Examples

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

Create a subscriber with participant.

```
const dds = require('vortexdds');

//create participant
consr participant = new dds.Participant();

//create Subscriber
const sub = participant.createSubscriber();
```

Create a subscriber with participant and QoS profile, where the DDS_DefaultQoS_All.xml file is located in the project directory.

```
const QOS_PATH = 'DDS_Get_Set_QoS.xml';
const QOS_PROFILE = 'DDS GetSetQosProfile';
const DOMAIN_ID = dds.DDS_DOMAIN_DEFAULT;

async function main(){

  // create a qos provider using qos xml file
  let qp = new dds.QoSProvider(QOS_PATH, QOS_PROFILE);

  // get participant qos from qos provider
```

```
  let pqos = qp.getParticipantQos();
  let participant = new dds.Participant(DOMAIN_ID, pqos);

  /*
     Get subscriber qos from qos provider and create a subscriber
     In the qos xml file provided with this example, there are
     two subscriber qos. The subscriber qos with
     id 'subscriber1' is chosen below

     NOTE: If there are more than one qos entries in the xml file
     for a dds entity, then the entity qos id must be specified.
     Otherwise an error will be thrown while trying to get the
     respective entity qos from the qos provider
   */
  let subqos = qp.getSubscriberQos('subscriber1');
  let subscriber = new dds.Subscriber(participant, subqos);
  }
```

5.7 Reader

The Node.js `Reader` class represents a DDS data reader entity.

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

A `Reader` class is required in order to write data to a DDS domain. It is attached to a DDS subscriber or a DDS participant.

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

Examples

Create a Vortex DDS domain reader with a subscriber and read/take data. Returns reader or throw a `DDSError` instance if the reader cannot be created. Consider the code snippet below taken from example: HelloWorld/HelloWorldSubscriber.js file.

```
const dds = require('vortexdds');
const HelloWorldTopic = require('./HelloWorldTopic');

main();

async function main(){

    //create domain participant
    let participant = new dds.Participant();

    //wait for topic to be created
    let topic = await HelloWorldTopic.create(participant);

    //create subscriber with subscriber QoS
    let sqos = dds.QoS.subscriberDefault();
    sqos.partition = 'HelloWorld example';
    let sub = participant.createSubscriber(sqos, null);

    //create reader with reader QoS
    let rqos = dds.QoS.readerDefault();
    rqos.durability = dds.DurabilityKind.Transient;
    rqos.reliability = dds.ReliabilityKind.Reliable;
    let reader = sub.createReader(topic, rqos, null);

    console.log('=== [Subscriber] Ready ...');

    attemptTake(participant, reader, 0);
```

```
    participant.delete();
   }

   /* Attempt to take from the reader every 200 ms.
    * If a successful take occurs, we are done.
    * If a successful take does not occur after 100 attempts,
    * return.
    */
   function attemptTake(participant, reader, numberAttempts){

    setTimeout(function() {
            let takeArray = reader.take(1);
            if (takeArray.length > 0 && (takeArray[0].info.valid_data === 1)) {
            console.log('=== [Subscriber] message received :');
            console.log('    userID  : ' + takeArray[0].sample.userID);
            console.log('    Message : ' + takeArray[0].sample.message);
            } else {
            if (numberAttempts < 100){
            return attemptTake(participant, reader, numberAttempts + 1);
            }
            }
            participant.delete();
            return;
    }, 200);
   }
```

Take data with read condition from a data reader.

```
const numberOfSamples = 10;
const cond = new dds.ReadCondition(reader, dds.StateMask.any);
let readArray = reader.takeCond(numberOfSamples, cond);
```

Read data with read condition from a data reader.

```
const numberOfSamples = 10;
const cond = new dds.ReadCondition(reader, dds.StateMask.any);
let readArray = reader.readCond(numberOfSamples, cond);
```

5.8 WaitSet

The Node.js `WaitSet` class represents a DDS wait set.

A WaitSet object allows an application to wait until one or more of the attached Condition objects evaluates to true or until the timeout expires.

Example

Create a `WaitSet` with a read condition to wait until publication is matched. Consider the code snippet below taken from example: PingPong/ping.js file.

```
const dds = require('vortexdds');
const PingPongTopic = require('./PingPongTopic');

// we iterate for 20 samples
const numSamples = 20;
main();

async function main() {
  // create our entities
  let participant = new dds.Participant();
  let topic = await PingPongTopic.create(participant);
```

```
  // create publisher on partition 'Ping' and writer on the publisher
  // on pong, we have a subscriber on the same partition and topic
  let pqos = dds.QoS.publisherDefault();
  pqos.partition = 'Ping';
  let publisher = participant.createPublisher(pqos, null);
  let writer = publisher.createWriter(topic);

  // create a subscriber on partition 'Pong' and a reader on the subscriber
  let sqos = dds.QoS.subscriberDefault();
  sqos.partition = 'Pong';
  let subscriber = participant.createSubscriber(sqos);
  let reader = subscriber.createReader(topic);

  // create waitset which waits until pong subscriber on partition
  // 'Ping' is found
  let pubMatchedWaitset = new dds.Waitset();
  let statCond = new dds.StatusCondition(writer);
  statCond.enable(dds.StatusMask.publication_matched);
  pubMatchedWaitset.attach(statCond, writer);

  // waitset for new data
  let newDataWaitset = new dds.Waitset();
  let newDataCond = new dds.ReadCondition(reader,
    dds.StateMask.sample.not_read);
  newDataWaitset.attach(newDataCond, reader);

  // block until pong has been found
  console.log('Waiting for pong subscriber to be found...');
  pubMatchedWaitset.wait(dds.DDSConstants.DDS_INFINITY, function(err, res) {
    if (err) throw err;
    console.log('Found pong subscriber.');
    // Ping writes the first message
    let msg = {userID: 0, message: 'from ping'};
    console.log('sending ' + JSON.stringify(msg));
    writer.write(msg);
    onDataAvailable(participant, newDataWaitset, reader, writer, 0);
  });
}

/* onDataAvailable waits for new data. When we have new data, it
 * reads it, prints it, and sends back a message with the userID
 * field incremented by one and 'from ping' in the message field.
 * It then calls onDataAvailable again.
 * We quit when iterLower === numSamples (20 iterations in this example).
 */
function onDataAvailable(
  participant,
  ws,
  reader,
  writer,
  iterLower
) {
  if (iterLower === numSamples) {
    console.log('Done');
    participant.delete();
    return;
  }
  // when we get a message from pong, read it, then send it back
  // with userID+1 and message 'from ping'
  ws.wait(dds.DDSConstants.DDS_INFINITY, function(err, res) {
    if (err) throw err;
    let takeArray = reader.take(1);
    if (takeArray.length > 0 && (takeArray[0].info.valid_data === 1)) {
```

```
      let sample = takeArray[0].sample;
      console.log('received: ' + JSON.stringify(sample));
      let msg = {userID: sample.userID + 1, message: 'from ping'};
      console.log('sending ' + JSON.stringify(msg));
      writer.write(msg);
    }
    onDataAvailable(participant, ws, reader, writer, iterLower + 1);
  });
}
```

5.9 QueryCondition

The Node.js `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 `QueryCondition` class instance references an existing `Reader` instance.

Example

Create a `QueryCondition` with a state mask and a filter expression for a reader and take data. Consider the code snippet below taken from example: jsshapes/read.js file.

```
const dds = require('vortexdds');
const ShapeTopicHelper = require('./ShapeTopicHelper');
const util = require('util');

// we are reading 100 samples of a blue circle
const shapeName = 'Circle';
const numSamples = 100;
const mask = dds.StateMask.sample.not_read;
const sqlExpression = 'color=%0';
const params = ['BLUE'];

main();

async function main() {
  const participant = new dds.Participant();

  // set up our topic
  const circleTopic = await ShapeTopicHelper.create(participant, 'Circle');

  // our reader has volatile qos
  let readerqosprovider = new dds.QoSProvider(
    './DDS_VolatileQoS_All.xml',
    'DDS VolatileQosProfile'
  );

  // set up circle reader
  const circleReader = participant.createReader(circleTopic,
    readerqosprovider.getReaderQos());

  // set up waitset
  const newDataWs = new dds.Waitset();
  const queryCond = new dds.QueryCondition(
    circleReader,
    mask,
    sqlExpression,
    params,
    1
  );
```

```
  // attach the query condition
  newDataWs.attach(queryCond, circleReader);

  console.log('Waiting for demo_ishapes to publish a blue circle...');
  onDataAvailable(participant, newDataWs, circleReader, queryCond, 0);
};

/* onDataAvailable waits until we have new data (with the query condition).
 * when we do, we read it, print the data (shape data), and then call
 * onDataAvailable with iterLower incremented by one, to wait for more data.
 * This repeats until iterLower === numSamples (100 samples in this case).
 * When iterLower === numSamples, we delete the participant and quit.
 */
function onDataAvailable(
  participant,
  ws,
  reader,
  queryCond,
  iterLower
) {
  if (iterLower === numSamples) {
    console.log('Done.');
    participant.delete();
    return;
  }
  ws.wait(dds.DDSConstants.DDS_INFINITY, function(err, res) {
    if (err) throw err;
    let sampleArray = reader.takeCond(1, queryCond);
    if (sampleArray.length > 0 && (sampleArray[0].info.valid_data === 1)) {
      let sample = sampleArray[0].sample;
      console.log(
        util.format(
          '%s %s of size %d at (%d,%d)',
          sample.color,
          shapeName,
          sample.shapesize,
          sample.x,
          sample.y
        )
      );
    }
    onDataAvailable(participant, ws, reader, queryCond, iterLower + 1);
  });
}
```

6

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 Node.js DCPS APIs. Both of these options are explained.

If a QoS setting for an entity is not set using an xml file or the Node.js 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. OSPL includes an XSD (XML schema), that is located in $OSPL_HOME/etc/DDS_QoSProfile.xml. This can be used with XML schema core editors to help create valid XML files.

Sample QoS Profile XML files can be found in the examples directory. Typically you will place the qos files in a subdirectory of your Node.js application.

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: Persistent QoS XML file

The example below specifies the persistent QoS XML file which is used by the QoSProvider.

```
<?xml version="1.0" encoding="UTF-8"?>
<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 PersistentQosProfile">
     <domainparticipant_qos>
          <user_data>
               <value></value>
          </user_data>
          <entity_factory>
               <autoenable_created_entities>true</autoenable_created_entities>
          </entity_factory>
     </domainparticipant_qos>
     <subscriber_qos name="subscriber1">
          <presentation>
               <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
               <coherent_access>true</coherent_access>
               <ordered_access>true</ordered_access>
          </presentation>
          <partition>
               <name>partition1</name>
          </partition>
          <group_data>
               <value></value>
          </group_data>
          <entity_factory>
               <autoenable_created_entities>true</autoenable_created_entities>
          </entity_factory>
     </subscriber_qos>
     <subscriber_qos name="subscriber2">
          <presentation>
               <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
               <coherent_access>false</coherent_access>
               <ordered_access>false</ordered_access>
          </presentation>
          <partition>
               <name></name>
          </partition>
          <group_data>
               <value></value>
          </group_data>
          <entity_factory>
               <autoenable_created_entities>true</autoenable_created_entities>
          </entity_factory>
     </subscriber_qos>
     <publisher_qos>
          <presentation>
               <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
               <coherent_access>true</coherent_access>
               <ordered_access>true</ordered_access>
          </presentation>
          <partition>
               <name>partition1</name>
          </partition>
          <group_data>
               <value></value>
          </group_data>
          <entity_factory>
               <autoenable_created_entities>true</autoenable_created_entities>
          </entity_factory>
```

```
    </publisher_qos>
    <datawriter_qos>
         <durability>
              <kind>PERSISTENT_DURABILITY_QOS</kind>
         </durability>
         <deadline>
              <period>
                   <sec>DURATION_INFINITE_SEC</sec>
                   <nanosec>DURATION_INFINITE_NSEC</nanosec>
              </period>
         </deadline>
         <latency_budget>
              <duration>
                   <sec>0</sec>
                   <nanosec>0</nanosec>
              </duration>
         </latency_budget>
         <liveliness>
              <kind>AUTOMATIC_LIVELINESS_QOS</kind>
              <lease_duration>
                   <sec>DURATION_INFINITE_SEC</sec>
                   <nanosec>DURATION_INFINITE_NSEC</nanosec>
              </lease_duration>
         </liveliness>
         <reliability>
              <kind>RELIABLE_RELIABILITY_QOS</kind>
              <max_blocking_time>
                   <sec>0</sec>
                   <nanosec>100000000</nanosec>
              </max_blocking_time>
        </reliability>
        <destination_order>
             <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
         </destination_order>
         <history>
              <kind>KEEP_LAST_HISTORY_QOS</kind>
              <depth>100</depth>
         </history>
         <resource_limits>
              <max_samples>LENGTH_UNLIMITED</max_samples>
              <max_instances>LENGTH_UNLIMITED</max_instances>
              <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
         </resource_limits>
         <transport_priority>
              <value>0</value>
         </transport_priority>
         <lifespan>
              <duration>
                   <sec>DURATION_INFINITE_SEC</sec>
                   <nanosec>DURATION_INFINITE_NSEC</nanosec>
              </duration>
         </lifespan>
         <user_data>
              <value></value>
        </user_data>
        <ownership>
             <kind>SHARED_OWNERSHIP_QOS</kind>
         </ownership>
         <ownership_strength>
              <value>0</value>
         </ownership_strength>
         <writer_data_lifecycle>
              <autodispose_unregistered_instances>true</autodispose_unregistered_instances>
```

```
        </writer_data_lifecycle>
    </datawriter_qos>
    <datareader_qos>
        <durability>
            <kind>PERSISTENT_DURABILITY_QOS</kind>
        </durability>
        <deadline>
            <period>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </period>
        </deadline>
        <latency_budget>
            <duration>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </duration>
        </latency_budget>
        <liveliness>
            <kind>AUTOMATIC_LIVELINESS_QOS</kind>
            <lease_duration>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </lease_duration>
        </liveliness>
        <reliability>
            <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
            <max_blocking_time>
                <sec>0</sec>
                <nanosec>100000000</nanosec>
            </max_blocking_time>
        </reliability>
        <destination_order>
             <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
        </destination_order>
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <resource_limits>
            <max_samples>LENGTH_UNLIMITED</max_samples>
            <max_instances>LENGTH_UNLIMITED</max_instances>
            <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
        </resource_limits>
        <user_data>
            <value></value>
        </user_data>
        <ownership>
            <kind>SHARED_OWNERSHIP_QOS</kind>
        </ownership>
        <time_based_filter>
            <minimum_separation>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </minimum_separation>
        </time_based_filter>
        <reader_data_lifecycle>
            <autopurge_nowriter_samples_delay>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </autopurge_nowriter_samples_delay>
            <autopurge_disposed_samples_delay>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
```

```
            </autopurge_disposed_samples_delay>
        </reader_data_lifecycle>
    </datareader_qos>
    <topic_qos>
        <topic_data>
            <value></value>
        </topic_data>
        <durability>
            <kind>PERSISTENT_DURABILITY_QOS</kind>
        </durability>
        <durability_service>
            <service_cleanup_delay>
                <sec>3600</sec>
                <nanosec>0</nanosec>
            </service_cleanup_delay>
            <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>
            <history_depth>100</history_depth>
            <max_samples>8192</max_samples>
            <max_instances>4196</max_instances>
            <max_samples_per_instance>8192</max_samples_per_instance>
        </durability_service>
        <deadline>
            <period>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </period>
        </deadline>
        <latency_budget>
            <duration>
                <sec>0</sec>
                <nanosec>0</nanosec>
            </duration>
        </latency_budget>
        <liveliness>
            <kind>AUTOMATIC_LIVELINESS_QOS</kind>
            <lease_duration>
                <sec>DURATION_INFINITE_SEC</sec>
                <nanosec>DURATION_INFINITE_NSEC</nanosec>
            </lease_duration>
        </liveliness>
        <reliability>
            <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
            <max_blocking_time>
                <sec>0</sec>
                <nanosec>100000000</nanosec>
                </max_blocking_time>
        </reliability>
        <destination_order>
            <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
        </destination_order>
        <history>
            <kind>KEEP_LAST_HISTORY_QOS</kind>
            <depth>1</depth>
        </history>
           <resource_limits>
            <max_samples>LENGTH_UNLIMITED</max_samples>
            <max_instances>LENGTH_UNLIMITED</max_instances>
            <max_samples_per_instance>LENGTH_UNLIMITED</max_samples_per_instance>
        </resource_limits>
        <transport_priority>
            <value>0</value>
        </transport_priority>
        <lifespan>
```

```
                <duration>
                     <sec>DURATION_INFINITE_SEC</sec>
                     <nanosec>DURATION_INFINITE_NSEC</nanosec>
                </duration>
           </lifespan>
           <ownership>
                <kind>SHARED_OWNERSHIP_QOS</kind>
           </ownership>
      </topic_qos>
</qos_profile>
</dds>
```

6.1.2 Applying QoS Profile

To set the QoS profile for a DDS entity using the Node.js DCPS API and an XML file, the user specifies the qos file URI or file path and the QoS profile name as parameters.

Example

Create a QoS provider and get the respective entity QoS. The example below uses the QoS XML file referred in *QoS Profile*. For a detailed example refer to 'GetSetQoSExample.js' file referred in *GetSetQoSExample* example.

```
const dds = require('vortexdds');
const HelloWorldTopic = require('./HelloWorldTopic.js');

/**
 * Path of the qos xml file
 * The DDS_PersistentQoS.xml file is used for this example
 * It is located in the same directory of this example
* */
const QOS_PATH = 'DDS_PersistentQoS.xml';
const QOS_PROFILE = 'DDS PersistentQosProfile';
const DOMAIN_ID = dds.DDS_DOMAIN_DEFAULT;

main();

async function main(){

  /**
   * Set QoS using QoS XML file
   */

  // create a qos provider using qos xml file
  let qp = new dds.QoSProvider(QOS_PATH, QOS_PROFILE);

  // get participant qos from qos provider and create a participant
  let pqos = qp.getParticipantQos();
  let participant = new dds.Participant(DOMAIN_ID, pqos);

  // get publisher qos from qos provider and create a publisher
  let pubqos = qp.getPublisherQos();
  let publisher = new dds.Publisher(participant, pubqos);

  let subqos = qp.getSubscriberQos('subscriber1');
  let subscriber = new dds.Subscriber(participant, subqos);

  // get topic qos from qos provider and create a topic
  let tqos = qp.getTopicQos();
  let topic = await HelloWorldTopic.create(participant, tqos);

  // get reader qos from qos provider and create a reader
  let rqos = qp.getReaderQos();
```

```
  let reader = new dds.Reader(subscriber, topic, rqos);

  // get writer qos from qos provider and create a writer
  let wqos = qp.getWriterQos();
  let writer = new dds.Writer(publisher, topic, wqos);

  // delete the participant
  participant.delete();
}
```

6.2 Setting QoS Using Node.js DCPS API Classes

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

Example

Below is a code snippet similar to the 'GetSetQoSExample/GetSetQoSExample.js' file. It demonstrates how to specify the QoS settings for a topic, writer and reader using the Node.js DCPS APIs.

```
const dds = require('vortexdds');
const HelloWorldTopic = require('./HelloWorldTopic.js');

main();

async function main(){

  let participant = new dds.Participant();

  // get the default topic qos
  let tqos = dds.QoS.topicDefault();

  // modify the topic qos policies

  // All of the policy variables can be set
  tqos.durabilityService = {
    serviceCleanupDelay: 5000,
    historyKind: dds.HistoryKind.KeepLast,
    historyDepth: 10,
    maxSamples: 10,
    maxInstances: 5,
    maxSamplesPerInstance: -1,
  };

  // Or a subset of the policy variables can be set
  tqos.durabilityService = {
    historyKind: dds.HistoryKind.KeepLast,
  };

  tqos.reliability = dds.ReliabilityKind.reliable;
  tqos.durability = dds.DurabilityKind.Volatile;
  tqos.topicdata = 'Hello world topic';
  tqos.userdata = 'connected to hello world';
  tqos.groupdata = 'hello world group';
  tqos.history = {
    kind: dds.HistoryKind.KeepLast,
    depth: 10,
  };
  tqos.resourceLimits = {
    maxSamples: 10,
    maxInstances: 5,
  };
```

```
tqos.lifespan = 5000;
tqos.deadline = 3000;
tqos.latencyBudget = 4000;
tqos.ownership = dds.OwnershipKind.Exclusive;
tqos.liveliness = {
  kind: dds.LivelinessKind.ManualByTopic,
  leaseDuration: 10,
};
tqos.reliability = {
  kind: dds.ReliabilityKind.Reliable,
  maxBlockingTime: 100,
};
tqos.transportPriority = 5;
tqos.destinationOrder = dds.DestinationOrderKind.BySourceTimestamp;

// create topic with qos
let topic = await HelloWorldTopic.create(participant, tqos);

// get topic qos policies
console.log('\n', '** Topic QoS **');
console.log('Reliability: ', tqos.reliability);
console.log('Durability: ', tqos.durability);
console.log('Topic data: ', tqos.topicdata);
console.log('User data: ', tqos.userdata);
console.log('Group data: ', tqos.groupdata);
console.log('History: ', tqos.history);
console.log('Resource Limits: ', tqos.resourceLimits);
console.log('Lifespan: ', tqos.lifespan);
console.log('Deadline: ', tqos.deadline);
console.log('Latency Budget: ', tqos.latencyBudget);
console.log('Ownership: ', tqos.ownership);
console.log('Liveliness: ', tqos.liveliness);
console.log('Transport Priority: ', tqos.transportPriority);
console.log('Destination Order: ', tqos.destinationOrder);
console.log('Durability Service: ', tqos.durabilityService);

// get the default writer qos
let wqos = dds.QoS.writerDefault();

// modify the writer qos policies
wqos.writerDataLifecycle = false;

// create a writer with qos
let writer = new dds.Writer(publisher, topic, wqos);

// get writer qos policies
console.log('\n', '** Writer QoS **');
console.log('Writer data lifecyle: ', wqos.writerDataLifecycle);

// get the default reader qos
let rqos = dds.QoS.readerDefault();

// modify the reader qos policies
rqos.partition = 'partition1';
rqos.timebasedFilter = 60000;
rqos.readerDataLifecycle = {
  autopurgeNoWriterSamples: 100,
  autopurgeDisposedSamplesDelay: 500,
};

// create a reader with qos
let reader = new dds.Reader(subscriber, topic, rqos);
```

```
  // get reader qos policies
  console.log('\n', '** Reader QoS **');
  console.log('Partition: ', rqos.partition);
  console.log('Time based filter: ', rqos.timebasedFilter);
  console.log('Reader data lifecycle: ', rqos.readerDataLifecycle);

  // delete the participant
  participant.delete();

  // NOTE: Actual reading and writing data  is not demonstrated in this example
}
```

7

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 Node.js DCPS API provides several ways of generating Node.js classes to represent DDS topics.

- over the wire discovery
- dynamic generation of Node.js classes using parameters IDL file and topic name

---

Note:

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

7.1 Over the Wire Discovery

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

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

A code snippet is provided from findTopicExample.js. This example finds a topic registered by another process, and writes a sample to that topic.

Example

```
...
const dds = require('vortexdds');

console.log('Connecting to DDS domain...');
const participant = new dds.Participant();

console.log('Finding topic...');
let topic = participant.findTopic('HelloWorldData_Msg');

console.log('Creating writer and sample data to write...');
let writer = participant.createWriter(topic);
let sample = { userID: 4, message: 7 };

console.log('Writing sample data...');
let status = writer.write(sample);
...
```

7.2 Dynamic Generation of Node.js Topic Classes Using IDL and Name

The Node.js DCPS API supports generation of Node.js topic classes from IDL. This section describes the details of the IDL-Node.js binding.

7.2.1 Dynamic Generation

The Node.js DCPS API provides an asynchronous function that returns a Map of `TypeSupport` objects.

A `TypeSupport` object includes the topic typename, keys and descriptor.

The structure type representation of a topic is created by the TypeSupport object. However, the usage of the structure type is internal to the Node.js DCPS API.

In order to create a topic, a topic name and a `TypeSupport` are passed into the `Participant` createTopicFor function. (qos and listener parameters are optional)

The code snippet below is taken from the 'IoTTopicHelper.js' file referred in *IoTData* example.

```
const dds = require('vortexdds');
const path = require('path');

//asynchronous function to create the topic
module.exports.create = async function(participant) {

  const topicName = 'IoTData';
  const idlName = 'dds_IoTData.idl';
  const idlPath = path.resolve(idlName);

  //wait for dds.getTopicTypeSupportsForIDL to return a map of typeSupports
  let typeSupports = await dds.getTopicTypeSupportsForIDL(idlPath);

  //idl contains 1 topic.
  let typeSupport = typeSupports.get('DDS::IoT::IoTData');

  return participant.createTopicFor(topicName, typeSupport);
};
```

7.2.2 Generated Artifacts

The following table defines the Node.js artifacts generated from IDL concepts:

IDL Concept	Node.js Concept
module	N/A
enum	enum from npm 'enum' package
enum value	enum value
struct	object
field	object property
union	object (IoTValue from dds_IoTData.idl is the only supported union)

Datatype mappings

The following table shows the Node.js equivalents to IDL primitive types:

IDL Type	Node.js Type
boolean	Boolean
char	Number
octet	Number
short	Number
ushort	Number
long	Number
ulong	Number
long long	Number
ulong long	Number
float	Number
double	Number
string	String
wchar	Unsupported
wstring	Unsupported
any	Unsupported
long double	Unsupported

Implementing Arrays and Sequences in Node.js

Both IDL arrays and IDL sequences are mapped to JavaScript arrays.

7.3 Limitations of Node.js Support

The Node.js binding has the following limitations:

- Listeners are not supported
- Only the IoTValue union from dds_IoTData.idl is supported in the beta
- JavaScript does not currently include standard support for 64-bit integer values. 64-bit integers with more than 53 bits of data are represented by String values to avoid loss of precision. If the value will fit inside a JavaScript Number without losing precision, a Number can be used, otherwise use a String. (Refer to *IoTData* example which demonstrates the usage and ranges for the unsigned and signed 64 bit integers within nodejs.)

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