OpenSplice DDS

Version 3.4

Migration Guide

PRISMTECH

OpenSplice DDS

MIGRATION GUIDE

Part Number: OS-MG

Doc Issue 07, 27 May 08

Copyright Notice

Preface

About the Migration Guide

The *Migration Guide* is intended to help users migrate their existing code from OpenSplice DDS *version 2.x* to OpenSplice DDS *version 3.x*.

OpenSplice DDS V3.x is compliant with the OMG's *Data Distribution Service for Real-Time Systems Specification Version 1.2* (DDS V1.2), whereas OpenSplice DDS V2.x is compliant with the OMG's DDS V1.1 specification.

This migration guide is only indended for customers that are currently using *OpenSplice V2.x* and want to migrate to *OpenSplice V3.x*. OpenSplice DDS V2.x and earlier versions progressively introduced extensions and improvements, nontheless their API and features (compliant with DDS V1.1) were not modified and users did not need to migrate their code. However, OpenSplice DDS V3.x contains changes in its API (compliant with DDS V1.2) which means that OpenSplice DDS V2.x-based code must modified in order to be compatible with the OpensSplice V3.x.

The *Migration Guide* covers the code in OpenSplice DDS V3.x that is incompatible with the previous V2.x versions. The Migration Guide describes the incompatibilities and provides realted solutions for the languages which OpenSplice supports.

This guide ***does not*** cover all of the differences which exist between the OpenSplice DDS V3.x and previous versions, only those which are needed for compatibility.

The C language binding is provided with a special *legacy mode* which enables pre-version 3.x code to be used without modification.

Organisation

The Migration Guide is organised as follows:

1. Incompatibilities between Version 3.x and previous versions are listed by category.
2. Solutions for resolving the incompatibilities are given under each language supported by OpenSplice, namely *C*, *C++* and *Java*. The solutions are grouped by categories.

Conventions

The conventions listed below are used to guide and assist the reader in understanding the Migration Guide.

Item of special significance or where caution needs to be taken.

Item contains helpful hint or special information.

WIN Information applies to Windows (e.g. NT, 2000, XP) only.

UNIX Information applies to Unix based systems (e.g. Solaris) only.

C C language specific

C++ C++ language specific

Java Java language specific

Hypertext links are shown as *blue italic underlined.*

On-Line (PDF) versions of this document: Items shown as cross references, e.g. *Contacts* on page iv, are as hypertext links: click on the reference to go to the item.

```
%  Commands or input which the user enters on the
   command line of their computer terminal
```

`Courier` fonts indicate programming code and file names.

Extended code fragments are shown in shaded boxes:

```
NameComponent newName[] = new NameComponent[1];

// set id field to “example” and kind field to an empty string
newName[0] = new NameComponent (“example”, “”);
```

Italics and ***Italic Bold*** indicate new terms, or emphasise an item.

Arial Bold indicate Graphical User Interface (GUI) elements and commands, for example, **File | Save** from a menu.

Step 1: One of several steps required to complete a task.

Contacts

PrismTech can be reached at the following contact points for information and technical support.

Corporate Headquarters

PrismTech Corporation
6 Lincoln Knoll Lane
Suite 100
Burlington, MA
01803
USA

Tel: +1 781 270 1177
Fax: +1 781 238 1700

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900
Fax: +44 (0)191 497 9901

Web:	*http://www.prismtech.com*
General Enquiries:	*info@prismtech.com*

Migration to Version 3.x

CHAPTER

1 Incompatibilities and Solutions

The incompatibilities between OpenSplice DDS's DCPS API version 3.x and previous 2.x versions, along with their solutions when migrating to version 3.x, are given here.

1.1 API Incompatibilities Between Version 3.x and 2.x Versions

1. Name changes

The following table lists the names which have been changed in version 3.x.

Original 2.x Version Name	Name Used in Version 3.x
DURATION_INFINITY	DURATION_INFINITE
DURATION_INFINITY_SEC	DURATION_INFINITE_SEC
DURATION_INFINITY_NSEC	DURATION_INFINITE_NSEC
PublicationMatchStatus	PublicationMatchedStatus
PUBLICATION_MATCH	PUBLICATION_MATCHED
get_publication_match_status()	get_ publication_matched_status()
on_publication_match()	on_publication_matched()
SubscriptionMatchStatus	SubscriptionMatchedStatus
SUBSCRIPTION_MATCH	SUBSCRIPTION_MATCHED
get_subscription_match_status	get_subscription_matched_status()
on_subscription_match()	on_subscription_matched()
QueryCondition.set_query_arguments()	get_query_parameters()
QueryCondition.set_query_arguments()	set_query_parameters()

2. Factory methods containing an additional *mask* parameter

```
DomainParticipantFactory.create_participant()
DomainParticipant.create_publisher()
DomainParticipant.create_subscriber()
Domainparticipant.create_topic()
Publisher.create_datareader()
Subscriber.create_datawriter()
```

3. **Method arguments which have been changed from *out* to *inout***

   `WaitSet.wait()`

   `WaitSet.get_conditions()`

4. **Methods which use *inout* parameters instead of a return value**

   `get…status() - methods`

   `ContentFilteredTopic.get_expression_parameters()`

   `MultiTopic.get_expression_parameters()`

5. ***WaitSet.wait()* can now return *RETCODE_TIME_OUT***

1.2 Migration Solutions

1.2.1 C Language Binding

Legacy Mode

A legacy mode is provided for the C language binding which enables existing code to be used, without changes, with OpenSplice's version 3.x API.

Legacy mode can be used as a temporary solution which can time for developers to progressively migrate their existing code base.

This legacy mode should be considered deprecated: it will ***not*** be available in future versions. Also, the new features which are supported in version 3.x will not be available when using this mode.

Legacy mode is activated by defining the ***OSPLV2_LEGACY_API*** compiler flag. This can be done by adding the following line to your makefile:

```
Cc_flags += -DOSPLV2_LEGACY_API
```

An OpenSpliceV2 compatible API is provided when this flag is used, noting that this API is not fully compliant with the V3.x DCPS API specification.

Name Changes

Perform a *search and replace* with your editor through all source files.

Factory methods have an added *mask* parameter

The V3.x API supports the setting of the *listener mask* when an entity is created. All ***_create()*** methods now have a parameter for this setting the listener mask.

Existing code can be migrated to V3.x by adding the parameter to all `_create()` methods using the following values:

- When a listener is ***used***, set the mask parameter value to ***DDS_ANY_STATUS***. For example:

```
participant =
   DDS_DomainParticipantFactory_create_participant(
      factory, NULL, DDS_PARTICIPANT_QOS_DEFAULT,
      participantListener, DDS_ANY_STATUS);
```

- When a listener is ***not used***, set the mask parameter value to ***0*** (zero). For example:

```
participant =
   DDS_DomainParticipantFactory_create_participant (
      factory, NULL, DDS_PARTICIPANT_QOS_DEFAULT, NULL, 0);
```

Method arguments changed from *out* to *inout*

Certain parameter have been changed from ***out*** to ***inout*** for several methods (see list under Section *1.1*). This significantly changes how these parameters are handled whereby now a *pointer* to a properly allocated and initialized variable should be supplied instead of the *address of a pointer.*

Example

```
DDS_ConditionSeq Conditions = {0,0,NULL,FALSE};
result = DDS_WaitSet_wait(w,&Conditions,&DURATION_ONE);
```

Methods using *inout* parameters instead of a return value

Several methods are now returning their results in an ***inout*** parameter instead of a return value (refer to Section *1.1*). A pointer to a properly allocated and initialized variable should now be used to retrieve values for the affected methods.

Example

```
DDS_RequestedIncompatibleQosStatus requestedStatus;
memset(&requestedStatus, 0, sizeof(requestedStatus));
DDS_DataReader_get_requested_incompatible_qos_status(
   reader, &requestedStatus);
```

i This example demonstrates a status `struct` containing a *sequence*, noting that sequences must be correctly initialized.

DDS_WaitSet_wait() can now return *RETCODE_TIME_OUT*

A *waitset* in DCPS API versions prior to V3.x returns *RETCODE_OK* and an empty condition list when it fails to trigger within a given timeout time. A waitset in version 3.x will return a *RETCODE_TIME_OUT* when it fails to trigger within a timeout.

Code that checks the return value and only checks or accepts *RETCODE_OK* should be changed to check or accept *RETCODE_TIME_OUT*.

1.2.2 C+ Language Binding

Name Changes

Perform a *search and replace* with your editor through all source files.

Factory methods have an added *mask* parameter

The V3.x API supports the setting of the *listener mask* when an entity is created. All ***_create()*** methods now have a parameter for this setting the listener mask.

Existing code can be migrated to V3.x by adding the parameter to all `_create()` methods using the following values:

- When a listener is ***used***, set the mask parameter value to ***DDS::ANY_STATUS***. For example:

```
participant = factory->create_participant(myDomain,
   PARTICIPANT_QOS_DEFAULT, participantListener,
      DDS::ANY_STATUS);
```

- When a listener is ***not used***, set the mask parameter value to ***0*** (zero). For example:

```
participant = factory->create_participant(myDomain,
   PARTICIPANT_QOS_DEFAULT, NULL, 0);
```

Method arguments changed from *out* to *inout*

This does not affect existing code in C++ and accordingly no changes are needed.

Methods use *inout* parameters instead of a return value

Several methods are now returning their results in an ***inout*** parameter instead of a return value (see list under Section *1.1*).

An object should be passed as a parameter to hold the method's result instead of assigning the method's return value.

Example

```
OfferedIncompatibleQosStatus info;
    writer->get_offered_incompatible_qos_status(info);
```

WaitSet.wait() can now return *RETCODE_TIME_OUT*

A *waitset* in DCPS API versions prior to V3.x returns *DDS::RETCODE_OK* and an empty condition list when the waitset fails to trigger within a given timeout time. A waitset in version 3.x will return a *RETCODE_TIME_OUT* when it fails to trigger within a timeout.

Code that checks the return value and only checks or accepts *RETCODE_OK* should be changed to check or accept *DDS::RETCODE_TIME_OUT*.

1.2.3 Java Language Binding

Name Changes

Perform a *search and replace* with your editor through all source files.

Factory methods have an added *mask* parameter

The V3.x API supports the setting of the *listener mask* when an entity is created. All ***_create()*** methods now have a parameter for this setting the listener mask.

Existing code can be migrated to V3.x by adding the parameter to all `_create()` methods using the following values:

- When a listener is ***used***, set the mask parameter value to ***DDS.ANY_STATUS***. For example:

```
participant = factory.create_participant (myDomain,
   PARTICIPANT_QOS_DEFAULT.value,
   participantListener, DDS.ANY_STATUS.value);
```

- When a listener is ***not used***, set the mask parameter value to ***0*** (zero). For example:

```
participant= factory.create_participant (myDomain,
   PARTICIPANT_QOS_DEFAULT.value, null, 0);
```

Method arguments changed from *out* to *inout*

This does not affect existing code in Java and accordingly no changes are needed.

Methods use *inout* parameters instead of a return value

Several methods are now returning their results in an ***inout*** parameter instead of a return value (see list under Section *1.1*).

An object should be passed as a parameter to hold the method's result instead of assigning the method's return value.

Example

```
OfferedIncompatibleQosStatusHolder infoHolder;
OfferedIncompatibleQosStatus info;
int retcode;

infoHolder = new OfferedIncompatibleQosStatusHolder();
retcode =
   writer.get_offered_incompatible_qos_status(infoHolder);
info = infoHolder.value;
```

WaitSet.wait() can now return *RETCODE_TIME_OUT*

A *waitset* in DCPS API versions prior to V3.x returns *DDS.RETCODE_OK* and an empty condition list when the waitset fails to trigger within a given timeout time. A waitset in version 3.x will return a *RETCODE_TIME_OUT* when it fails to trigger within a timeout.

Code that checks the return value and only checks or accepts *RETCODE_OK* should be changed to check or accept *DDS.RETCODE_TIME_OUT*.