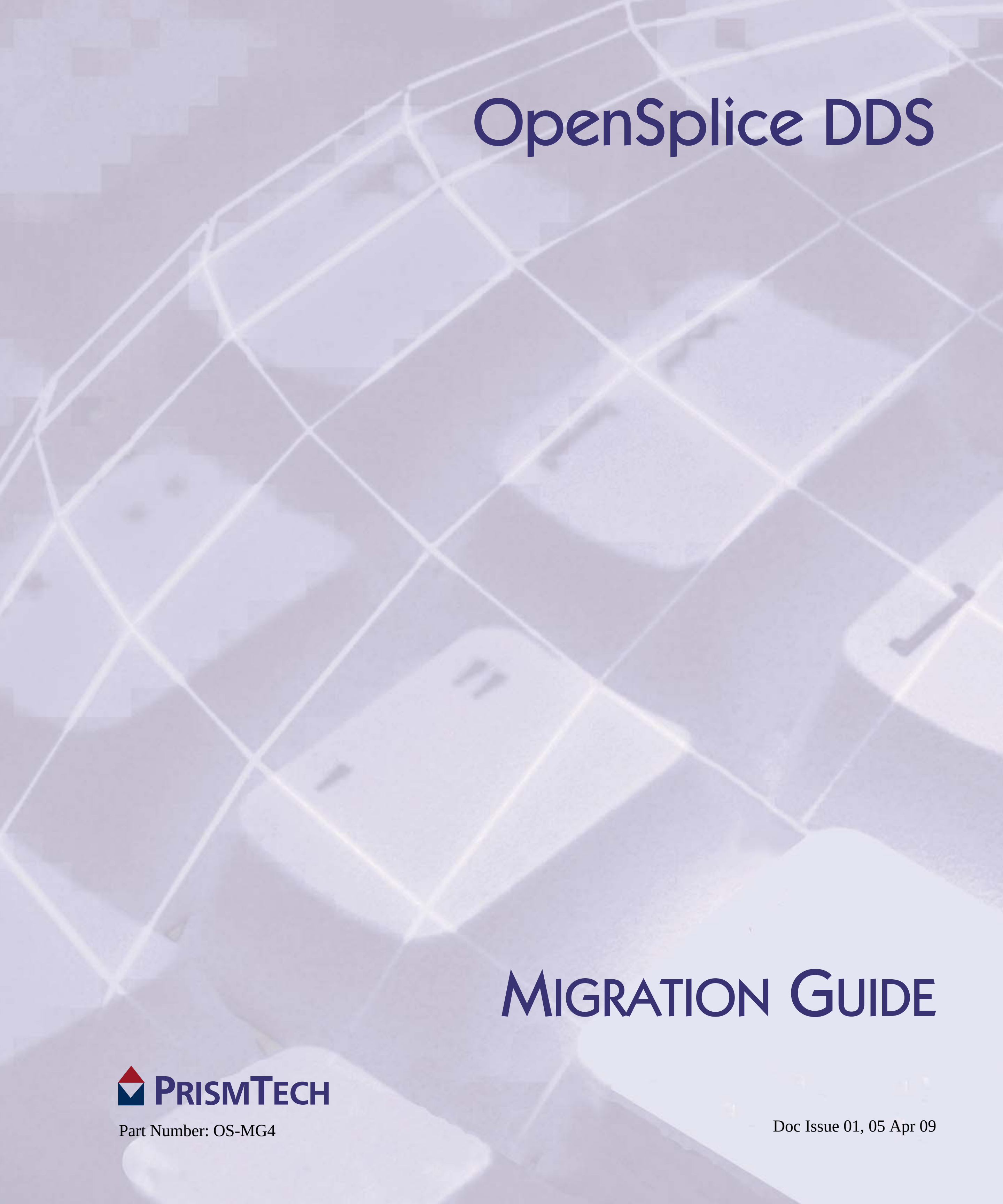
OpenSplice DDS

Version 4.1

Migration Guide

PrismTech



OpenSplice DDS

Migration Guide

PrismTech

Part Number: OS-MG4

Doc Issue 01, 05 Apr 09

Preface

About this Migration Guide

This *Migration Guide* is intended to help users migrate their existing code from OpenSplice DDS *version 3.x* to OpenSplice DDS *version 4.x*.

⚠ This guide is **only** intended for customers who are currently using *OpenSplice V3.x* and who want to migrate to *OpenSplice V4.x*. Migration from V2.x to V3.x is covered by a separate Migration Guide.

⚠ This guide ***does not*** cover all of the differences which exist between the OpenSplice DDS V4.x and previous versions, only those which are needed for compatibility.

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

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.
i	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.

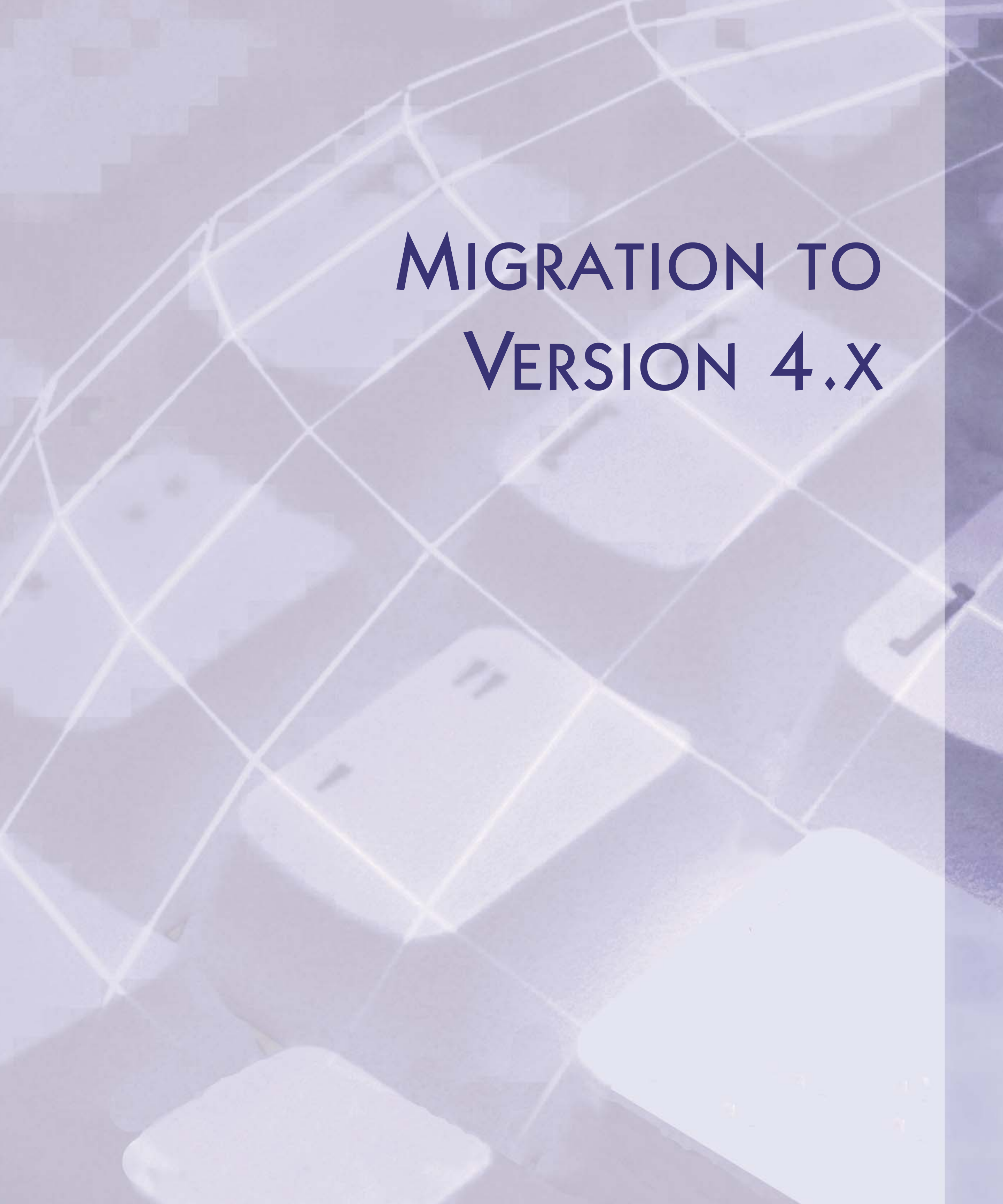
Step 1: One of several steps required to complete a task.

Contacts

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

Corporate Headquarters	**European Head Office**
PrismTech Corporation	PrismTech Limited
6 Lincoln Knoll Lane	PrismTech House
Suite 100	5th Avenue Business Park
Burlington, MA	Gateshead
01803	NE11 0NG
USA	UK
Tel: +1 781 270 1177	Tel: +44 (0)191 497 9900
Fax: +1 781 238 1700	Fax: +44 (0)191 497 9901

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



Migration to Version 4.x

CHAPTER

1 Incompatibilities and Solutions

This guide is intended to help users of OpenSplice DDS Version 3 with the migration of their existing code-base to the new version of OpenSplice DDS (V4.1).

Not all changes in the product are described here, as this document is focused on those changes that cause incompatibilities between existing code and the new version of OpenSplice.

A complete list of all changes in the product can be found in the release notes.

1.1 Data Structure Changes

SampleRejectedStatusKind

The values of `SampleRejectedStatusKind` have changed. This datatype is used for `sampleRejectedStatus.last_reason`.

V3:

```
enum SampleRejectedStatusKind {
      DDS_REJECTED_BY_INSTANCE_LIMIT,
      DDS_REJECTED_BY_TOPIC_LIMIT
 };
```

V4:

```
enum SampleRejectedStatusKind {
      DDS_NOT REJECTED,
      DDS_REJECTED_BY_INSTANCE_LIMIT,
      DDS_REJECTED_BY_SAMPLES_LIMIT,
      DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT
 };
```

In the old situation there was no proper value when a sample was not rejected. Now the values indicate exactly which of the resource limits caused the sample to be rejected.

If the `sampleRejectedStatus.last_reason` field is examined in your application, the code should be adapted to these new values.

OwnershipQosPolicy

The `OwnershipQosPolicy` is only part of the `TopicQos` in V3. This implies that DataWriters and DataReaders automatically behave as specified in the `TopicQos`. In V4, the interface and according behavior are compliant with OMG/DDS specification V1.2. This means that the `OwnershipQosPolicy` now is also part of the `DataWriterQos` and the `DataReaderQos`. Besides that, the `OwnershipQosPolicy` is now also part of the `PublicationInfo` and `SubscriptionInfo` built-in topics.

The `OwnershipQosPolicy` is `RxO` and the kind is, by default, `SHARED`. If existing applications use `COPY_FROM_TOPIC_QOS` the application will not have problems, otherwise it is possible that the writer will not be connected with the reader, because they specify different `OwnershipQosPolicy` values.

V3:

```
struct DataWriterQos {
     DurabilityQosPolicy durability;
     DeadlineQosPolicy deadline;
     LatencyBudgetQosPolicy latency_budget;
     LivelinessQosPolicy liveliness;
     ReliabilityQosPolicy reliability;
     DestinationOrderQosPolicy destination_order;
     HistoryQosPolicy history;
     ResourceLimitsQosPolicy resource_limits;
     TransportPriorityQosPolicy transport_priority;
     LifespanQosPolicy lifespan;
     UserDataQosPolicy user_data;
     OwnershipStrengthQosPolicy ownership_strength;
     WriterDataLifecycleQosPolicy writer_data_lifecycle;
};

struct DataReaderQos {
     DurabilityQosPolicy durability;
     DeadlineQosPolicy deadline;
     LatencyBudgetQosPolicy latency_budget;
     LivelinessQosPolicy liveliness;
     ReliabilityQosPolicy reliability;
     DestinationOrderQosPolicy destination_order;
     HistoryQosPolicy history;
     ResourceLimitsQosPolicy resource_limits;
     UserDataQosPolicy user_data;
     TimeBasedFilterQosPolicy time_based_filter;
     ReaderDataLifecycleQosPolicy reader_data_lifecycle;
     SubscriptionKeyQosPolicy subscription_keys;
};

struct PublicationBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    LifespanQosPolicy lifespan;
    UserDataQosPolicy user_data;
    OwnershipStrengthQosPolicy ownership_strength;
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    TopicDataQosPolicy topic_data;
    GroupDataQosPolicy group_data;
};

struct SubscriptionBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    UserDataQosPolicy user_data;
    TimeBasedFilterQosPolicy time_based_filter;
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    TopicDataQosPolicy topic_data;
    GroupDataQosPolicy group_data;
};
```

V4:

```
struct DataWriterQos {
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    HistoryQosPolicy history;
    ResourceLimitsQosPolicy resource_limits;
    TransportPriorityQosPolicy transport_priority;
    LifespanQosPolicy lifespan;
    UserDataQosPolicy user_data;
    OwnershipQosPolicy ownership;
    OwnershipStrengthQosPolicy ownership_strength;
    WriterDataLifecycleQosPolicy writer_data_lifecycle;
};

struct DataReaderQos {
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    HistoryQosPolicy history;
    ResourceLimitsQosPolicy resource_limits;
    UserDataQosPolicy user_data;
    OwnershipQosPolicy ownership;
    TimeBasedFilterQosPolicy time_based_filter;
    ReaderDataLifecycleQosPolicy reader_data_lifecycle;
    SubscriptionKeyQosPolicy subscription_keys;
};

struct PublicationBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    LifespanQosPolicy lifespan;
    UserDataQosPolicy user_data;
    OwnershipQosPolicy ownership;
    OwnershipStrengthQosPolicy ownership_strength;
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    TopicDataQosPolicy topic_data;
    GroupDataQosPolicy group_data;
};

struct SubscriptionBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
```

```
        ReliabilityQosPolicy reliability;
        OwnershipQosPolicy ownership;
        DestinationOrderQosPolicy destination_order;
        UserDataQosPolicy user_data;
        TimeBasedFilterQosPolicy time_based_filter;
        PresentationQosPolicy presentation;
        PartitionQosPolicy partition;
        TopicDataQosPolicy topic_data;
        GroupDataQosPolicy group_data;
    };
```

Subscrtiption and publication match status

V3:

```
/* legacy*/
typedef struct DDS_PublicationMatchStatus_s
DDS_PublicationMatchStatus;
struct DDS_PublicationMatchStatus_s {
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_subscription_handle;
};
typedef struct DDS_SubscriptionMatchStatus_s
DDS_SubscriptionMatchStatus;
struct DDS_SubscriptionMatchStatus_s {
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_publication_handle;
};

struct DDS_PublisherListener {
    void *listener_data;
    DDS_PublisherListener_OfferedDeadlineMissedListener
on_offered_deadline_missed;
    DDS_PublisherListener_OfferedIncompatibleQosListener
on_offered_incompatible_qos;
    DDS_PublisherListener_LivelinessLostListener on_liveliness_lost;
    DDS_PublisherListener_PublicationMatchListener
on_publication_match; /*legacy*/
    DDS_PublisherListener_PublicationMatchedListener
on_publication_matched;
};

struct DDS_SubscriberListener {
    void *listener_data;
    DDS_SubscriberListener_RequestedDeadlineMissedListener
on_requested_deadline_missed;
    DDS_SubscriberListener_RequestedIncompatibleQosListener
on_requested_incompatible_qos;
```

```
    DDS_SubscriberListener_SampleRejectedListener on_sample_rejected;
    DDS_SubscriberListener_LivelinessChangedListener
on_liveliness_changed;
    DDS_SubscriberListener_DataAvailableListener on_data_available;
    DDS_SubscriberListener_SubscriptionMatchListener
on_subscription_match; /*legacy*/
    DDS_SubscriberListener_SubscriptionMatchedListener
on_subscription_matched;
    DDS_SubscriberListener_SampleLostListener on_sample_lost;
    DDS_SubscriberListener_DataOnReadersListener on_data_on_readers;
};

struct DDS_DomainParticipantListener {
    void *listener_data;
    DDS_DomainParticipantListener_InconsistentTopicListener
on_inconsistent_topic;
    DDS_DomainParticipantListener_OfferedDeadlineMissedListener
on_offered_deadline_missed;
    DDS_DomainParticipantListener_OfferedIncompatibleQosListener
on_offered_incompatible_qos;
    DDS_DomainParticipantListener_LivelinessLostListener
on_liveliness_lost;
    DDS_DomainParticipantListener_PublicationMatchListener
on_publication_match; /* legacy */
    DDS_DomainParticipantListener_PublicationMatchedListener
on_publication_matched;
    DDS_DomainParticipantListener_RequestedDeadlineMissedListener
on_requested_deadline_missed;
    DDS_DomainParticipantListener_RequestedIncompatibleQosListener
on_requested_incompatible_qos;
    DDS_DomainParticipantListener_SampleRejectedListener
on_sample_rejected;
    DDS_DomainParticipantListener_LivelinessChangedListener
on_liveliness_changed;
    DDS_DomainParticipantListener_DataAvailableListener
on_data_available;
    DDS_DomainParticipantListener_SubscriptionMatchListener
on_subscription_match;  /* legacy */
    DDS_DomainParticipantListener_SubscriptionMatchedListener
on_subscription_matched;
    DDS_DomainParticipantListener_SampleLostListener on_sample_lost;
    DDS_DomainParticipantListener_DataOnReadersListener
on_data_on_readers;
};
```

V4:

```
struct DDS_PublisherListener {
    void *listener_data;
    DDS_PublisherListener_OfferedDeadlineMissedListener
on_offered_deadline_missed;
    DDS_PublisherListener_OfferedIncompatibleQosListener
on_offered_incompatible_qos;
    DDS_PublisherListener_LivelinessLostListener on_liveliness_lost;
    DDS_PublisherListener_PublicationMatchedListener
on_publication_matched;
};

struct DDS_SubscriberListener {
    void *listener_data;
    DDS_SubscriberListener_RequestedDeadlineMissedListener
on_requested_deadline_missed;
    DDS_SubscriberListener_RequestedIncompatibleQosListener
on_requested_incompatible_qos;
    DDS_SubscriberListener_SampleRejectedListener on_sample_rejected;
    DDS_SubscriberListener_LivelinessChangedListener
on_liveliness_changed;
    DDS_SubscriberListener_DataAvailableListener on_data_available;
    DDS_SubscriberListener_SubscriptionMatchedListener
on_subscription_matched;
    DDS_SubscriberListener_SampleLostListener on_sample_lost;
    DDS_SubscriberListener_DataOnReadersListener on_data_on_readers;
};

struct DDS_DomainParticipantListener {
    void *listener_data;
    DDS_DomainParticipantListener_InconsistentTopicListener
on_inconsistent_topic;
    DDS_DomainParticipantListener_OfferedDeadlineMissedListener
on_offered_deadline_missed;
    DDS_DomainParticipantListener_OfferedIncompatibleQosListener
on_offered_incompatible_qos;
    DDS_DomainParticipantListener_LivelinessLostListener
on_liveliness_lost;
    DDS_DomainParticipantListener_PublicationMatchedListener
on_publication_matched;
    DDS_DomainParticipantListener_RequestedDeadlineMissedListener
on_requested_deadline_missed;
    DDS_DomainParticipantListener_RequestedIncompatibleQosListener
on_requested_incompatible_qos;
    DDS_DomainParticipantListener_SampleRejectedListener
on_sample_rejected;
    DDS_DomainParticipantListener_LivelinessChangedListener
on_liveliness_changed;
    DDS_DomainParticipantListener_DataAvailableListener
on_data_available;
    DDS_DomainParticipantListener_SubscriptionMatchedListener
on_subscription_matched;
    DDS_DomainParticipantListener_SampleLostListener on_sample_lost;
    DDS_DomainParticipantListener_DataOnReadersListener
on_data_on_readers;
};
```

Duration definition

The following V2 compatible duration definitions, which were available in V3 with and without the `OSPLV2_LEGACY_API` compiler flag, have been removed and are no longer supported:

```
DDS_DURATION_INFINITY_SEC
DDS_DURATION_INFINITY_NSEC
DDS_DURATION_INFINITY
```

1.2 Return code changes

Some calls will return a different return value under the same circumstances, compared with previous versions. If an existing application explicitly checks these return values, the code should be adapted to correctly handle the new values.

*.write()

All `write..()` calls will be able to return `RETCODE_BAD_PARAMETER` if the associated data type contains bounded elements (*e.g.*, bounded string or bounded sequence) and the supplied data violates these bounds.

In V3 these boundary violations were only checked in the *Java* language binding, and would throw an exception. These exceptions will no longer be thrown, but an error code is returned instead.

For *C* and *C++* the bounds checking must be enabled by defining the `OSPL_BOUNDS_CHECK` macro.

```
#define OSPL_BOUNDS_CHECK
```

Without this definition, bound checking is not enabled for *C* and *C++*.

Waitset.detach_condition()

If the condition parameter for this call is not attached to the Waitset, than this call will now return `RETCODE_PRECONDITION_NOT_MET` instead of `BAD_PARAMETER`.

*.delete_contained_entities()

If any of the contained entities still have outstanding loans, this call will now return `RETCODE_PRECONDITION_NOT_MET`. In this case the contained entities that still have outstanding loans are not deleted.

*.set_qos() /set_default_qos() (PresentationQos)

When unsupported values for the `PresentationQos` are supplied, these calls will return `RETCODE_UNSUPPORTED`. In V3 these values were simply ignored and the call would return `RETCODE_OK`.

1.3 Behavioral Changes

WRITER_DATA_LIFECYCLE QoS policy

The `autodispose_unregistered_instances` attribute of the `WRITER_DATA_LIFECYCLE` QoS policy of the `DataWriterQoS` specifies whether an instance should be automatically disposed at the time it is unregistered.

In V3 this behavior is only achieved when an instance is implicitly unregistered (for instance, by the deletion of the DataWriter) and not when the instance is explicitly unregistered by calling `unregister_instance()`.

In V4 the instance is also disposed when explicitly unregistering the instance. The resulting state of the instance in all connected DataReaders will become `NOT_ALIVE_DISPOSED` after unregistering.

DEADLINE QoS policy

On the publishing side the DEADLINE QoS policy establishes a contract that the application must meet. On the subscribing side the policy establishes a minimum requirement for the remote publishers that are expected to supply the data values.

In V3 the DataWriter notifies itself when it detects a violation of the deadline contract and automatically unregisters the instance for which the contract is violated in case the OWNERSHIP QoS policy is set to `EXCLUSIVE`. The DataReader notifies itself when it detects a violation of the contract. This implementation is *not* according to the OMG DDS specification. The ownership of an instance should be determined by each DataReader itself. It is a DataReader-specific view on the instance.

In V4 the DataWriter only notifies itself when it detects a violation of the deadline contract and no longer unregisters the instance. The DataReader also notifies itself and additionally resets the ownership of the instance in case the DataReader OWNERSHIP QoS policy is set to `EXCLUSIVE`.

V2 Compatibility Mode

In V3 the old syntax of several calls would still be accepted when the `OSPLV2_LEGACY_API` compiler flag was set (please refer to *OpenSplice V2 to V3 Migration Guide*).

This compatibility mode is no longer available in V4. It is therefore necessary to use the new syntax for these calls. Use the solutions from *OpenSplice V2 to V3 Migration Guide* to adapt your applications accordingly, if you still use the compatibility mode.

Thread-priorities

In V3 a thread priority is set to the average value of the maximum and minimum allowed for the platform. In V4 the thread priority is set to the *nearest* valid value in such a case. This will result in *either* the minimum or the maximum value.

DataWriter bounds checking

The IDL pre-processor generates copy routines to copy application data to shared memory. Until now, the copy routines did not validate the contents of the data with respect to sequence and string bounds. V4.1 copy routines do check bounds of supplied application data. All DDS DataWriter functions that have application data as (one of) the input parameter now return `BAD_PARAMETER` in case invalid application data is provided.

In V3 these boundary violations were only checked in the *Java* language binding, and would throw an exception. These exceptions will no longer be thrown

For *C* and *C++* this bounds checking must be enabled by defining a `OSPL_BOUNDS_CHECK` macro.

```
#define OSPL_BOUNDS_CHECK
```

Without this definition, bound checking is not enabled for *C* and *C++*.

1.4 DLRL profile changes

Change in liveliness behavior.

OpenSplice V3 liveliness behavior:

When an Object in DLRL no longer has any writers it enters a final lifecycle state internally. It can only exit this state if a new sample for that instance arrives. When this occurs the DLRL destroys the ‘old’ Object and creates a new Object to signal that this has taken place. As a result the user will see an Object in the `created_objects` list and an Object in the `deleted_object` list that represents the same instance.

This is rather confusing when used in applications as it means an Object can be destroyed in the Cache without ever receiving a dispose event. Basically the `no_writers` situation is treated as a ‘postponed’ dispose event, to be processed when a new sample for that instance arrives! It makes management of such scenarios at application level rather tedious.

New behavior for liveliness in OpenSplice V4:

The DCPS concepts of `NOT_ALIVE_DISPOSED` and `NOT_ALIVE_NO_WRITERS` will now be treated differently on DLRL level. Only an explicit dispose leads to Objects being destroyed and changes from `NOT_ALIVE_NO_WRITERS` back to `ALIVE` and vice versa only lead to Object modifications (if a new sample is available of course).

It is important to realize that DLRL objects may still be contained within the `created_objects` and `deleted_objects` lists within the same update round, if for example a dispose event and a new creation is processed in one update round.

OSPLDCG: sacpp generates is_xxx_modified functions for simple types with wrong parameter, should not have no parameter at all

The DCG generates the `is_xxx_modified` operations wrongly for simple types for the SACPP language binding. For example for an idl attribute of a valuetype `string message`, it generates:

```
DDS::Boolean
is_message_modified(
    DDS::ObjectScope scope);
```

which should in fact be:

```
DDS::Boolean
is_message_modified(
    );
```

Application code will need to be adapted and then recompiled whereever such operations were used to reflect the removal of the parameter.

DLRL error log file renamed

The error log generated by the DLRL component is renamed from `splice-dds-dlrl-error.log` to `ospl-dlrl-error.log`.

OSPLDCG: The DCG will no longer rename generated implementation classes

For DLRL it is possible to specify custom operations onto objects. By default the OSPLDCG tool will generate an implementation class containing dummy implementations for any specified operations, so that the generated code always compiles for a good out-of-the-box experience.

In subsequent runs using OpenSplice V3 however the DCG will not overwrite the generated implementation class, but rename it to a `.old` file, to prevent overwriting any manual edits done by the user (*e.g.*, adding specific code to the implementation class operations).

This renaming feature has been disabled in OpenSplice V4, so that the DCG will now always overwrite the generated implementation class. The preferred and only supported way to add implementation code for the custom operations specified onto objects is to use the `implClass` and `implPath` attributes within the XML file to specify a pre-existing file containing the implementation class with the implemented operations. This feature is fully described within the *DLRL Code Generator Guide*; refer to the section regarding adding local operations to DLRL valuetypes.