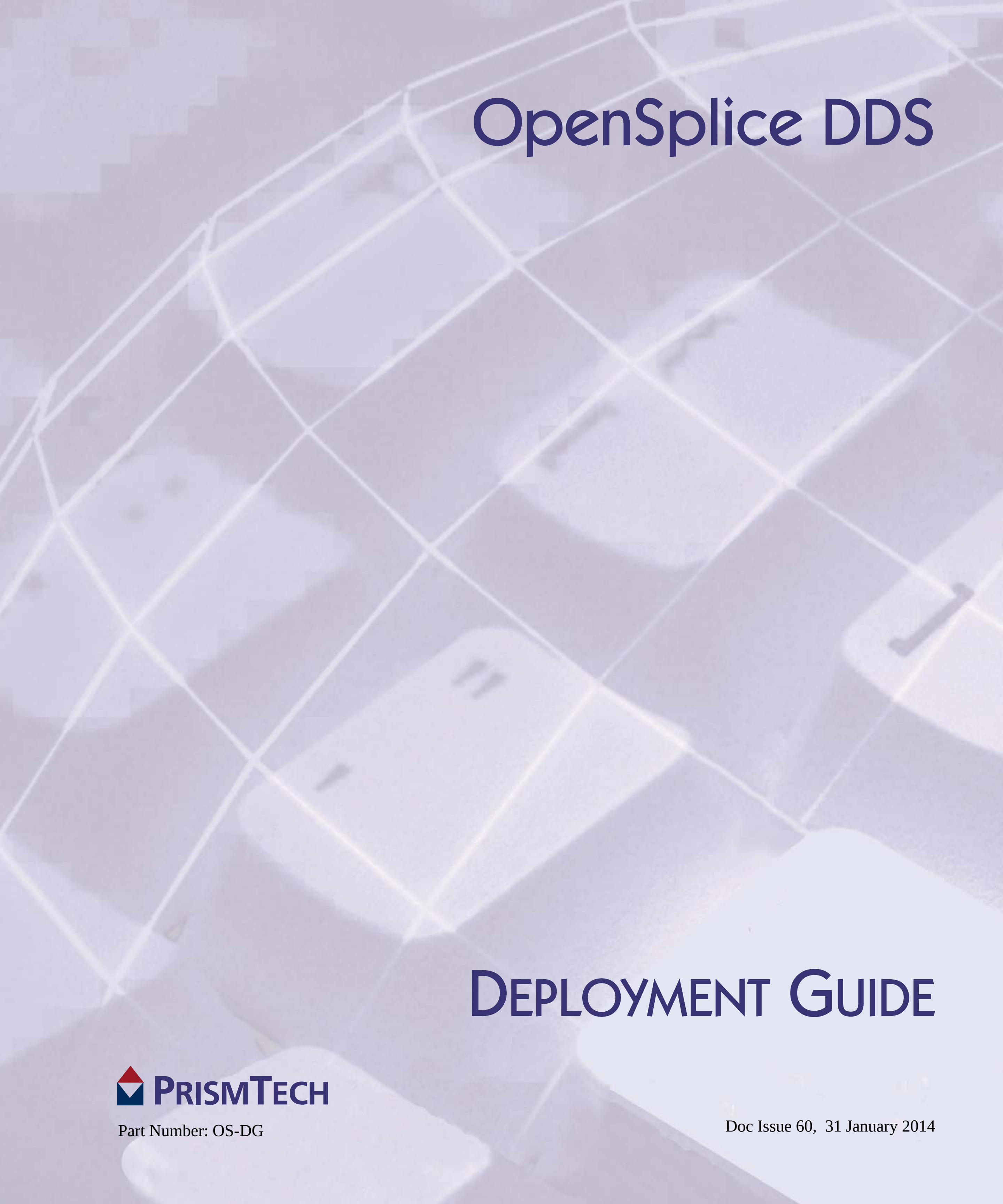
OpenSplice DDS

Version 6.x

Deployment Guide

PrismTech



OpenSplice DDS

Deployment Guide

PrismTech

Part Number: OS-DG

Doc Issue 60, 31 January 2014

Copyright Notice

Contents

Table of Contents

PrismTech

PrismTech

Preface

About the Deployment Guide

The OpenSplice DDS *Deployment Guide* is intended to provide a complete reference on how to configure the OpenSplice service and tune it as required.

The *Deployment Guide* is included with the OpenSplice DDS *Documentation Set*. The *Deployment Guide* is intended to be used after reading and following the instructions in the OpenSplice *Getting Started. Guide*.

Intended Audience

The *Deployment Guide* is intended to be used by anyone who wishes to use and configure the *OpenSplice DDS* service.

Organisation

Chapter 1, *OpenSplice DDS Overview*, gives a general description of the OpenSplice architecture.

Chapter 2, *The OpenSplice DDS Services* describes how OpenSplice provides integration of real-time DDS and the non-/near-real-time enterprise DBMS domains.

Chapter 3, *Tools*, introduces the OpenSplice system management tools.

Chapter 4, *Service Configuration*. describes how to configure the OpenSplice daemons.

Conventions

The conventions listed below are used to guide and assist the reader in understanding the Deployment 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.* XP, 2003, Windows 7) only.
UNIX	Information applies to Unix based systems (*e.g.* Solaris) only.
C	C language specific.
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 xviii) act 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*** are used to indicate new terms, or emphasise an item.

Sans-serif and **Sans-serif Bold** are used to indicate elements of a Graphical User Interface (GUI) or Integrated Development Environment (IDE), such as a Properties tab, and sequences of actions, such as selecting **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.

USA Corporate Headquarters	**European Head Office**
PrismTech Corporation 400 TradeCenter Suite 5900 Woburn, MA 01801 USA	PrismTech Limited PrismTech House 5th Avenue Business Park Gateshead NE11 0NG UK
Tel: +1 781 569 5819	Tel: +44 (0)191 497 9900 Fax: +44 (0)191 497 9901

Web:	*http://www.prismtech.com*	
Technical questions:	*crc@prismtech.com*	(Customer Response Center)
Sales enquiries:	*sales@prismtech.com*	

DEPLOYING OPENSPLICE DDS

CHAPTER

1 OpenSplice DDS Overview

This chapter explains the OpenSplice DDS middleware from a configuration perspective. It shows the different components running on a single node and briefly explains the role of each entity. Furthermore, it defines a reference system that will be used throughout the rest of the document as an example.

1.1 The OpenSplice DDS Architecture

OpenSplice DDS is highly configurable, even allowing the architectural structure of the DDS middleware to be chosen by the user at deployment time. OpenSplice DDS can be configured to run using a so-called ‘federated’ **shared memory** architecture, where both the DDS related administration (including the optional pluggable services) and DDS applications interface directly with shared memory. Alternatively, OpenSplice DDS also supports a so-called ‘standalone’ **single process** architecture, where one or more DDS applications, together with the OpenSplice administration and services, can all be grouped into a single operating system process. Both deployment modes support a configurable and extensible set of services, providing functionality such as:

- *networking* - providing QoS-driven real-time networking based on multiple reliable multicast ‘channels’
- *durability* - providing fault-tolerant storage for both real-time state data as well as persistent settings
- *remote control and monitoring SOAP service* - providing remote web-based access using the SOAP protocol from various OpenSplice tools
- *dbms service* - providing a connection between the real-time and the enterprise domain by bridging data from DDS to DBMS and *vice versa*

The OpenSplice DDS middleware can be easily configured, on the fly, using its pluggable service architecture: the services that are needed can be specified together with their configuration for the particular application domain, including networking parameters, and durability levels for example).

There are advantages to both the single process and shared memory deployment architectures, so the most appropriate deployment choice depends on the user’s exact requirements and DDS scenario.

1.1.1 Single Process architecture

This deployment allows the DDS applications and OpenSplice administration to be contained together within one single operating system process. This ‘standalone’ single process deployment option is most useful in environments where shared memory is unavailable or undesirable. As dynamic heap memory is utilized in the single process deployment environment, there is no need to pre-configure a shared memory segment which in some use cases is also seen as an advantage of this deployment option.

Each DDS application on a processing node is implemented as an individual, self-contained standalone operating system process (*i.e.* all of the DDS administration and necessary services have been linked into the application process). This is known as a **single process application**. Communication between multiple single process applications co-located on the same machine node is done via the (loop-back) network, since there is no memory shared between them.

An extension to the single process architecture is the option to co-locate multiple DDS applications into a single process. This can be done be creating application libraries rather than application executables that can be ‘linked’ into the single process in a similar way to how the DDS middleware services are linked into the single process. This is known as a **single process application cluster**. Communication between clustered applications (that together form a single process) can still benefit from using the process’s heap memory, which typically is an order of magnitude faster than using a network, yet the lifecycle of these clustered applications will be tightly coupled.

The Single Process deployment is the default deployment architecture provided within OpenSplice and allows for easy deployment with minimal configuration required for a running DDS system.

Figure 1 shows an overview of the single process architecture of OpenSplice DDS.

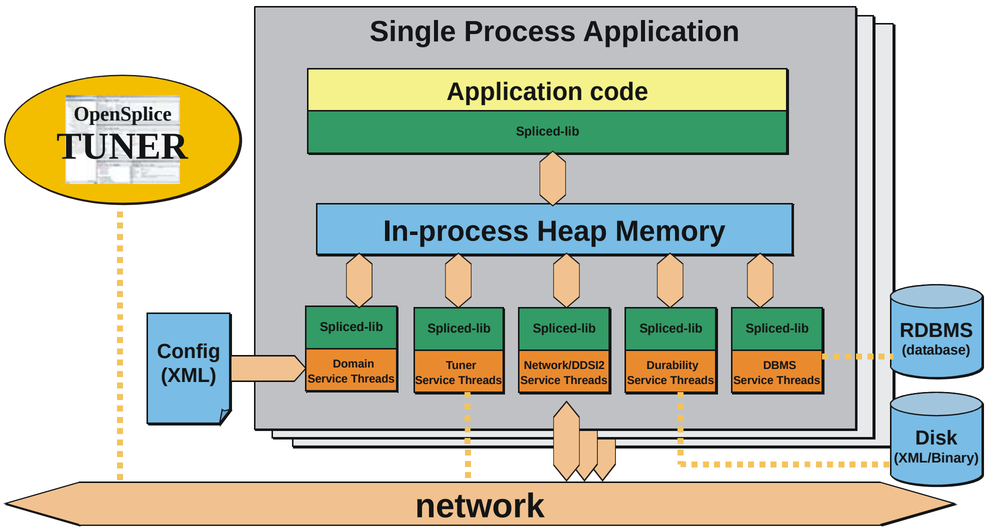


Figure 1 The OpenSplice 'standalone' Single Process Architecture

1.1.2 Shared Memory architecture

In the 'federated' shared memory architecture data is physically present only once on any machine but smart administration still provides each subscriber with his own private view on this data. Both the DDS applications and OpenSplice administration interface directly with the shared memory which is created by the OpenSplice daemon on start up. This architecture enables a subscriber's data cache to be seen as an individual database and the content can be filtered, queried, etc. by using the OpenSplice content subscription profile.

Typically for advanced DDS users, the shared memory architecture is a more powerful mode of operation and results in extremely low footprint, excellent scalability and optimal performance when compared to the implementation where each reader/writer are communication end points each with its own storage (i.e. historical data both at reader and writer) and where the data itself still has to be moved, even within the same platform.

Figure 2 shows an overview of the shared memory architecture of OpenSplice DDS on one computing node. Typically, there are many nodes within a system.

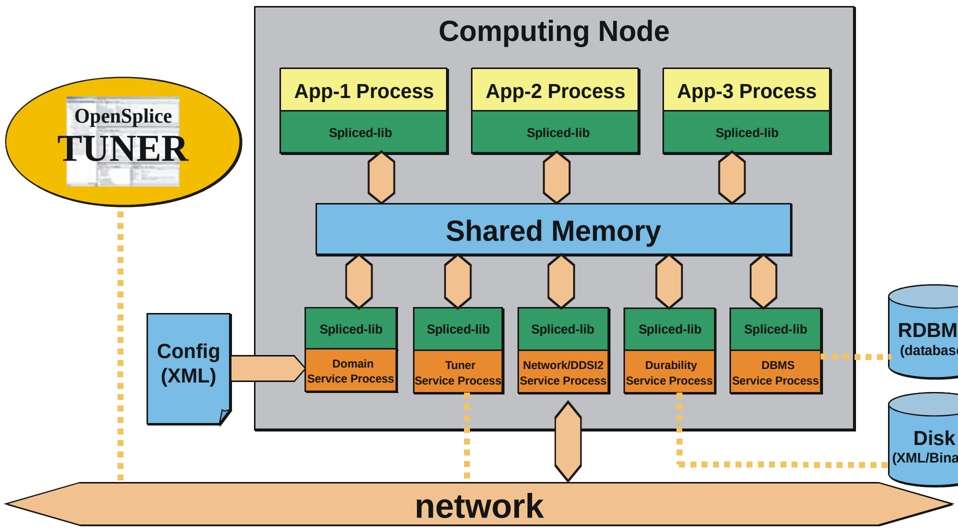


Figure 2 The OpenSplice 'federated' Shared Memory Architecture

1.1.3 Comparison of Deployment Architectures

Simple when sufficient, Performant when required

The choice between the 'federated' or 'standalone' deployment architecture is basically about going for out-of-the-box simplicity or for maximum performance:

Federated Application Cluster

- Co-located applications *share* a common set of pluggable services (daemons)
- Resources (*e.g.* memory/networking) are managed *per* 'federation'
- Added value: *performance* (scalability *and* determinism)

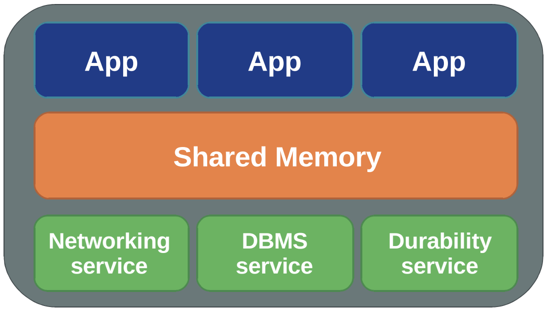


Figure 3 Federated Application Cluster

Non-federated,'single process' Applications

- Each application links the required DDS services as libraries into a standalone 'single process'
- Resources are managed by each individual application
- Added value: *Ease-of-use* ('*zero-configuration*', middleware lifecycle is simply coupled to that of the application process)

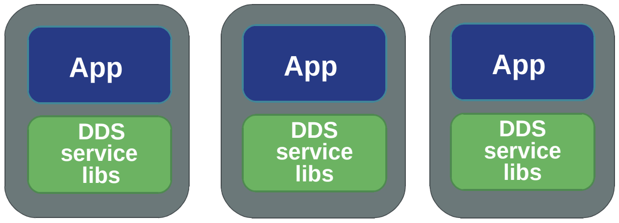


Figure 4 Non-federated, 'single process' Applications

1.1.4 Configuring and Using the Deployment Architectures

The deployment architecture choice between a shared-memory federation or a standalone 'single process' is a runtime choice driven by a simple single configuration parameter in the domain configuration xml file:

```
<SingleProcess>true</SingleProcess>
```

Note that there is absolutely *no need* to recompile or even re-link an application when selecting or changing the deployment architecture.

The deployment modes can be mixed at will, so even on a single computing node, one could have some applications that are deployed as a federation as well as other applications that are deployed as individual 'single processes'.

To facilitate the 'out-of-the-box' experience, the default `ospl.xml` configuration file specifies the standalone 'single process' deployment architecture where the middleware is simply linked as libraries into an application: no need to configure shared-memory, no need to 'fire up' OpenSplice first to start the related services. The middleware lifecycle (and with that the information lifecycle) is directly coupled to that of the application.

When, with growing system scale, scalability and determinism require efficient sharing of memory and networking resources, the deployment architecture can be switched easily to the federated archtirecture; thereafter the middleware and application(s) lifecycles are decoupled and a single set of services facilitate the federation of applications with regard to scheduling data transfers over the wire (based upon the actual importance and urgency of each published data-sample), maintaining data for late joining applications (on the same or other nodes in the system) and efficient (single-copy) sharing of all data within the computing node regardless of the number of applications in the federation.

The OpenSplice DDS distribution contains multiple example configuration files that exploit both deployment architectures. Configurations that exploit the single-process architecture start with `ospl_sp_` whereas federated-deployment configurations start with `ospl_shmem_`.

1.2 OpenSplice DDS Usage

The OpenSplice environment has to be set up to instruct the node where executables and libraries can be found in order to be able to start the Domain Service.

On UNIX-like platforms this can be realized by starting a shell and sourcing the release.com file located in the root directory of the OpenSplice installation:

```
%  . ./release.com
```

On the Windows platform the environment must be set up by running `release.bat`, or else the OpenSplice DDS Command Prompt must be used.

1.2.1 Starting OpenSplice DDS for a Single Process Deployment

For ‘standalone’ single process deployment, there is no need to start the OpenSplice DDS middleware before starting the DDS application, since the application itself will implicitly start the library threads of OpenSplice Domain Service and associated services at the point when the DDS `create_participant` operation is invoked by the standalone application process.

1.2.2 Starting OpenSplice DDS for a Shared Memory Deployment

For a shared memory deployment, it is necessary to start the OpenSplice DDS Domain Service prior to running a DDS application. The `ospl` command-tool is provided to manage OpenSplice DDS for shared memory deployments. To start OpenSplice DDS in this way, enter:

```
%  ospl start
```

This will start the Domain Service using the default configuration.

NOTE: The **Integrity** version of OpenSplice DDS does not include the `ospl` program. Instead there is a project generator, `ospl_projgen`, which generates projects containing the required address spaces which will auto-start when loaded. Please refer to the *Getting Started Guide* for details.

NOTE: The **VxWorks** version of OpenSplice DDS does not include the `ospl` program. Please refer to the *Getting StartedGuide* for details of how to use VxWorks projects and Real Time Processes to deploy OpenSplice DDS applications.

1.2.3 Monitoring

The OpenSplice Domain Service can be monitored and tuned in numerous ways after it has been started. The monitoring and tuning capabilities are described in the following subsections.

1.2.3.1 Diagnostic Messages

OpenSplice outputs diagnostic information. This information is written to the *`ospl-info.log`* file located in the start-up directory, by default. Error messages are written to the *`ospl-error.log`* file, by default. The state of the system can be determined from the information written into these files.

The location where the information and error messages are stored can be overridden by setting the `OSPL_LOGPATH` environment variable to a location on disk (by specifying a path), to standard out (by specifying <*stdout*>) or to standard error (by specifying <*stderr*>). The names of these log files can also be changed by setting the `OSPL_INFOFILE` and `OSPL_ERRORFILE` variables.

OpenSplice also accepts the environment properties `OSPL_VERBOSITY` and `OSPL_LOGAPPEND`. These provide an alternate method of specifying values for *Attribute append* and *Attribute verbosity* of the `Domain/Report` configuration element. See Section 4.2.19, *Element Report*, on page 130.

Values specified in the domain configuration override the environment values.

1.2.3.2 OpenSplice Tuner

The intention of OpenSplice Tuner, *`ospltun`*, is to provide facilities for monitoring and controlling OpenSplice, as well as the applications that use OpenSplice for the distribution of data. The *OpenSplice Tuner User Guide* specifies the capabilities of OpenSplice Tuner and describes how they should be used.

Note that the Tuner will only be able to connect to the memory running in a particular DDS Domain by being ran on a node that is already running OpenSplice DDS using the shared memory deployment mode.

The Tuner will also be able to monitor and control a Domain ran as a single process if the Tuner itself is started as the single process application with other DDS applications clustered in the process by deploying as a single process application cluster. Please refer to Section 4.2.10, *Element Application*, on page 111, for a description of how to cluster applications together in a single process.

1.2.3.3 OpenSplice Memory Management Statistics Monitor

The OpenSplice Memory Management Statistics Tool, *`mmstat`*, provides a command line interface that allows monitoring the status of the nodal shared administration (shared memory) used by the middleware and the applications. Use the help switch (*`mmstat -h`*) for usage information. Please refer to Section 3.4, *mmstat: Memory Management Statistics*, on page 86, for detailed information about *`mmstat`*.

Please note that *`mmstat`* is only suitable for diagnostic purposes, and its use is only applicable in shared memory mode.

1.2.4 Stopping OpenSplice DDS

1.2.4.1 Stopping a Single Process deployment

When deployed as a single process, the application can either be terminated naturally when the end of the main function is reached, or stopped prematurely by means of a signal interrupt, for example Ctrl-C. In either case, the OpenSplice DDS middleware running within the process will be stopped and the process will terminate.

1.2.4.2 Stopping a Shared Memory deployment

In shared memory deployment mode, the OpenSplice Domain Service can be stopped by issuing the following command on the command-line.

```
%  ospl stop
```

The OpenSplice Domain Service will react by announcing the shutdown using the shared administration. Applications will not be able to use DDS functionality anymore and services will terminate elegantly. Once this has succeeded, the Domain Service will destroy the shared administration and finally terminate itself.

1.2.4.2.1 Stopping OSPL by using signals

Alternatively the OpenSplice DDS domain service can also be stopped by sending a signal to the ospl process, assuming the process was started using the `-f` option. For example, on Unix you could use the following command to send a termination signal to the `ospl` tool, where `pid` identifies the `ospl` tool pid:

.

```
%  kill –SIGTERM <pid>
```

Sending such a signal will cause the `ospl` tool to exit gracefully, properly terminating all services and exiting with returncode `0`.

The following table shows a list of all POSIX signals and what the behavior of OSPL is when that signal is sent to the `ospl` tool.

Signal	Default action	OSPL action	Description
`SIGHUP`	Term.	Graceful exit	Hang up on controlling process
`SIGINT`	Term.	Graceful exit	Interrupt from keyboard
`SIGQUIT`	Core	Graceful exit	Quit from keyboard
`SIGILL`	Core	Graceful exit	Illegal instruction
`SIGABRT`	Core	Graceful exit	Abort signal from abort function
`SIGFPE`	Core	Graceful exit	Floating point exception
`SIGKILL`	Term.	Term.	Kill signal (can't catch, block, ignore)
`SIGSEGV`	Core	Graceful exit	Invalid memory reference
`SIGPIPE`	Term.	Graceful exit	Broken pipe: write to pipe with no readers
`SIGALRM`	Term.	Graceful exit	Timer signal from alarm function
`SIGTERM`	Term.	Graceful exit	Termination signal
`SIGUSR1`	Term.	Graceful exit	User defined signal 1
`SIGUSR2`	Term.	Graceful exit	User defined signal 2
`SIGCHLD`	Ignore	Ignore	A child process has terminated or stopped
`SIGCONT`	Ignore	Ignore	Continue if stopped
`SIGSTOP`	Stop	Stop	Stop process (can't catch, block, ignore)
`SIGTSTOP`	Stop	Graceful exit	Stop typed at tty
`SIGTTIN`	Stop	Graceful exit	Tty input for background process
`SIGTTOUT`	Stop	Graceful exit	Tty output for background process

1.2.4.2.2 Stopping Applications in Shared Memory Mode

Applications that are connected to and use OpenSplice DDS in shared memory mode must *not* be terminated with a `KILL` signal. This will ensure that OpenSplice DDS shared memory always remains in a valid, functional state.

When OpenSplice applications terminate naturally, a cleanup mechanism is executed that releases any references held to the shared memory within OpenSplice which that application was using. This mechanism will be executed even when an application is terminated by other means, *e.g.* by terminating with **Ctrl+C**, or even if the application crashes in the user code.

The cleanup mechanisms are *not* executed when an application is terminated with a `KILL` signal. For this reason a user must not terminate an application with a `kill -9` command (or, on Windows, must not use TaskManager's **End Process** option) because the process will be forcibly removed and the cleanup mechanisms will be prevented from executing. If an application is killed in this manner, the shared memory regions of OpenSplice will become inconsistent and no recovery will then be possible other than re-starting OpenSplice and all applications on the node.

1.2.5 Deploying OpenSplice DDS on VxWorks 6.x

The **VxWorks** version of OpenSplice DDS does not include the `ospl` program. Please refer to Chapter 7 of the *Getting StartedGuide* for details of how to use VxWorks projects and Real Time Processes to deploy OpenSplice DDS applications.

1.2.6 Deploying OpenSplice DDS on Integrity

The **Integrity** version of OpenSplice DDS does not include the `ospl` program. Instead there is a project generator, `ospl_projgen`, which generates projects containing the required address spaces which will auto-start when loaded. Please refer to Chapter 8 of the *Getting Started Guide* for detailed information about OpenSplice deployment on Integrity.

1.2.7 Installing/Uninstalling the OpenSplice DDS C# Assembly to the Global Assembly Cache

The installer for the commercial distribution of OpenSplice DDS includes the option to install the C# Assembly to the Global Assembly Cache during the installation process. If you chose to omit this step, or you are an open source user, then you should follow the instructions in the next few paragraphs, which describe how to manually install and uninstall the assembly to the Global Assembly Cache.

1.2.7.1 Installing the C# Assembly to the Global Assembly Cache

To install an assembly to the Global Assembly Cache, you need to use the `gacutil.exe` tool. Start a Visual Studio command prompt and type:

```
%  gacutil /i <OpenSpliceDDS installation
path>\bin\dcpssacsAssembly.dll
```

where `<OpenSpliceDDS installation path>` is the installation path of the OpenSplice DDS distribution. If you are successful you will see a message similar to the following:

```
%  C:\Program Files\Microsoft Visual Studio 9.0\VC>gacutil.exe /i
"C:\Program Files \PrismTech\OpenSpliceDDS\V5.1.0\HDE\x86.win32\
bin\dcpssacsAssembly.dll"

%

%  Microsoft (R) .NET Global Assembly Cache Utility.  Version
3.5.30729.1

%  Copyright (c) Microsoft Corporation.  All rights reserved.

%

%  Assembly successfully added to the cache

%

%  C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

1.2.7.2 Uninstalling the C# Assembly from the Global Assembly Cache

To uninstall an assembly from the Global Assembly Cache, you need to use the `gacutil.exe` tool. Start a Visual Studio command prompt and type:

```
%  gacutil /u dcpssacsAssembly,Version=<version_number_goes_here>
```

The version number of the assembly is defined in the `<OpenSpliceDDS installation path>\etc\RELEASEINFO` file, in the `CS_DLL_VERSION` variable.

If you are successful you will see a message similar to the following:

```
%  C:\Program Files\Microsoft Visual Studio 9.0\VC>gacutil /u
dcpssacsAssembly,Version=5.1.0.14734

%  Microsoft (R) .NET Global Assembly Cache Utility.  Version
3.5.30729.1

%  Copyright (c) Microsoft Corporation.  All rights reserved.

%

%  Assembly: dcpssacsAssembly, Version=5.1.0.14734,
Culture=neutral, PublicKeyToken=5b9310ab51310fa9,
processorArchitecture=MSIL
```

```
%  Uninstalled: dcpssacsAssembly, Version=5.1.0.14734,
Culture=neutral, PublicKeyToken=5b9310ab51310fa9,
processorArchitecture=MSIL

%  Number of assemblies uninstalled = 1

%  Number of failures = 0

%

%  C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

If you do not specify a version to the uninstall option, then all installed OpenSplice DDS C# Assemblies in the GAC, called `dcpssacsAssembly`, will be removed from the GAC, so take care with this option as it can adversely affect any deployed applications that rely on other versions of these assemblies.

We strongly recommend that every time you uninstall an OpenSplice DDS C# Assembly you specify the version you want to uninstall.

1.3 OpenSplice DDS Configuration

Each application domain has its own characteristics. Therefore OpenSplice allows configuring a wide range of parameters that influence its behaviour to be able to achieve optimal performance in every situation. This section describes generally how to instruct OpenSplice to use a configuration that is different from the default. This requires the creation of a custom configuration file and an instruction to the middleware to use this custom configuration file.

1.3.1 Configuration Files

OpenSplice expects the configuration to be defined in the XML format. The expected syntax and semantics of the configuration parameters will be discussed further on in this document. Within the context of OpenSplice, a reference to a configuration is expressed in the form of a Uniform Resource Identifier (URI). Currently, only file URIs are support (for example *`file:///opt/ospl/config/ospl.xml`*).

When OpenSplice is started, the Domain Service parses the configuration file using the provided URI. According to this configuration, it creates the DDS administration and initialises it. After that, the Domain Service starts the configured services. The Domain Service passes on its own URI to all services it starts, so they will also be able to resolve their configuration from this resource as well. (Of course, it is also possible to configure a different URI for each of the services, but usually one configuration file for all services will be the most convenient option.) The services will use default values for the parameters that have not been specified in the configuration.

1.3.2 Environment Variables

The OpenSplice middleware will read several environment variables for different purposes. These variables are mentioned in this document at several places. To some extent, the user can customize the OpenSplice middleware by adapting the environment variables.

When specifying configuration parameter values in a configuration file, environment variables can be referenced using the notation `${VARIABLE}`. When parsing the XML configuration, the Domain Service will replace the symbol with the variable value found in the environment.

1.3.2.1 The `OSPL_URI` environment variable

The environment variable `OSPL_URI` is a convenient mechanism to pass the configuration file to the Domain Service and DDS applications. The variable will refer to the default configuration that comes with OpenSplice DDS but of course can be overridden to refer to a customer configuration.

For single process mode operation this variable is *required*; see also Section 1.1.1, *Single Process architecture*, on page 4, and Section 4.2.7, *Element SingleProcess*, on page 100.

On Linux/Unix-based platforms, this variable can be initialized by sourcing the `release.com` script that is created by the OpenSplice installer.

On Windows platforms, this variable may already be initialized in your environment by the Windows installer. Alternatively, it can be set by executing the supplied `release.bat` script or the OpenSplice DDS Command Prompt.

1.3.3 Configuration of Single Process deployment

A single process deployment is enabled when the `OSPL_URI` environment variable refers to an XML configuration containing the `<SingleProcess>` attribute within the `Domain` section. See section 4.2.7 on page 100 for full details. In such a deployment, each OpenSplice DDS service including the Domain Service will be started as threads within the existing application process.

In this case there is no need to start the OpenSplice DDS administration manually since this is implicitly handled within the DDS code when the application first invokes the DDS `create_participant` operation. Since the `OSPL_URI` environment variable describes the OpenSplice system, there is no requirement to pass any OpenSplice DDS configuration parameters to the application.

1.3.4 Configuration of Shared Memory deployment

In order to have OpenSplice start with a custom configuration file, use:

```
%  ospl start <URI>
```

where `URI` denotes the URI of the Domain Service configuration file. In order to stop a specific OpenSplice instance, the same mechanism holds. Use:

```
%  ospl stop <URI>
```

Several instances of OpenSplice can run simultaneously, as long as their configurations specify different domain names. Typically, only one instance of the middleware is needed. Multiple instances of the middleware are only required when one or more applications on the computing node participate in different or multiple DDS Domains. At any time, the system can be queried for all running OpenSplice instances by using the command:

```
%  ospl list
```

To stop all active OpenSplice Domains, use:

```
%  ospl -a stop
```

Note that the `<URI>` parameter to the above commands is not required if the `OSPL_URI` environment variable refers to the configuration that is intended to be started or stopped.

1.3.5 Temporary Files

Please note that for a shared memory deployment, OpenSplice uses temporary files that are used to describe the shared memory that has been created. The exact nature of these files varies according to the operating system; however, the user does not need to manipulate these files directly.

On Linux systems the location of the temp files is `/tmp` by default, while on Windows the location is the value of the `TEMP` (or `TMP` if `TEMP` is not set) environment variable. These location can be over-ridden, if required, by setting the `OSPL_TEMP` variable to a location on disk by specifying a path. Please note, however, that this *must* be consistent for *all* environments on a particular node.

1.4 Applications which operate in multiple domains

OpenSplice can be configured to allow a DDS application to operate in multiple domains.

⚠ Please note that an application operating in multiple domains is currently only supported in shared memory deployments.

In order to achieve multi-domain operation, the host node for the application must run OpenSplice instances for every domain in which applications on that node will interact. For example, if an application A wants to operate in domains X, Y and Z then the node on which application A operates must run appropriate services for X, Y and Z.

OpenSplice utilises shared memory regions for intra-node communication. Each domain running on a node must have its own shared memory region, and subsequently the shared memory region for each domain that an application wants to operate within must be mapped into that application's virtual address space. The mapping must occur at a virtual address in memory that is common to both the OpenSplice daemon (and any services) for that domain and the application itself.

This requires some thought when configuring multiple OpenSplice domains on a single node. Care must be taken to ensure that the XML configuration files contain unique and non-overlapping addresses for the shared memory mapping (see the XML element "`OpenSplice/Domain/Database/Address`" in section 4.2.8.3, *Element Address*, on page 102).

When designing and coding applications, care must also be taken with regard to usage of the default domain. If a domain is not explicitly identified in the application code, then appropriate steps must be taken at deployment in order to ensure that applications operate in the domain they were intended to.

1.4.1 Interaction with a Networking Service

Where multiple domains are running on a single node, each domain must run its own instance of a networking service if that domain is to participate in remote communication.

- Each domain should have its own pre-determined port numbers configured in the XML for that domain.
- These port numbers must be common for that domain across the system.

CHAPTER

2 The OpenSplice DDS Services

2.1 Introduction

The OpenSplice DDS middleware and its services can be configured using easy to maintain XML files. These services are described in the following sections, and the XML configuration is described in detail in Chapter 4, *Service Configuration*.

The OpenSplice DDS middleware includes several services: each service has a particular responsibility. *Figure 2* on page 6 shows the services included with OpenSplice DDS. Each service can be enabled or disabled. The services can be configured or tuned to meet the optimum requirements of a particular application domain (noting that detailed knowledge of the requirement is needed for effective tuning).

The following sections briefly explain each of the services and their responsibilities.

2.2 The Domain Service

The Domain Service is responsible for creating and initialising the database which is used by the administration to manage the DDS data.

In the single process architecture the Domain Service is started as a new thread within the DDS application. This is done implicitly when the application invokes the DDS `create_participant` operation and no such service currently exists within the process. The Domain Service creates the DDS database within the heap memory of the process and so is limited only to the maximal heap that the operating system supports.

In the shared memory architecture, the user is responsible for managing the DDS administration separately from the DDS application. In this mode, the Domain Service is started as a separate process; it then creates and initialises the database by allocating a particular amount of shared memory as dictated by the configuration. Without this administration, no other service or application is able to participate in the DDS Domain.

In either deployment mode, once the database has been initialised, the Domain Service starts the set of pluggable services. In single process mode these services will be started as threads within the existing process, while in shared memory mode the services will be represented by new separate processes that can interface with the shared memory segment.

When a shutdown of the OpenSplice Domain Service is requested in shared memory mode, it will react by announcing the shutdown using the shared administration. Applications will not be able to use DDS functionality anymore and services are requested to terminate elegantly. Once this has succeeded, the Domain Service will destroy the shared administration and finally terminate itself.

The exact fulfilment of these responsibilities is determined by the configuration of the Domain Service. There is an overview of the available configuration parameters and their purpose in Section 4.2, *The Domain Service*, on page 94.

2.3 The Durability Service

This section provides a description the most important concepts and mechanisms of the current durability service implementation, starting with a description of the purpose of the service. After that all its concepts and mechanisms are described.

The exact fulfilment of the durability responsibilities is determined by the configuration of the Durability Service. There is an overview of the available configuration parameters and their purpose in Section 4.3, *The Durability Service*, on page 149.

2.3.1 Purpose

OpenSplice DDS will make sure data is delivered to all ‘compatible’ subscribers that are available at the time the data is published using the ‘communication paths’ that are implicitly created by the middleware based on the interest of applications that participate in the domain. However, subscribers that are created after the data has been published (called late-joiners) may also be interested in the data that was published before they were created (called historical data). To facilitate this use case, DDS provides a concept called durability in the form of a Quality of Service (DurabilityQosPolicy).

The DurabilityQosPolicy prescribes how published data needs to be maintained by the DDS middleware and comes in four flavours:

VOLATILE — Data does not need to be maintained for late-joiners (default).

TRANSIENT_LOCAL — Data needs to be maintained for as long as the DataWriter is active.

TRANSIENT — Data needs to be maintained for as long as the middleware is running on at least one of the nodes.

PERSISTENT — Data needs to outlive system downtime. This implicates that it must be kept somewhere on permanent storage in order to be able to make it available again for subscribers after the middleware is restarted.

In OpenSplice DDS, the realisation of the non-volatile properties is the responsibility of the durability service. Maintaining and providing of historical data could in theory be done by a single durability service in the domain, but for fault-tolerance and efficiency purposes one durability service is usually running on every computing node. These durability services are on one hand responsible for maintaining the set of historical data and on the other hand responsible for providing historical data to late-joining subscribers. The configurations of the different services drive the behaviour on where and when specific data will be maintained and how it will be provided to late-joiners.

2.3.2 Concepts

The following subsections describe the concepts that drive the implementation of the OpenSplice durability service.

2.3.2.1 Role and scope

Each OpenSplice node can be configured with a so-called role. A role is a logical name and different nodes can be configured with the same role. The role itself does not impose anything, but multiple OpenSplice services use the role as a mechanism to distinguish behaviour between nodes with the equal and different roles.

The durability service allows configuring a so-called scope, which is an expression that is matches against roles of other nodes. By using a scope, the durability service can be instructed to apply different behaviour with respect to merging of historical data sets (see section *2.3.2.3.4* on page 25 about merge-policies) to and from nodes that have equal or different roles.

See //OpenSplice/Domain/Role (section *4.2.4* on page 97).

See //OpenSplice/DurabilityService/NameSpaces/Policy/Merge[@scope] (section *4.3.4.2.3.2* on page 180).

2.3.2.2 Name-spaces

A sample published in DDS for a specific topic and instance is bound to one logical partition. This means that in case a publisher is associated with multiple partitions, a separate sample for each of the associated partitions is created. Even though they are syntactically equal, they have different semantics (consider for instance the situation where you have a sample in the ‘simulation’ partition versus one in the ‘real world’ partition).

Because applications might impose semantic relationships between instances published in different partitions, a mechanism is required to express this relationship and ensure consistency between partitions. For example, an application might expect a specific instance in partition *Y* to be available when it reads a specific instance from partition *X*.

This implies that the data in both partitions need to be maintained as one single set. For persistent data, this dependency implies that the durability services in a domain needs to make sure that this data set is re-published from one single persistent store instead of combining data coming from multiple stores on disk. To express this semantic relation between instances in different partitions to the durability service, the user can configure so-called 'name-spaces' in the durability configuration file. Each name-space is formed by a collection of partitions and all instances in such a collection are always handled as an atomic data-set by the durability service. In other words, the data is guaranteed to be stored and reinserted as a whole.

This atomicity also implies that name-spaces are a system wide concept meaning that different durability services need to agree on its definition, *i.e.* which partitions belong to one name-space. This doesn't mean that each durability service needs to know all name-spaces; as long as the name-spaces one does know don't conflict with one of the others in the domain. Name-spaces that are completely disjoint can co-exist (their intersection is an empty set) and name-spaces conflict when they intersect. For example: name-spaces {p1, q} and {p2, r} can co-exist, but name-spaces {s, t} and {s, u} cannot.

Furthermore it is important to know that there is a set of configurable policies for name-spaces, allowing durability services throughout the domain to take different responsibilities for each name-space with respect to maintaining and providing of data that belongs to the name-space. The durability name-spaces define the mapping between logical partitions and the responsibilities that a specific durability service needs to play. In the default configuration file there is only one name-space by default (holding all partitions).

Next to the capability of associating a semantic relationship for data in one name-space, the need to differentiate the responsibilities of a particular durability service for a specific data-set is the second purpose of a name-space. Even though there may not be any relation between instances in different partitions, the choice of grouping specific partitions in different name-spaces can still be perfectly valid. The need for availability of non-volatile data under specific conditions (fault-tolerance) on one hand versus requirements on performance (memory usage, network bandwidth, CPU usage, etc.) on the other hand may force the user to split up the maintaining of the non-volatile data-set over multiple durability services in the domain. Illustrative of this balance between fault-tolerance and performance is the example of maintaining all data in all durability services, which is maximally fault-tolerant, but also requires the most resources. The name-spaces concept allows the user to divide the total set of non-volatile data over multiple name-spaces and assign different responsibilities to different durability-services in the form of so-called name-space policies.

See //OpenSplice/DurabilityService/NameSpaces/NameSpace
(section *4.3.4.1* on page 176).

2.3.2.3 Name-space policies

This section describes the policies that can be configured per name-space giving the user full control over the fault-tolerance versus performance aspect on a *per name-space* level.

See //OpenSplice/DurabilityService/NameSpaces/Policy
(section *4.3.4.2* on page 177).

2.3.2.3.1 Alignment policy

The durability services in a domain are on one hand responsible for maintaining the set of historical data between services and on the other hand responsible for providing historical data to late joining applications. The configurations of the different services drive the behaviour on where and when specific data will be kept and how it will be provided to late-joiners. The optimal configuration is driven by fault-tolerance on one hand and resource usage (like CPU usage, network bandwidth, disk space and memory usage) on the other hand. One mechanism to control the behaviour of a specific durability service is the usage of alignment policies that can be configured in the durability configuration file. This configuration option allows a user to specify if and when data for a specific name-space (see section about name-spaces) will be maintained by the durability service and whether or not it is allowed to act as an aligner for other durability services when they require (part of) the information.

The alignment responsibility of a durability service is therefore configurable by means of two different configuration options being the aligner and alignee responsibilities of the service:

Aligner policy

`TRUE` — The durability service will align others if needed.

`FALSE` — The durability service will not align others.

Alignee policy

`INITIAL` — Data will be retrieved immediately when the data is available and continuously maintained from that point forward.

`LAZY` — Data will be retrieved on first arising interest on the local node and continuously maintained from that point forward.

`ON_REQUEST` — Data will be retrieved only when requested by a subscriber, but not maintained. Therefore each request will lead to a new alignment action.

See //OpenSplice/DurabilityService/NameSpaces/Policy[@aligner]
(section *4.3.4.2.6* on page 182)

See //OpenSplice/DurabilityService/NameSpaces/Policy[@alignee] (section *4.3.4.2.5* on page 181).

2.3.2.3.2 Durability policy

The durability service is capable of maintaining (part of) the set of non-volatile data in a domain. Normally this results in the outcome that data which is written as volatile is not stored, data written as transient is stored in memory and data that is written as persistent is stored in memory and on disk. However, there are use cases where the durability service is required to 'weaken' the DurabilityQosPolicy associated with the data, for instance by storing persistent data only in memory as if it were transient. Reasons for this are performance impact (CPU load, disk I/O) or simply because no permanent storage (in the form of some hard-disk) is available on a node. Be aware that it is not possible to 'strengthen' the durability of the data (Persistent > Transient > Volatile). The durability service has the following options for maintaining a set of historical data:

`PERSISTENT` — Store persistent data on permanent storage, keep transient data in memory and don't maintain volatile data.

`TRANSIENT` — Keep both persistent and transient data in memory and don't maintain volatile data.

`VOLATILE` — Don't maintain persistent, transient or volatile data.

This configuration option is called the 'durability policy'.

See //OpenSplice/DurabilityService/NameSpaces/Policy[@durability] (section *4.3.4.2.4* on page 180).

2.3.2.3.3 Delayed alignment policy

The durability service has a mechanism in place to make sure that when multiple services with a persistent dataset exist, only one set (typically the one with the newest state) will be injected in the system (see Section 2.3.4.4, *Persistent data injection*, on page 32). This mechanism will, during the startup of the durability service, negotiate with other services which one has the best set (see Section 2.3.4.3, *Master selection*, on page 32). After negotiation the 'best' persistent set (which can be empty) is restored and aligned to all durability services.

Once persistent data has been re-published in the domain by a durability service for a specific name-space, other durability services in that domain cannot decide to re-publish their own set for that name-space from disk any longer. Applications may already have started their processing based on the already published set and re-publishing another set of data may confuse the business logic inside applications. Other durability services will therefore back-up their own set of data and align and store the set that is already available in the domain. It is important to realise that an empty set of data is also considered a set. This means that once a durability service in the domain decides that there is no data (and has triggered applications that the set is complete), other late-joining durability services will not re-publish their persistent data that they potentially have available.

Some systems however do require re-publishing persistent data from disk if the already re-published set is empty and no data has been written for the corresponding name-space. The durability service can be instructed to still re-publish data from disk in this case by means of an additional policy in the configuration called 'delayed alignment'. This Boolean policy instructs a late-joining durability service whether or not to re-publish persistent data for a name-space that has been marked complete already in the domain, but for which no data exists and no DataWriters have been created. Whatever setting is chosen, it should be consistent between all durability services in a domain to ensure proper behaviour on system level.

See //OpenSplice/DurabilityService/NameSpaces/Policy[@delayedAlignment] (section *4.3.4.2.2* on page 178).

2.3.2.3.4 Merge policy

A split-brain syndrome can be described as the situation in which two different nodes (possibly) have a different perception on (part of) the set of historical data. This split-brain occurs when two nodes or two sets of nodes (*i.e.* two systems) that are participating in the same DDS domain have been running separately for some time and suddenly get connected to each other. Equally this syndrome arises when nodes re-connect after being disconnected for some time. Applications on these nodes may have been publishing information for the same topic in the same partition without this information reaching the other party. Therefore their perception on the set of data will be different.

In many cases, exchanging information after this is no longer allowed, because there is no guarantee that data between the connected systems doesn't conflict. For example, consider a fault-tolerant (distributed) global id service: this service will provide globally-unique ids, this however will only be guaranteed if and only if there is no disruption of communication between all services. In such a case a disruption must be considered permanent and a reconnection must be avoided at any cost.

Some new environments demand supporting the possibility to (re)connect two separate systems though. One can think of ad-hoc networks where nodes dynamically connect when they are near each other and disconnect again when they're out of range, but also systems where temporal loss of network connections is normal. Another use case is the deployment of OpenSplice DDS in a hierarchical network, where higher-level 'branch' nodes need to combine different historical data sets from multiple 'leaves' into its own data set. In these new environments there is the same strong need for the availability of data for 'late-joining' applications (non-volatile data) as in any other system.

For these kinds of environments the durability service has additional functionality to support the alignment of historical data when two nodes get connected. Of course, the basic use case of a newly started node joining an existing system is supported, but in contradiction to that situation there is no universal truth in determining who has the best (or the right) information when two already running nodes (re)connect. When this situation occurs, the durability service provides the following possibilities to handle the situation:

`IGNORE` — Ignore the situation and take no action at all. This means new knowledge is not actively built up. Durability is passive and will only build up knowledge that is ‘implicitly’ received from that point forward (simply by receiving updates that are published by applications from that point forward and delivered using the normal publish-subscribe mechanism).

`DELETE` — Dispose and delete all historical data. This means existing data is disposed and deleted and other data is not actively aligned. Durability is passive and will only maintain data that is ‘implicitly’ received from that point forward.

`REPLACE` — Dispose and replace all historical data by the data set that is available on the connecting node.

`MERGE` — Merge the historical data with the data set that is available on the connecting node.

From this point forward this set of options will be referred to as ‘merge policies’.

Like the networking service, the durability service also allows configuration of a so-called scope to give the user full control over what merge policy should be selected based on the role of the re-connecting node. The scope is a logical expression and every time nodes get physically connected, they match the role of the other party against the configured scope to see whether communication is allowed and if so, whether a merge action is required.

As part of the merge policy configuration, one can also configure a scope. This scope is matched against the role of remote durability services to determine what merge policy to apply. Because of this scope, the merge behaviour for (re-)connections can be configured on a *per role* basis. It might for instance be necessary to merge data when re-connecting to a node with the same role, whereas (re-)connecting to a node with a different role requires no action.

See //OpenSplice/DurabilityService/NameSpaces/Policy/Merge
(section *4.3.4.2.3* on page 179).

2.3.2.4 Dynamic name-spaces

As specified in the previous sections, a set of policies can be configured for a (set of) given name-space(s). One may not know the complete set of name-spaces for the entire domain though, especially when new nodes dynamically join the domain.

However, in case of maximum fault-tolerance, one may still have the need to define behaviour for a durability service by means of a set of policies for name-spaces that have not been configured on the current node.

Every name-space in the domain is identified by a logical name. To allow a durability service to fulfil a specific role for any name-space, each policy needs be configured with a name-space expression that is matched against the name of name-spaces in the domain. If the policy matches a name-space, it will be applied by the durability service, independent of whether or not the name-space itself is configured on the node where this durability service runs. This concept is referred to as 'dynamic name-spaces'.

See //OpenSplice/DurabilityService/NameSpaces/Policy[@nameSpace]
(section *4.3.4.2.1* on page 178).

2.3.2.5 Master/slave

Each durability service that is responsible for maintaining data in a namespace must maintain the complete set for that namespace. It can achieve this by either requesting data from a durability service that indicates it has a complete set or, if none is available, request all data from all services for that namespace and combine this into a single complete set. This is the only way to ensure all available data will be obtained. In a system where all nodes are started at the same time, none of the durability services will have the complete set, because applications on some nodes may already have started to publish data. In the worst case every service that starts then needs to ask every other service for its data. This concept is not very scalable and also leads to a lot of unnecessary network traffic, because multiple nodes may (partly) have the same data. Besides that, start-up times of such a system will exponentially grow when adding new nodes. Therefore the so-called 'master' concept has been introduced.

Durability services will determine one 'master' for every name-space per configured role amongst themselves. Once the master has been selected, this master is the one that will obtain all historical data first (this also includes re-publishing its persistent data from disk) and all others wait for that process to complete before asking the master for the complete set of data. The advantage of this approach is that only the master (potentially) needs to ask all other durability services for their data and all others only need to ask just the master service for its complete set of data after that.

Additionally, a durability service is capable of combining alignment requests coming from multiple remote durability services and will align them all at the same time using the internal multicast capabilities. The combination of the master concept and the capability of aligning multiple durability services at the same time make the alignment process very scalable and prevent the start-up times from growing when the number of nodes in the system grows. The timing of the durability protocol can be tweaked by means of configuration in order to increase chances of combining alignment requests. This is particularly useful in environments where multiple nodes or the entire system is usually started at the same time and a considerable amount of non-volatile data needs to be aligned.

2.3.3 Mechanisms

2.3.3.1 Interaction with other durability services

To be able to obtain or provide historical data, the durability service needs to communicate with other durability services in the domain. These other durability services that participate in the same domain are called ‘fellows’. The durability service uses regular DDS to communicate with its fellows. This means all information exchange between different durability services is done with via standard DataWriters and DataReaders (without relying on non-volatile data properties of course).

Depending on the configured policies, DDS communication is used to determine and monitor the topology, exchange information about available historical data and alignment of actual data with fellow durability services.

2.3.3.2 Interaction with other OpenSplice services

In order to communicate with fellow durability services through regular DDS DataWriters and DataReaders, the durability service relies on the availability of a network service. This can be either the interoperable DDSI or the real-time networking service. It can even be a combination of multiple networking services in more complex environments. As networking services are pluggable like the durability service itself, they are separate processes or threads that perform tasks asynchronously next to the tasks that the durability service is performing. Some configuration is required to instruct the durability service to synchronise its activities with the configured networking service(s). The durability service aligns data separately per partition-topic combination. Before it can start alignment for a specific partition-topic combination it needs to be sure that the networking service(s) have detected the partition-topic combination and ensure that data published from that point forward is delivered from c.q. sent over the network. The durability service needs to be configured to instruct it which networking service(s) need to be attached to a partition-topic combination before starting alignment. This principle is called ‘wait-for-attachment’.

Furthermore, the durability service is responsible to announce its liveliness periodically with the splice-daemon. This allows the splice-daemon to take corrective measures in case the durability service becomes unresponsive. The durability service has a separate so-called watch-dog thread to perform this task. The configuration file allows configuring the scheduling class and priority of this watch-dog thread.

Finally, the durability service is also responsible to monitor the splice-daemon. In case the splice-daemon itself fails to update its lease or initiates regular termination, the durability service will terminate automatically as well.

See //OpenSplice/DurabilityService/Network
(section *4.3.2* on page 150).

2.3.3.3 Interaction with applications

The durability service is responsible to provide historical data to late-joining subscribers. Applications can use the DCPS API call `wait_for_historical_data` on a DataReader to synchronise on the availability of the complete set of historical data. Depending on whether the historical data is already available locally, data can be delivered immediately after the DataReader has been created or must be aligned from another durability service in the domain first. Once all historical data is delivered to the newly-created DataReader, the durability service will trigger the DataReader unblocking the `wait_for_historical_data` performed by the application. If the application does not need to block until the complete set of historical data is available before it starts processing, there is no need to call `wait_for_historical_data`. It should be noted that in that case historical data still is delivered by the durability service when it becomes available.

2.3.3.4 Parallel alignment

When a durability service is started and joins an already running domain, it usually obtains historical data from one or more already running durability services. In case multiple durability services are started around the same time, each one of them needs to obtain a set of historical data from the already running domain. The set of data that needs to be obtained by the various durability services is often the same or at least has a large overlap. Instead of aligning each newly joining durability service separately, aligning all of them at the same time is very beneficial, especially if the set of historical data is quite big. By using the built-in multi-cast and broadcast capabilities of DDS, a durability service is able to align as many other durability services as desired in one go. This ability reduces the CPU, memory and bandwidth usage of the durability service and makes the alignment scale also in situations where many durability services are started around the same time and a large set of historical data exists. The concept of aligning multiple durability service at the same time is referred to as ‘parallel alignment’.

To allow this mechanism to work, durability services in a domain determine a master durability service for each name-space. Every durability service elects the same master for a given name-space based on a set of rules that will be explained later on in this document. When a durability service needs to be aligned, it will always send its request for alignment to its selected master. This results in only one durability service being asked for alignment by any other durability service in the domain for a specific name-space, but also allows the master to combine similar requests for historical data. To be able to combine alignment requests from different sources, a master will wait a period of time after receiving a request and before answering a request. This period of time is called the ‘request-combine period’.

The actual amount of time that defines the ‘request-combine period’ for the durability service is configurable. Increasing the amount of time will increase the likelihood of parallel alignment, but will also increase the amount of time before it will start aligning the remote durability service and in case only one request comes in within the configured period, this is non-optimal behaviour. The optimal configuration for the request-combine period therefore depends heavily on the anticipated behaviour of the system and optimal behaviour may be different in every use case.

In some systems, all nodes are started simultaneously, but from that point forward new nodes start or stop sporadically. In such systems, different configuration with respect to the request-combine period is desired when comparing the start-up and operational phases. That is why the configuration of this period is split into different settings: one during the start-up phase and one during the operational phase.

See //OpenSplice/DurabilityService/Network/Alignment/RequestCombinePeriod (section *4.3.2.5.6* on page 161).

2.3.3.5 Tracing

Configuring durability services throughout a domain and finding out what exactly happens during the lifecycle of the service can prove difficult. Especially OpenSplice developers sometimes have a need to get more detailed durability specific state information than is available in the regular OpenSplice info and error logs to be able to analyse what is happening. To allow retrieval of more internal information about the service for (off-line) analysis to improve performance or analyse potential issues, the service can be configured to trace its activities to a specific output file on disk.

By default, this tracing is turned off for performance reasons, but it can be enabled by configuring it in the XML configuration file.

The durability service supports various tracing verbosity levels. In general can be stated that the more verbose level is configured (`FINEST` being the most verbose), the more detailed the information in the tracing file will be.

See //OpenSplice/DurabilityService/Tracing (section *4.3.7* on page 187).

2.3.4 Lifecycle

During its lifecycle, the durability service performs all kinds of activities to be able to live up to the requirements imposed by the DDS specification with respect to non-volatile properties of published data. This section describes the various activities that a durability service performs to be able to maintain non-volatile data and provide it to late-joiners during its lifecycle.

2.3.4.1 Determine connectivity

Each durability service constantly needs to have knowledge on all other durability services that participate in the domain to determine the logical topology and changes in that topology (*i.e.* detect connecting, disconnecting and re-connecting nodes). This allows the durability service for instance to determine where non-volatile data potentially is available and whether a remote service will still respond to requests that have been sent to it reliably.

To determine connectivity, each durability service sends out a heartbeat periodically (every configurable amount of time) and checks whether incoming heartbeats have expired. When a heartbeat from a fellow expires, the durability service considers that fellow disconnected and expects no more answers from it. This means a new aligner will be selected for any outstanding alignment requests for the disconnected fellow. When a heartbeat from a newly (re)joining fellow is received, the durability service will assess whether that fellow is compatible and if so, start exchanging information.

See //OpenSplice/DurabilityService/Network/Heartbeat
(section *4.3.2.3* on page 151).

2.3.4.2 Determine compatibility

When a durability service detects a remote durability service in the domain it is participating in, it will determine whether that service has a compatible configuration before it will decide to start communicating with it. The reason not to start communicating with the newly discovered durability service would be a mismatch in configured name-spaces. As explained in a previous section about the name-space concept, having different name-spaces is not an issue as long as they do not overlap. In case an overlap is detected, no communication will take place between the two ‘incompatible’ durability services. Such an incompatibility in your system is considered a mis-configuration and is reported as such in the OpenSplice error log.

Once the durability service determines name-spaces are compatible with the ones of all discovered other durability services, it will continue with selection of a master for every name-space, which is the next phase in its lifecycle.

2.3.4.3 Master selection

To ensure a single source for re-publishing of persistent data and to allow parallel alignment, each durability service will select a master for every name-space. The rules for determining a master is as follows:

1. If some other durability service in the domain already selected a master, pick the same one.
2. If no master has been selected, pick the one with the newest initial set of persistent data.
3. If multiple durability services exist with the newest set of initial persistent data, pick the one with the highest id (this id is a domain-wide unique number that is generated at start-up of each OpenSplice federation)

In case an existing master is no longer available, due to a disconnection, crash or regular termination, a new master is selected based on the same rules.

See //OpenSplice/DurabilityService/Network/InitialDiscoveryPeriod (section *4.3.2.4* on page 155).

2.3.4.4 Persistent data injection

As persistent data needs to outlive system downtime, this data needs to be re-published in DDS once a domain is started. In case only one node is started, the durability service on that node can simply re-publish the persistent data from its disk. However, if multiple nodes are started at the same time, things become more difficult. Each one of them may have a different set available on permanent storage due to the fact that durability services have been stopped on a different moment in time. Therefore only one of them can be allowed re-publish its data to prevent inconsistencies and duplication of data. The steps below describe how a durability service currently determines whether or not to inject its data during start-up:

1. *Determine validity of own persistent data* — During this step the durability service determines whether its persistent store has initially been completely filled with all persistent data in the domain in the last run. In case the service was shut down in the last run during initial alignment of the persistent data, the set of data will be incomplete and the service will restore its back-up of a full set of (older) data if that is available from a run before that. This is done because it is considered better to re-publish an older but complete set of data instead of a part of a newer set.
2. *Determine quality of own persistent data* — In case persistence has been configured, the durability service will inspect the quality of its persistent data on start-up. The quality is determined on a per-name-space level by looking at the time-stamps of the persistent data on disk. The latest time-stamp of the data on disk is used as the quality of the name-space. This information is useful when

multiple nodes are started at the same time. Since there can only be one source per name-space that is allowed to actually inject the data from disk into DDS, this mechanism allows the durability services to select the source that has the latest data, because this is generally considered the best data. In case this is not true then an intervention is required. The data on the node must be replaced by the correct data either by a supervisory (human or system management application) replacing the data files or starting the nodes in the desired sequence so that data is replaced by alignment.

3. *Determine topology* — During this step, the durability service determines whether there are other durability services in the domain and what their state is. If this service is the only one, it will select itself as the 'best' source for the persistent data.

4. *Determine master* — During this step the durability service will determine who will inject persistent data or who has injected persistent data already. The one that will or already has injected persistent data is called the 'master'. This process is done on a *per name-space* level (see previous section).

   a) *Find existing master* – In case the durability service joins an already running domain, the master has already been determined and this one has already injected the persistent data from its disk or is doing it right now. In this case, the durability service will set its current set of persistent data aside and will align data from the already existing master node. If there is no master yet, persistent data has not been injected yet.

   b) *Determine new master* – In case the master has not been determined yet, the durability service determines the master for itself based on who has the best quality of persistent data. In case there is more than one service with the 'best' quality, the one with the highest system id (unique number) is selected. Furthermore, a durability service that is marked as not being an aligner for a name-space cannot become master for that name-space.

5. *Inject persistent data* — During this final step the durability service injects its persistent data from disk into the running domain. This is only done when the service has determined that it is the master. In any other situation the durability service backs up its current persistent store and fills a new store with the data it aligns from the master durability service in the domain or waits with alignment until a master becomes available in the domain.

2.3.4.5 Discover historical data

During this phase, the durability service finds out what historical data is available in the domain that matches any of the locally configured name-spaces. All necessary topic definitions and partition information are retrieved during this phase. This step is performed before the historical data is actually aligned from others. The process of discovering historical data continues during the entire lifecycle of the service and is based on the reporting of locally created partition-topic combinations by each durability service to all others in the domain.

2.3.4.6 Align historical data

Once all topic and partition information for all configured name-spaces are known, the initial alignment of historical data takes place. Depending on the configuration of the service, data is obtained immediately after discovering it or only once local interest in the data arises. The process of aligning historical data continues during the entire lifecycle of the durability service.

2.3.4.7 Provide historical data

Once (a part of) the historical data is available in the durability service, it is able to provide historical data to local DataReaders as well as other durability service.

Providing of historical data to local DataReaders is performed automatically as soon as the data is available. This may be immediately after the DataReader is created (in case historical data is already available in the local durability service at that time) or immediately after it has been aligned from a remote durability service.

Providing of historical data to other durability services is done only on request by these services. In case the durability service has been configured to act as an aligner for others, it will respond to requests for historical data that are received. The set of locally available data that matches the request will be sent to the durability service that requested it.

2.3.4.8 Merge historical data

When a durability service discovers a remote durability service and detects that neither that service nor the service itself is in start-up phase, it concludes that they have been running separately for a while (or the entire time) and both may have a different (but potentially complete) set of historical data. When this situation occurs, the configured merge-policies will determine what actions are performed to recover from this situation. The process of merging historical data will be performed every time two separately running systems get (re-)connected.

2.4 The Networking Service

When communication endpoints are located on different computing nodes or on different single processes, the data produced using the local Domain Service must be communicated to the remote Domain Services and the other way around. The Networking Service provides a bridge between the local Domain Service and a network interface. Multiple Networking Services can exist next to each other; each serving one or more physical network interfaces. The Networking Service is responsible for forwarding data to the network and for receiving data from the network.

There are two implementations of the networking service, the Native Networking Service and the DDSI2 Networking Service.

2.4.1 The Native Networking Service

The ‘native’ Networking Service is the optimal implementation of DDS networking for OpenSplice DDS and is both highly scalable and configurable.

The Native Networking Service can be configured to distinguish multiple communication channels with different QoS policies. These policies will be used to schedule individual messages to specific channels, which may be configured to provide optimal performance for a specific application domain.

The exact fulfilment of these responsibilities is determined by the configuration of the Networking Service. There is an overview of the available configuration parameters and their purpose in Section 4.4.1, *The Network Service*, on page 189.

2.4.2 The Secure Native Networking Service

There is a secure version of the native networking service available.

Please refer to the OpenSplice *Security Configuration Guide* for details.

2.5 The DDSI2 and DDSI2E Networking Services

The purpose and scope of the “Data-Distribution Service Interoperability Wire Protocol” is to ensure that applications based on different vendors’ implementations of DDS can interoperate. The protocol was standardized by the OMG in 2008, and was designed to meet the specific requirements of data-distribution systems.

Features of the DDSI protocol include:

- **Performance and Quality-of-Service** properties to enable best-effort and reliable publish-subscribe communications for real-time applications over standard IP networks.
- **Fault tolerance** to allow the creation of networks without single points of failure.
- **Plug-and-play Connectivity** so that new applications and services are automatically discovered and applications can join and leave the network at any time without the need for reconfiguration.
- **Configurability** to allow balancing the requirements for reliability and timeliness for each data delivery.
- **Scalability** to enable systems to potentially scale to very large networks.

DDSI-Extended (DDSI2E) is an extended version of the DDSI2 networking service, giving extra features for:

- **Network partitions:** Network partitions provide the ability to use alternative multicast addresses for combinations of DCPS topics and partitions to separate out traffic flows, for example for routing or load reduction.
- **Security:** Encryption can be configured per network partition. This enables configuring encrypted transmission for subsets of the data.
- **Bandwidth limiting and traffic scheduling:** Any number of 'network channels' can be defined, each with an associated transport priority. Application data is routed *via* the network channel with the best matching priority. For each network channel, outgoing bandwidth limits can be set and the IP 'differentiated services' options can be controlled.

The remainder of this section gives background on these two services and describes how the various mechanisms and their configuration parameters interact. A complete list of configuration parameters is in Section 4.7, *The DDSI2 and DDSI2 Enhanced Networking Service*, on page 293 .

2.5.1 DDSI Concepts

The DDSI 2.1 standard is very intimately related to the DDS 1.2 standard, with a clear correspondence between the entities in DDSI and those in DCPS. However, this correspondence is not one-to-one.

In this section we give a high-level description of the concepts of the DDSI specification, with hardly any reference to the specifics of the OpenSplice implementation, DDSI2, which are addressed in the subsequent sections. This division was chosen to aid the reader interested in interoperability in understanding where the specification ends and the OpenSplice implementation begins.

2.5.1.1 Mapping of DCPS domains to DDSI domains

In DCPS, a domain is uniquely identified by a non-negative integer, the domain id. DDSI maps this domain id to UDP/IP port numbers to be used for communicating with the peer nodes—these port numbers are particularly important for the discovery protocol—and this mapping of domain ids to UDP/IP port numbers ensures accidental cross-domain communication is impossible with the default mapping.

DDSI does not communicate the DCPS port number in the discovery protocol; it assumes each domain ids maps to unique port numbers. While it is unusual to change the mapping, the specification requires this to be possible, and this means that two different DCPS domain ids can be mapped to a single DDSI domain.

2.5.1.2 Mapping of DCPS entities to DDSI entities

Each DCPS domain participant in a domain is mirrored in DDSI as a DDSI participant. These DDSI participants drive the discovery of participants, readers and writers in DDSI *via* the discovery protocols. By default each DDSI participant has a unique address on the network in the form of its own UDP/IP socket with a unique port number.

Any data reader or data writer created by a DCPS domain participant is mirrored in DDSI as a DDSI reader or writer. In this translation, some of the structure of the DCPS domain is lost, because DDSI has no knowledge of DCPS Subscribers and Publishers. Instead, each DDSI reader is the combination of the corresponding DCPS data reader and the DCPS subscriber it belongs to; and similarly, each DDSI writer is a combination of the corresponding DCPS data writer and DCPS publisher. This corresponds to the way the DCPS built-in topics describe the DCPS data readers and data writers, as there are no built-in topics for describing the DCPS subscribers and publishers either.

In addition to the application-created readers and writers (referred to as ‘endpoints’), DDSI participants have a number of DDSI built-in endpoints used for discovery and liveliness checking/asserting. The most important ones are those absolutely required for discovery: readers and writers for the discovery data concerning DDSI participants, DDSI readers and DDSI writers. Some other ones exist as well, and a DDSI implementation can leave out some of these if it has no use for them. For example, if a participant has no writers, it doesn’t strictly need the DDSI built-in endpoints for describing writers, nor the DDSI built-in endpoint for learning of readers of other participants.

2.5.1.3 Reliable communication

Best-effort communication is simply a wrapper around UDP/IP: the packet(s) containing a sample are sent to the addresses at which the readers reside. No state is maintained on the writer. If a packet is lost, the reader will simply drop the sample and continue with the next one.

When reliable communication is used, the writer does maintain a copy of the sample, in case a reader detects it has lost packets and requests a retransmission. These copies are stored in the writer history cache (or WHC) of the DDSI writer. The DDSI writer is required to periodically send Heartbeats to its readers to ensure that all readers will learn of the presence of new samples in the WHC even when packets get lost.

If a reader receives a Heartbeat and detects it did not receive all samples, it requests a retransmission by sending an AckNack message to the writer, in which it simultaneously informs the writer up to what sample it has received everything, and which ones it has not yet received. Whenever the writer indicates it requires a response to a Heartbeat the readers will send an AckNack message even when no samples are missing. In this case, it becomes a pure acknowledgement.

The combination of these behaviours in principle allows the writer to remove old samples from its WHC when it fills up too far, and allows readers to always receive all data. A complication exists in the case of unresponsive readers, readers that do not respond to a Heartbeat at all, or that for some reason fail to receive some samples despite resending it. The specification leaves the way these get treated unspecified.

Note that while this Heartbeat/AckNack mechanism is very straightforward, the specification actually allows suppressing heartbeats, merging of AckNacks and retransmissions, &c. The use of these techniques is required to allow for a performant DDSI implementation, whilst avoiding the need for sending redundant messages.

2.5.1.4 DDSI-specific transient-local behaviour

The above describes the essentials of the mechanism used for samples of the ‘volatile’ durability kind, but the DCPS specification also provides ‘transient-local’, ‘transient and ‘persistent’ data. Of these, the DDSI specification currently only covers transient-local, and this is the only form of durable data available when interoperating across vendors.

In DDSI, transient-local data is implemented using the WHC that is normally used for reliable communication. For transient-local data, samples are retained even when all readers have acknowledged them. With the default history setting of KEEP_LAST with history_depth = 1, this means that late-joining readers can still obtain the latest sample for each existing instance.

Naturally, once the DCPS writer is deleted (or disappears for whatever reason), the DDSI writer disappears as well, and with it, its history. For this reason, transient data is generally much to be preferred over transient-local data. In OpenSplice the durability service implements all three durability kinds without requiring any special support from the networking services, ensuring the full set of durability features is always available between OpenSplice nodes.

2.5.1.5 Discovery of participants & endpoints

DDSI participants discover each other by means of the ‘Simple Participant Discovery Protocol’, or ‘SPDP’ for short. This protocol is based on periodically sending a message containing the specifics of the participant to a set of known addresses. By default, this is a standardised multicast address (239.255.0.1; the port number is derived from the domain id) that all DDSI implementations listen to.

Particularly important in the SPDP message are the unicast and multicast addresses at which the participant can be reached. Typically, each participant has a unique unicast address, which in practice means all participants on a node all have a different UDP/IP port number in their unicast address. In a multicast-capable network, it doesn't matter what the actual address (including port number) is, because all participants will learn them through these SPDP messages.

The protocol does allow for unicast-based discovery, which requires listing the addresses of machines where participants may be located, and ensuring each participant uses one of a small set of port numbers. Because of this, some of the port numbers are derived not only from the domain id, but also from a 'participant index', which is a small non-negative integer, unique to a participant within a node. (The DDSI2 service adds an indirection and uses at most one participant index regardless of how many DCPS participants it handles.)

Once two participants have discovered each other, and both have matched the DDSI built-in endpoints their peer is advertising in the SPDP message, the 'Simple Endpoint Discovery Protocol' or 'SEDP' takes over, exchanging information on the DCPS data readers and data writers in the two participants.

The SEDP data is handled as reliable, transient-local data. Therefore, the SEDP writers send Heartbeats, the SEDP readers detect they have not yet received all samples and send AckNacks requesting retransmissions, the writer responds to these and eventually receives a pure acknowledgement informing it that the reader has now received the complete set.

Note that the discovery process necessarily creates a burst of traffic each time a participant is added to the system: all existing participants respond to the SPDP message, following which all start exchanging SEDP data.

2.5.2 OpenSplice DDSI2 specifics

2.5.2.1 Translating between OpenSplice and DDSI

Given that DDSI is the DDS interoperability specification, that the mapping between DCPS entities and DDSI entities is straightforward, and that OpenSplice is a full implementation of the DDS specification, one might expect that relationship between OpenSplice and its DDSI implementation, DDSI2, is trivial. Unfortunately, this is not the case, and it does show in a number of areas. A high-level overview such as this paragraph is not the place for the details of these cases, but they will be described in due course.

The root cause of these complexities is a difference in design philosophy between OpenSplice and the more recent DDSI.

DDSI is very strictly a peer-to-peer protocol at the level of individual endpoints, requiring lots of discovery traffic, and (at least when implemented naively) very bad scalability. It is exactly these three problems that OpenSplice was designed to avoid, and it does so successfully with its native RTNetworking service.

Because of this design for scalability and the consequent use of service processes rather than forcing everything into self-contained application processes, there are various ways in which DDSI2 has to translate between the two worlds. For example, queuing and buffering and, consequently, blocking behaviour is subtly different; DDSI2 needs to also perform local discovery of DCPS endpoints to gather enough information for faithfully representing the system in terms of DDSI, it needs to translate between completely different namespaces (native OpenSplice identifiers are very different from the GUIDs used by DDSI), and it needs to work around receiving asynchronous notifications for events one would expect to be synchronous in DDSI.

This guide aims to not only provide guidance in configuring DDSI2, but also help in understanding the trade-offs involved.

2.5.2.2 Federated versus Standalone deployment

As has been described elsewhere in detail, OpenSplice has multiple deployment models selectable in the configuration file (some of these require a license). For DDSI2, there is no difference between the various models: it simply serves whatever DCPS participants are in the same 'instance', whether that instance be a federation of processes on a single node, all attached to a shared memory segment managed by a set of OpenSplice service processes on that node, or a standalone one in which a single process incorporates the OpenSplice services as libraries.

This document ignores the various deployment modes, using the terminology associated with the federated deployment mode because that mode is the driving force behind several of the user-visible design decisions in DDSI2. In consequence, for a standalone deployment, the term 'node' as used in this guide refers to a single process.

2.5.2.3 Discovery behaviour

2.5.2.3.1 Local discovery and built-in topics

Inside one node, DDSI2 monitors the creation and deletion of local DCPS domain participants, data readers and data writers. It relies on the DCPS built-in topics to keep track of these events, and hence the use of DDSI requires that these topics are enabled in the configuration, which is the default (see the description of `//OpenSplice/Domain/BuiltinTopics[@enabled]` in Section *4.2.12.1* on page 115).

If the built-in topics must be disabled to reduce network load, then the alternative is to instruct DDSI2 to completely ignore them using the DCPS topic/partition to network partition mapping available in the enhanced version, DDSI2E.

A separate issue is that of the DCPS built-in topics when interoperating with other implementations. In OpenSplice the built-in topics are first-class topics, *i.e.*, the only difference between application topics and the built-in topics in OpenSplice is that the built-in topics are pre-defined and that they are published and used by the OpenSplice services. This in turn allows the RTNetworking service to avoid discovery of individual domain participants and endpoints, enabling its excellent scalability.

Conversely, DDSI defines a different and slightly extended representation for the information in the built-in topics as part of the discovery protocol specification, with a clear intent to locally reconstruct the samples of the built-in topics. Unfortunately, this also means that the DCPS built-in topics become a special case.

Taken together, DDSI2 is in the unfortunate situation of having to straddle two very different approaches. While local reconstruction of the DCPS built-in topics by DDSI2 is clearly possible, it would negatively impact the handling of transient data. Since handling transient data is one of the true strengths of OpenSplice, DDSI2 currently does not perform this reconstruction, with the unfortunate implication that loss of liveliness will not be handled fully when interoperating with another DDSI implementation.

2.5.2.3.2 Proxy participants and endpoints

DDSI2 is what the DDSI specification calls a ‘stateful’ implementation. Writers only send data to discovered readers and readers only accept data from discovered writers. (There is one exception: the writer may choose to multicast the data, and anyone listening will be able to receive it, if a reader has already discovered the writer but not vice-versa, it may accept the data even though the connection is not fully established yet.) Consequently, for each remote participant and reader or writer, DDSI2 internally creates a proxy participant, proxy reader or proxy writer.

In the discovery process, writers are matched with proxy readers, and readers are matched with proxy writers, based on the topic and type names and the QoS settings.

Proxies have the same natural hierarchy that ‘normal’ DDSI entities have: each proxy endpoint is owned by some proxy participant, and once the proxy participant is deleted, all its proxy endpoints are deleted as well. Participants assert their liveliness periodically, and when nothing has been heard from a participant for the lease duration published by that participant in its SPDP message, the lease becomes expired triggering a clean-up.

Under normal circumstances, deleting endpoints simply triggers disposes and unregisters in SEDP protocol, and, similarly, deleting a participant also creates special messages that allow the peers to immediately reclaim resources instead of waiting for the lease to expire.

2.5.2.3.3 Sharing of discovery information

DDSI2 is designed to service any number of participants, as one would expect for a service capable of being deployed in a federated system. This obviously means it is aware of all participants, readers and writers on a node. It also means that the discovery protocol as sketched earlier is rather wasteful: there is no need for each individual participant serviced by DDSI2 to run the full discovery protocol for itself.

Instead of implementing the protocol as suggested by the standard, DDSI2 shares all discovery activities amongst the participants, allowing one to add participants on a node with only a minimal impact on the system. It is even possible to have only a single DDSI participant on each node, which then becomes the virtual owner of all the endpoints serviced by that instance of DDSI2. (See Section 2.5.3.2, *Combining multiple participants*, on page 50, and Section *4.7.1.1.7.9* on page 320 for the Internal/SquashParticipants setting.) In this latter mode, there is no discovery penalty at all for having many participants, but evidently, any participant-based liveliness monitoring will be affected.

Because other implementations of the DDSI specification may be written on the assumption that all participants perform their own discovery, it is possible to simulate that with DDSI2. It will not actually perform the discovery for each participant independently, but it will generate the network traffic *as if* it does (see the descriptions of Internal elements in Section 4.7.1.1.7.11, *Element BuiltinEndpointSet*, on page 321, and Section 4.7.1.1.7.13, *Element ConservativeBuiltinReaderStartup*, on page 322; however, please note that at the time of writing, we are not aware of any DDSI implementation requiring the use of these settings.)

By sharing the discovery information across all participants in a single node, each new participant or endpoint is immediately aware of the existing peers and will immediately try to communicate with these peers. This may generate some redundant network traffic if these peers take a significant amount of time for discovering this new participant or endpoint.

Another advantage (particularly in a federated deployment) is that the amount of memory required for discovery and the state of the remote entities is independent of the number of participants that exist locally.

2.5.2.3.4 Lingering writers

When an application deletes a reliable DCPS data writer, there is no guarantee that all its readers have already acknowledged the correct receipt of all samples. In such a case, DDSI2 lets the writer (and the owning participant if necessary) linger in the system for some time, controlled by the Internal/WriterLingerDuration option (Section *4.7.1.1.7.20* on page 326). The writer is deleted when all samples have been acknowledged by all readers or the linger duration has elapsed, whichever comes first.

The writer linger duration setting is currently not applied when DDSI2 is requested to terminate. In a federated deployment it is unlikely to visibly affect system behaviour, but in a standalone deployment data written just before terminating the application may be lost.

2.5.2.3.5 Start-up mode

A similar issue exists when starting DDSI2: DDSI discovery takes time, and when data is written immediately after DDSI2 has started, it is likely that the discovery process hasn’t completed yet and some remote readers have not yet been discovered. This would cause the writers to throw away samples for lack of interest, even though matching readers already existed at the time of starting. For best-effort writers, this is perhaps surprising but still acceptable; for reliable writers, however, it would be very counter-intuitive.

Hence the existence of the so-called ‘start-up mode’, during which all volatile reliable writers are treated as-if they are transient-local writers. Transient-local data is meant to ensure samples are available to late-joining readers, the start-up mode uses this same mechanism to ensure late-discovered readers will also receive the data. This treatment of volatile data as-if it were transient-local happens entirely within DDSI2 and is invisible to the outside world, other than the availability of some samples that would not otherwise be available.

Once DDSI2 has completed its initial discovery, it has built up its view of the network and can locally match new writers against already existing readers, and consequently keeps any new samples published in a writer history cache because these existing readers have not acknowledged them yet. Hence why this mode is tied to the start-up of the DDSI2 service, rather than to that of an individual writer.

Unfortunately it is impossible to detect with certainty when the initial discovery process has been completed and therefore the time DDSI2 remains in this start-up mode is controlled by an option: General/StartupModeDuration.

While in general, this start-up mode is beneficial, it is not always so. There are two downsides: the first is that during the start-up period, the writer history caches can grow significantly larger than one would normally expect; the second is that it does mean large amounts of historical data may be transferred to readers discovered relatively late in the process.

In a federated deployment on a local-area network, the likelihood of this behaviour causing problems is negligible, as in such a configuration the DDSI2 service typically starts seconds before the applications and besides the discovery times are short. The other extreme is a single-process deployment in a wide-area network, where the application starts immediately and discovery times may be long.

2.5.2.4 Writer history QoS and throttling

The DDSI specification heavily relies on the notion of a writer history cache (WHC) within which a sequence number uniquely identifies each sample. The original OpenSplice design has a different division of responsibilities between various components than what is assumed by the DDSI specification and this includes the WHC. Despite the different division, the resulting behaviour is the same.

DDSI2 bridges this divide by constructing its own WHC when needed. This WHC integrates two different indices on the samples published by a writer: one is on sequence number, which is used for retransmitting lost samples, and one is on key value and is used for retaining the current state of each instance in the WHC.

The index on key value allows dropping samples from the index on sequence number when the state of an instance is overwritten by a new sample. For transient-local, it conversely (also) allows retaining the current state of each instance even when all readers have acknowledged a sample.

The index on sequence number is required for retransmitting old data, and is therefore needed for all reliable writers. The index on key values is always needed for transient-local data, and can optionally be used for other writers using a history setting of KEEP_LAST with depth 1. (The Internal/AggressiveKeepLast1Whc setting controls this behaviour—see Section *4.7.1.1.7.12* on page 322.) The advantage of an index on key value in such a case is that superseded samples can be dropped aggressively, instead of having to deliver them to all readers; the disadvantage is that it is somewhat more resource-intensive.

Writer throttling is based on the WHC size using a simple bang-bang controller. Once the WHC contains `Internal/Watermarks/WhcHigh` unacknowledged samples (see Section *4.7.1.1.7.34.2* on page 333), it stalls the writer until the number of unacknowledged samples drops below Internal/Watermarks/WhcLow (see Section *4.7.1.1.7.34.1* on page 333). While ideally only the one writer would be stalled, the interface between the OpenSplice kernel and DDSI2 is such that other outgoing traffic may be stalled as well. See Section 2.5.2.5, *Unresponsive readers & head-of-stream blocking*, on page 45.

2.5.2.5 Unresponsive readers & head-of-stream blocking

For reliable communications, DDSI2 must retain sent samples in the WHC until they have been acknowledged. Especially in case of a KEEP_ALL history kind, but also in the default case when the WHC is not aggressively dropping old samples of instances (Internal/AggressiveKeepLast1Whc, see Section *4.7.1.1.7.12* on page 322), a reader that fails to acknowledge the samples timely will cause the WHC to run into resource limits.

The correct treatment suggested by the DDS specification is to simply take the writer history QoS setting, apply this to the DDSI2 WHC, and block the writer up to its 'max_blocking_time' QoS setting. However, the scalable architecture of OpenSplice renders this simple approach infeasible because of the division of labour between the application processes and the various services. Of course, even if it were a possible approach, the problem would still not be gone entirely, as one unresponsive (for whatever reason) reader would still be able to prevent the writer from making progress and thus prevent the system from making progress if the writer is a critical one.

Because of this, once DDSI2 hits a resource limit on a WHC, it blocks the sequence of outgoing samples for up to Internal/ResponsivenessTimeout (see Section *4.7.1.1.7.2* on page 317). If this timeout is set larger than roughly the domain expiry time (//OpenSplice/Domain/Lease/ExpiryTime, see Section *4.2.5.1* on page 98), it may cause entire nodes to lose liveliness. The enhanced version, DDSI2E, has the ability to use multiple queues and can avoid this problem; please refer to Section 2.5.5.2, *Channel configuration*, on page 59.

Any readers that fail to acknowledge samples in time will be marked 'unresponsive' and be treated as best-effort readers until they start acknowledging data again.

One particular case where this can easily occur is if a reader becomes unreachable, for example because a network cable is unplugged. While this will eventually cause a lease to expire, allowing the proxy reader to be removed and the writer to no longer retain data for it, in the meantime the writer can easily run into a WHC limit. This will then cause the writer to mark the reader as unresponsive, and the system will continue to operate. The presence of unacknowledged data in a WHC as well as the existence of unresponsive readers will force the publication of Heartbeats, and so unplugging a network cable will typically induce a stream of Heartbeats from some writers.

Another case where this can occur is with a very fast writer, and a reader on a slow host, and with large buffers on both sides: then the time needed by the receiving host to process the backlog can become longer than this responsiveness timeout, causing the writer to mark the reader as unresponsive, in turn causing the backlog to be dropped. This allows the reader catch up, at which point it once again acknowledges data promptly and will be considered responsive again, causing a new backlog to build up, &c.

2.5.2.6 Handling of multiple partitions and wildcards

2.5.2.6.1 Publishing in multiple partitions

A variety of design choices allow OpenSplice in combination with its RTNetworking service to be fully dynamically discovered, yet without requiring an expensive discovery protocol. A side effect of these choices is that a DCPS writer publishing a single sample in multiple partitions simultaneously will be translated by the current version of DDSI2 as a writer publishing multiple identical samples in all these partitions, but with unique sequence numbers.

When DDSI2 is used to communicate between OpenSplice nodes, this is not an application-visible issue, but it is visible when interoperating with other implementations. Fortunately, publishing in multiple partitions is rarely a wise choice in a system design.

Note that this only concerns publishing in multiple partitions, subscribing in multiple partitions works exactly as expected, and is also a far more common system design choice.

2.5.2.6.2 Wildcard partitions

DDSI2 fully implements publishing and subscribing using partition wildcards, but depending on many (deployment time and application design) details, the use of partition wildcards for publishing data can easily lead to the replication of data as mentioned in the preceding subsection (*2.5.2.6.1*).

Secondly, because DDSI2 implements transient-local data internally in a different way from the way the OpenSplice durability service does, it is strongly recommended that the combination of transient-local data and publishing using partition wildcards be avoided completely..

2.5.3 Network and discovery configuration

2.5.3.1 Networking interfaces

DDSI2 uses a single network interface, the ‘preferred’ interface, for transmitting its multicast packets and advertises only the address corresponding to this interface in the DDSI discovery protocol.

To determine the default network interface, DDSI2 ranks the eligible interfaces by quality, and then selects the interface with the highest quality. If multiple interfaces are of the highest quality, it will select the first enumerated one. Eligible interfaces are those that are up and have the right kind of address family (IPv4 or IPv6). Priority is then determined as follows:

- interfaces with a non-link-local address are preferred over those with a link-local one;
- multicast-capable is preferred, or if none is available
- non-multicast capable but neither point-to-point, or if none is available
- point-to-point, or if none is available
- loopback

If this procedure doesn't select the desired interface automatically, it can be overridden by setting General/NetworkInterfaceAddress to either the name of the interface, the IP address of the host on the desired interface, or the network portion of the IP address of the host on the desired interface. An exact match on the address is always preferred and is the only option that allows selecting the desired one when multiple addresses are tied to a single interface.

The default address family is IPv4, setting General/UseIPv6 will change this to IPv6. Currently, DDSI2 does not mix IPv4 and IPv6 addressing. Consequently, all DDSI participants in the network must use the same addressing mode. When interoperating, this behaviour is the same, *i.e.*, it will look at either IPv4 or IPv6 addresses in the advertised address information in the SPDP and SEDP discovery protocols.

IPv6 link-local addresses are considered undesirable because they need to be published and received *via* the discovery mechanism, but there is in general no way to determine to which interface a received link-local address is related.

If IPv6 is requested and the preferred interface has a non-link-local address, DDSI2 will operate in a 'global addressing' mode and will only consider discovered non-link-local addresses. In this mode, one can select any set of interface for listening to multicasts. Note that this behaviour is essentially identical to that when using IPv4, as IPv4 does not have the formal notion of address scopes that IPv6 has.

If instead only a link-local address is available, DDSI2 will run in a 'link-local addressing' mode. In this mode it will accept any address in a discovery packet, assuming that a link-local address is valid on the preferred interface. To minimise the risk involved in this assumption, it only allows the preferred interface for listening to multicasts.

When a remote participant publishes multiple addresses in its SPDP message (or in SEDP messages, for that matter), it will select a single address to use for communicating with that participant. The address chosen is the first eligible one on the same network as the locally chosen interface, else one that is on a network corresponding to any of the other local interfaces, and finally simply the first one. Eligibility is determined in the same way as for network interfaces.

2.5.3.1.1 Multicasting

DDSI2 allows configuring to what extent multicast is to be used:

- whether to use multicast for data communications,
- whether to use multicast for participant discovery,
- on which interfaces to listen for multicasts.

It is advised to allow multicasting to be used. However, if there are restrictions on the use of multicasting, or if the network reliability is dramatically different for multicast than for unicast, it may be attractive to disable multicast for normal communications. In this case, setting General/AllowMulticast (see Section *4.7.1.1.9.2* on page 336) to false will force DDSI2 to use unicast communications for everything except the periodic distribution of the participant discovery messages.

If at all possible, it is strongly advised to leave multicast-based participant discovery enabled, because that avoids having to specify a list of nodes to contact, and it furthermore reduces the network load considerably. However, if need be, one can disable the participant discovery from sending multicasts by setting Internal/SuppressSpdpMulticast to true (see Section *4.7.1.1.7.18* on page 325).

To disable incoming multicasts, or to control from which interfaces multicasts are to be accepted, one can use the General/MulticastRecvInterfaceAddresses setting (see Section *4.7.1.1.9.11* on page 340). This allows listening on no interface, the preferred, all or a specific set of interfaces.

2.5.3.1.2 Discovery configuration

2.5.3.1.3 Discovery addresses

The DDSI discovery protocols, SPDP for the domain participants and SEDP for their endpoints, usually operate well without any explicit configuration. Indeed, the SEDP protocol never requires any configuration.

DDSI2 by default uses the domain id as specified in //OpenSplice/Domain/Id (see Section *4.2.1* on page 95), but allows overriding it for special configurations using the Discovery/DomainId setting. The domain id is the basis for all UDP/IP port number calculations, which can be tweaked when necessary using the configuration settings under Discovery/Ports. It is however rarely necessary to change the standardised defaults.

The SPDP protocol periodically sends, for each domain participant, an SPDP sample to a set of addresses, which by default contains just the multicast address, which is standardised for IPv4 (`239.255.0.1`), but not for IPv6 (it uses `ff02::ffff:239.255.0.1`). The actual address can be overridden using the Discovery/SPDPMulticastAddress setting (see Section *4.7.1.1.5.1* on page 305), which requires a valid multicast address.

In addition (or as an alternative) to the multicast-based discovery, any number of unicast addresses can be configured as addresses to be contacted by specifying peers in the Discovery/Peers section (see Section *4.7.1.1.5.6* on page 311). Each time an SPDP message is sent, it is sent to all of these addresses.

Default behaviour of DDSI2 is to include each IP address several times in the set, each time with a different UDP port number (corresponding to another participant index), allowing at least several applications to be present on these hosts.

Obviously, configuring a number of peers in this way causes a large burst of packets to be sent each time an SPDP message is sent out, and each local DDSI participant causes a burst of its own. Most of the participant indices will not actually be use, making this rather wasteful behaviour.

DDSI2 allows explicitly adding a port number to the IP address, formatted as IP:PORT, to avoid this waste, but this requires manually calculating the port number. In practice it also requires fixing the participant index using Discovery/ParticipantIndex (see the description of ‘PI’ in Section 2.5.3.3, *Controlling port numbers*, starting on page 51) to ensure that the configured port number indeed corresponds to the remote DDSI2 (or other DDSI implementation), and therefore is really practicable only in a federated deployment.

2.5.3.1.4 Asymmetrical discovery

On reception of an SPDP packet, DDSI2 adds the addresses advertised in that SPDP packet to this set, allowing asymmetrical discovery. In an extreme example, if SPDP multicasting is disabled entirely, host A has the address of host B in its peer list and host B has an empty peer list, then B will eventually discover A because of an SPDP message sent by A, at which point it adds A’s address to its own set and starts sending its own SPDP message to A, allowing A to discover B. This takes a bit longer than normal multicast based discovery, though.

2.5.3.1.5 Timing of SPDP packets

The interval with which the SPDP packets are transmitted is configurable as well, using the Discovery/SPDPInterval setting. A longer interval reduces the network load, but also increases the time discovery takes, especially in the face of temporary network disconnections.

2.5.3.1.6 Endpoint discovery

Although the SEDP protocol never requires any configuration, the network partitioning of OpenSplice DDSI2E does interact with it: so-called ‘ignored partitions’ can be used to instruct DDSI2 to completely ignore certain DCPS topic and partition combinations, which will prevent DDSI2 from forwarding data for these topic/partition combinations to and from the network.

While it is rarely necessary, it is worth mentioning that by overriding the domain id used by DDSI in conjunction with ignored partitions and unique SPDP multicast addresses allows partitioning the data and giving each partition its own instance of DDSI2.

2.5.3.2 Combining multiple participants

In an OpenSplice standalone deployment the various configured services, such as spliced and DDSI2, still retain their identity by creating their own DCPS domain participants. DDSI2 faithfully mirrors all these participants in DDSI, and it will appear at the DDSI level as if there is a large system with many participants, whereas in reality there are only a few application participants.

The Internal/SquashParticipants option (see Section *4.7.1.1.7.9* on page 320) can be used to simulate the existence of only one participant, the DDSI2 service itself, which owns all endpoints on that node. This reduces the background messages because far fewer liveliness assertions need to be sent.

Clearly, the liveliness monitoring features that are related to domain participants will be affected if multiple DCPS domain participants are combined into a single DDSI domain participant. The OpenSplice services all use a liveliness QoS setting of AUTOMATIC, which works fine.

In a federated deployment, the effect of this option is to have only a single DDSI domain participant per node. This is of course much more scalable, but in no way resembles the actual structure of the system if there are in fact multiple application processes running on that node.

However, in OpenSplice the built-in topics are not derived from the DDSI discovery, and hence in an OpenSplice-only network the use of the Internal/SquashParticipants setting (see Section *4.7.1.1.7.9* on page 320) will not result in any loss of information in the DCPS API or in the OpenSplice tools such as the Tester.

When interoperability with another vendor is not needed, enabling the SquashParticipants option is often a good choice.

2.5.3.3 Controlling port numbers

The port numbers used by DDSI2 are determined as follows, where the first two items are given by the DDSI specification and the third is unique to DDSI2 as a way of serving multiple participants by a single DDSI instance:

- 2 ‘well-known’ multicast ports: B and B+1
- 2 unicast ports at which only this instance of DDSI2 is listening: B+PG*PI+10 and B+PG*PI+11
- 1 unicast port per domain participant it serves, chosen by the kernel from the anonymous ports, *i.e.*, >= 32768

where:

- B is Discovery/Ports/Base (7400) + Discovery/Ports/DomainGain (250) * Domain/Id
- PG is Discovery/Ports/ParticipantGain (2)
- PI is Discovery/ParticipantIndex

The default values, taken from the DDSI specification, are in parentheses. There are actually even more parameters, here simply turned into constants as there is absolutely no point in ever changing these values—but they are configurable and the interested reader is referred to the DDSI 2.1 specification, section 9.6.1.

PI is the most interesting, as it relates to having multiple instances of DDSI2 in the same domain on a single node. In a federated deployment, this never happens (exceptional cases excluded). Its configured value is either ‘auto’, ‘none’ or a non-negative integer. This setting matters:

- When it is ‘auto’ (which is the default), DDSI2 probes UDP port numbers on start-up, starting with PI = 0, incrementing it by one each time until it finds a pair of available port numbers, or it hits the limit. The maximum PI it will ever choose is currently still hard-coded at 9 as a way of limiting the cost of unicast discovery. (It is recognised that this limit can cause issues in a standalone deployment.)
- When it is ‘none’ it simply ignores the ‘participant index’ altogether and asks the kernel to pick two random ports (>= 32768). This eliminates the limit on the number of standalone deployments on a single machine and works just fine with multicast discovery while complying with all other parts of the specification for interoperability. However, it is incompatible with unicast discovery.
- When it is a non-negative integer, it is simply the value of PI in the above calculations. If multiple instances of DDSI2 on a single machine are needed, they will need unique values for PI, and so for standalone deployments this particular alternative is hardly useful.

Clearly, to fully control port numbers, setting Discovery/ParticipantIndex (= PI) to a hard-coded value is the only possibility. In a federated deployment this is an option that has very few downsides, and generally 0 will be a good choice.

By fixing PI, the port numbers needed for unicast discovery are fixed as well. This allows listing peers as IP:PORT pairs, significantly reducing traffic, as explained in the preceding subsection.

The other non-fixed ports that are used are the per-domain participant ports, the third item in the list. These are used only because there exist some DDSI implementations that assume each domain participant advertises a unique port number as part of the discovery protocol, and hence that there is never any need for including an explicit destination participant id when intending to address a single domain participant by using its unicast locator. DDSI2 never makes this assumption, instead opting to send a few bytes extra to ensure the contents of a message are all that is needed. With other implementations, you will need to check.

If all DDSI implementations in the network include full addressing information in the messages, like DDSI2, then the per-domain participant ports serve no purpose at all. The default `false` setting of Compatibility/ManySocketsMode disables the creation of these ports (see Section 4.7.1.1.4.7, *Element ManySocketsMode*, on page 304).

This setting has a few other side benefits as well, as there will generally be more participants using the same unicast locator, improving the chances for requiring but a single unicast even when addressing a multiple participants in a node. The obvious case where this is beneficial is when one host has not received a multicast.

2.5.3.4 Coexistence with OpenSplice RTNetworking

DDSI2 has a special mode, configured using General/CoexistWithNativeNetworking, to allow it to operate in conjunction with OpenSplice RTNetworking: in this mode DDSI2 only handles packets sent by other vendors' implementations, allowing all intra-OpenSplice traffic to be handled by the RTNetworking service while still providing interoperability with other vendors.

2.5.4 Data path configuration

2.5.4.1 Data path architecture

The data path in DDSI2 consists of a transmitting and a receiving side. The main path in the transmit side accepts data to be transmitted from the OpenSplice kernel *via* a network queue and administrates and formats the data for transmission over the network.

The secondary path handles asynchronous events such as the periodic generation of writer Heartbeats and the transmitting of acknowledgement messages from readers to writers, in addition to handling the retransmission of old data on request. These requests can originate in packet loss, but also in requests for historical data from transient-local readers.

Figure 5 overleaf gives an overview of the main data flow and the threads in a configuration using two channels. Configuring multiple channels is an enhanced feature that is available only in DDSI2E, but the principle is the same in both variants.

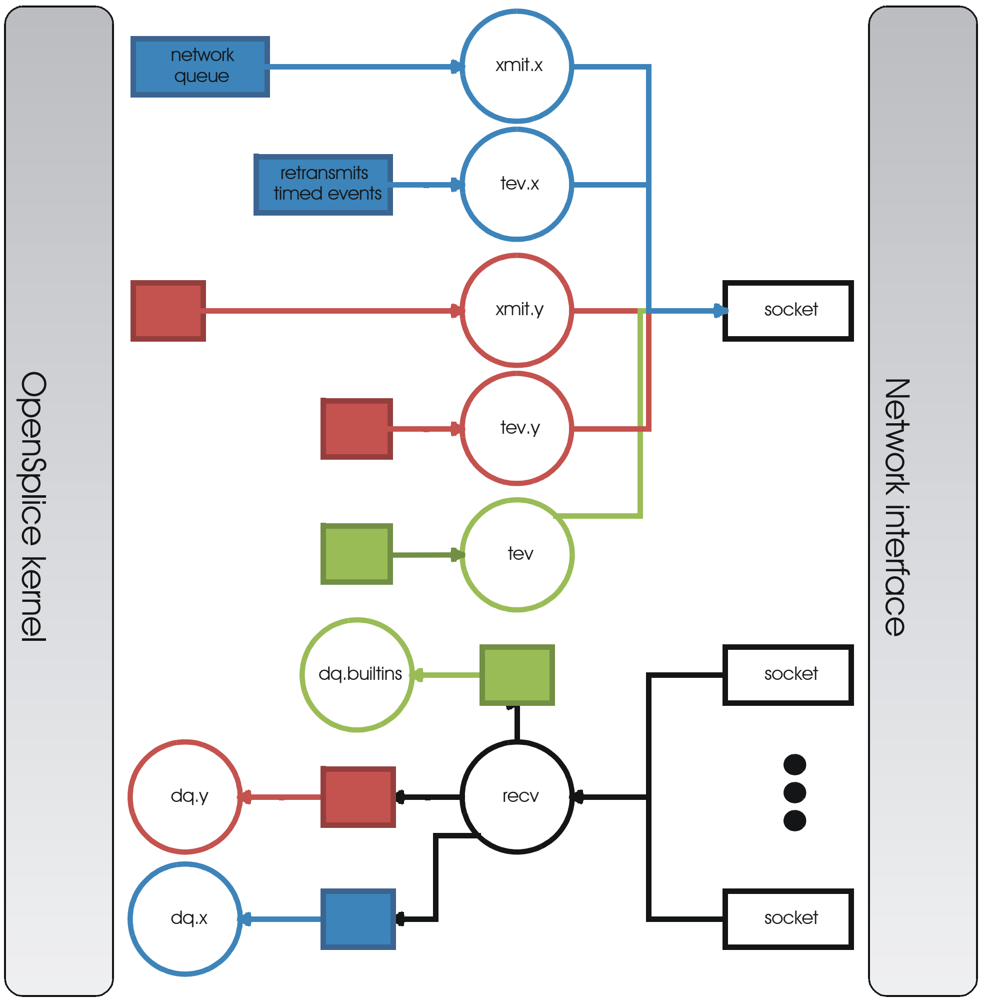


Figure 5 Data flow using two channels

2.5.4.2 Transmit-side configuration

2.5.4.2.1 Transmit processing

DDSI2E divides the outgoing data stream into prioritised channels. These channels are handled completely independently, effectively allowing mapping DDS transport priorities to operating system thread priorities. Although the ability to define multiple channels is limited to DDSI2E, DDSI2 uses the same mechanisms but is restricted to what in DDSI2E is the default channel if none are configured explicitly. For details on configuring channels, see Section 2.5.5.2, *Channel configuration*, on page 59.

Each channel has its own transmit thread, draining a queue with samples to be transmitted from the OpenSplice kernel. The maximum size of the queue can be configured per channel, and the default for the individual channels is configured using the Sizing/NetworkQueueSize setting. In DDSI2, this setting simply controls the queue size, as the default channel of DDSI2E has the default queue size. A larger queue size increases the potential latency and (shared) memory requirements, but improves the possibilities for smoothing out traffic if the applications publish it in bursts.

Once a networking service has taken a sample from the queue, it takes responsibility for it. Consequently, if it is to be sent reliably and there are insufficient resources to store it in the WHC, it must wait for resources to become available. See Section 2.5.2.5, *Unresponsive readers & head-of-stream blocking*, on page 45.

The DDSI control messages (Heartbeat, AckNack, &c.) are sent by a thread dedicated to handling timed events and asynchronous transmissions, including retransmissions of samples on request of a reader. This thread is known as the ‘timed-event thread’ and there is at least one such thread, but each channel can have its own one.

DDSI2E can also perform traffic shaping and bandwidth limiting, configurable per channel, and with independent limits for data on the one hand and control and retransmissions on the other hand.

2.5.4.2.2 Retransmit merging

A remote reader can request retransmissions whenever it receives a Heartbeat and detects samples are missing. If a sample was lost on the network for many or all readers, the next heartbeat is likely to trigger a ‘storm’ of retransmission requests. Thus, the writer should attempt merging these requests into a multicast retransmission, to avoid retransmitting the same sample over & over again to many different readers. Similarly, while readers should try to avoid requesting retransmissions too often, in an interoperable system the writers should be robust against it.

In DDSI2, upon receiving a Heartbeat that indicates samples are missing, a reader will schedule a retransmission request to be sent after Internal/NackDelay (see Section *4.7.1.1.7.26* on page 329), or combine it with an already scheduled request if possible. Any samples received in between receipt of the Heartbeat and the sending of the AckNack will not need to be retransmitted.

Secondly, a writer attempts to combine retransmit requests in two different ways. The first is to change messages from unicast to multicast when another retransmit request arrives while the retransmit has not yet taken place. This is particularly effective when bandwidth limiting causes a backlog of samples to be retransmitted.

The behaviour of the second can be configured using the Internal/RetransmitMerging setting (see Section *4.7.1.1.7.4* on page 318). Based on this setting, a retransmit request for a sample is either honoured unconditionally, or it may be suppressed (or 'merged') if it comes in shortly after a multicasted retransmission of that very sample, on the assumption that the second reader will likely receive the retransmit, too. The Internal/RetransmitMergingPeriod (see Section *4.7.1.1.7.7* on page 319) controls the length of this time window.

2.5.4.2.3 Retransmit backlogs

Another issue is that a reader can request retransmission of many samples at once. When the writer simply queues all these samples for retransmission, it may well result in a huge backlog of samples to be retransmitted. As a result, the ones near the end of the queue may be delayed by so much that the reader issues another retransmit request. DDSI2E provides bandwidth limiting, which makes the situation even worse, as it can significantly increase the time it takes for a sample to be sent out once it has been queued for retransmission.

Therefore, DDSI2 limits the number of samples queued for retransmission and ignores (those parts of) retransmission requests that would cause the retransmit queue to contain too many samples or take too much time to process. There are two settings governing the size of these queues, and the limits are applied per timed-event thread (*i.e.*, the global one, and typically one for each configured channel with limited bandwidth when using DDSI2E). The first is Internal/MaxQueuedRexmitMessages (see Section *4.7.1.1.7.17* on page 324), which limits the number of, retransmit messages, the second Internal/MaxQueuedRexmitBytes (see Section *4.7.1.1.7.14* on page 323) which limits the number of bytes. The latter is automatically set based on the combination of the allowed transmit bandwidth and the Internal/NackDelay (see Section *4.7.1.1.7.26* on page 329) setting, as an approximation of the likely time until the next potential retransmit request from the reader.

2.5.4.2.4 Controlling fragmentation

Samples in DDS can be arbitrarily large, and will not always fit within a single datagram. DDSI has facilities to fragment samples so they can fit in UDP datagrams, and similarly IP has facilities to fragment UDP datagrams to into network packets. The DDSI specification states that one must not unnecessarily fragment at the DDSI level, but DDSI2 simply provides a fully configurable behaviour.

If the serialised form of a sample is at least Internal/FragmentSize (see Section *4.7.1.1.7.28* on page 329), it will be fragmented using the DDSI fragmentation. All but the last fragment will be exactly this size; the last one may be smaller.

Control messages, non-fragmented samples, and sample fragments are all subject to packing into datagrams before sending it out on the network, based on various attributes such as the destination address, to reduce the number of network packets.

This packing allows datagram payloads of up to Internal/MaxMessageSize (see Section *4.7.1.1.7.25* on page 328), overshooting this size if the set maximum is too small to contain what must be sent as a single unit. Note that in this case, there is a real problem anyway, and it no longer matters where the data is rejected, if it is rejected at all. UDP/IP header sizes are not taken into account in this maximum message size.

The IP layer then takes this UDP datagram, possibly fragmenting it into multiple packets to stay within the maximum size the underlying network supports. A trade-off to be made is that while DDSI fragments can be retransmitted individually, the processing overhead of DDSI fragmentation is larger than that of UDP fragmentation.

2.5.4.3 Receive-side configuration

2.5.4.3.1 Receive processing

Receiving of data is split into multiple threads, as also depicted in the overall DDSI2 data path diagram above:

- A single receive thread responsible for retrieving network packets and running the protocol state machine;
- A delivery thread dedicated to processing DDSI built-in data: participant discovery, endpoint discovery and liveliness assertions;
- One or more delivery threads dedicated to the handling of application data: deserialisation and delivery to the DCPS data reader caches.

The receive thread is responsible for retrieving all incoming network packets, running the protocol state machine, which involves scheduling of AckNack and Heartbeat messages and queueing of samples that must be retransmitted, and for defragmenting and ordering incoming samples.

For a specific proxy writer—the local manifestation of a remote DDSI data writer—with a number of data readers, the organisation is as shown in *Figure 6* overleaf:

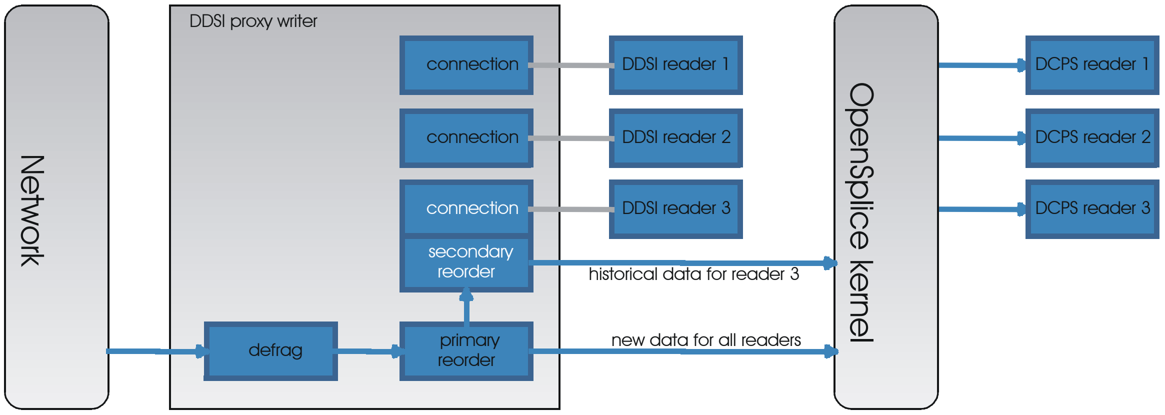


Figure 6 Proxy writer with multiple data readers

Fragmented data first enters the defragmentation stage, which is per proxy writer. The number of samples that can be defragmented simultaneously is limited, for reliable data to Internal/DefragReliableMaxSamples (see Section *4.7.1.1.7.10* on page 321) and for unreliable data to Internal/DefragUnreliableMaxSamples (see Section *4.7.1.1.7.8* on page 320).

Samples (defragmented if necessary) received out of sequence are buffered, primarily per proxy writer, but, secondarily, per reader catching up on historical (transient-local) data. The size of the first is limited to Internal/PrimaryReorderMaxSamples (see Section *4.7.1.1.7.3* on page 317), the size of the second to Internal/SecondaryReorderMaxSamples (see Section *4.7.1.1.7.5* on page 318).

In between the receive thread and the delivery threads sit queues, of which the maximum size is controlled by the Internal/DeliveryQueueMaxSamples (see Section *4.7.1.1.7.31* on page 331) setting. Generally there is no need for these queues to be very large, their primary function is to smooth out the processing when batches of samples become available at once, for example following a retransmission.

When any of these receive buffers hit their size limit, DDSI2 will drop incoming (fragments of) samples and/or buffered (fragments of) samples to ensure the receive thread can continue to make progress. Such dropped samples will eventually be retransmitted.

2.5.4.3.2 Minimising receive latency

In low-latency environments, a few microseconds can be gained by processing the application data directly in the receive thread, or synchronously with respect to the incoming network traffic, instead of queueing it for asynchronous processing by a delivery thread. This happens for data transmitted with the max_latency QoS setting at most a configurable value and the transport_priority QoS setting at least a configurable value. By default, these values are `0` and the maximum transport priority, effectively disabling synchronous delivery for all but the most important and urgent data. See the Internal/SynchronousDeliveryLatencyBound and Internal/SynchronousDeliveryPriorityThreshold settings (see Secion *4.7.1.1.7.24* on page 327 and Section *4.7.1.1.7.23* on page 327).

2.5.4.4 Direction-independent settings

2.5.4.4.1 Maximum sample size

DDSI2 provides a setting, `Internal/MaxSampleSize` (see Section 4.7.1.1.7.32, *Element MaxSampleSize*, on page 331), to control the maximum size of samples that the service is willing to process. The size is the size of the (CDR) serialised payload, and the limit holds both for built-in data and for application data. The (CDR) serialised payload is never larger than the in-memory representation of the data.

On the transmitting side, samples larger than `MaxSampleSize` are dropped with a warning in the OpenSplice info log. DDSI2 behaves as-if the sample never existed. The current structure of the interface between the OpenSplice kernel and the OpenSplice networking services unfortunately prevents DDSI2 from properly reporting this back to the application that wrote the sample, so the only guaranteed way of detecting the dropping of the sample is by checking the info log.

Similarly, on the receiving side, samples large than `MaxSampleSize` are dropped, and this is done as early as possible, immediately following the reception of a sample or fragment of one, to prevent any resources from being claimed for longer than strictly necessary. Where the transmitting side completely ignores the sample, on the receiving side DDSI2 pretends the sample has been correctly received and, at the DDSI2 level, acknowledges reception to the writer when asked. This allows communication to continue.

When the receiving side drops a sample, readers will get a ‘sample lost’ notification at the next sample that does get delivered to those readers. This condition means that again checking the info log is ultimately the only truly reliable way of determining whether samples have been dropped or not.

While dropping samples (or fragments thereof) as early as possible is beneficial from the point of view of reducing resource usage, it can make it hard to decide whether or not dropping a particular sample has been recorded in the log already. Under normal operational circumstances, DDSI2 will report a single event for each sample dropped, but it may on occasion report multiple events for the same sample.

Finally, it is technically allowed to set `MaxSampleSize` to very small sizes, even to the point that the discovery data can’t be communicated anymore. The dropping of the discovery data will be duly reported, but the usefulness of such a configuration seems doubtful.

2.5.5 DDSI2E Enhanced features

2.5.5.1 Introduction

DDSI2E is an enhanced version of the DDSI2 service, adding three major features:

- Channels: parallel processing of independent data stream, with prioritisation based on the transport priority setting of the data writers, and supporting traffic-shaping of outgoing data;
- Network partitions: use of special multicast addresses for some partition-topic combinations as well as allowing ignoring data; and
- Encryption: encrypting all traffic for a certain network partition.

This section provides details on the configuration of these three features.

2.5.5.2 Channel configuration

2.5.5.2.1 Overview

DDSI2E allows defining channels, which are independent data paths within the DDSI service. OpenSplice chooses a channel based by matching the transport priority QoS setting of the data writer with the threshold specified for the various channels. Because each channel has a set of dedicated threads to perform the processing and the thread priorities can all be configured, it is straightforward to guarantee that samples from high-priority data writers will get precedence over those from low-priority data throughout the service stack.

A second aspect to the use of channels is that the head-of-line blocking mentioned in Section 2.5.2.5, *Unresponsive readers & head-of-stream blocking*, on page 45. Unresponsive readers & head-of-stream blocking is per channel, guaranteeing that a high-priority channel will not be disrupted by an unresponsive reader of low-priority data.

The channel-specific threads perform essentially all processing (serialisation, writer history cache management, deserialisation, delivery to DCPS data readers, &c.), but there still is one shared thread involved. This is the receive thread (‘recv’) that demultiplexes incoming packets and implements the protocol state machine. The receive thread only performs minimal work on each incoming packet, and never has to wait for the processing of user data.

The existence of the receive thread is the only major difference between DDSI2E channels and those of the OpenSplice RTNetworking service: in the RTNetworking service, each thread is truly independent. This change is the consequence of DDSI2E interoperating with implementations that are not aware of channels and with DDSI2E nodes that have differently configured channels, unlike the RTNetworking service where all nodes must use exactly the same channel definitions.

When configuring multiple channels, it is recommended to set the CPU priority of the receive thread to at least that of the threads of the highest priority channel, to ensure the receive thread will be scheduled in promptly.

If no channels are defined explicitly, a single, default channel is used. In DDSI2 (rather than DDSI2E), the processing is as-if only this default channel exists.

2.5.5.2.2 Transmit side

For each discovered local data writer, DDSI2E determines the channel to use. This is the channel with the lowest threshold priority of all channels that have a threshold priority that is higher than the writer’s transport priority. If there is no such channel, *i.e.*, the writer has a transport priority higher than the highest channel threshold, the channel with the highest threshold is used.

Each channel has its own network queue into which the OpenSplice kernel writes samples to be transmitted and that DDSI2E reads. The size of this queue can be set for each channel independently by using Channels/Channel/QueueSize, with the default taken from the global Sizing/NetworkQueueSize.

Bandwidth limiting and traffic shaping are configured per channel as well. The following parameters are configurable:

- bandwidth limit
- auxiliary bandwidth limit
- IP QoS settings

The traffic shaping is based on a leaky bucket algorithm: transmit credits are added at a constant rate, the total transmit credit is capped, and each outgoing packet reduces the available transmit credit. Outgoing packets must wait until enough transmit credits are available.

Each channel has two separate credits: data and auxiliary. The data credit is used strictly for transmitting fresh data (*i.e.*, directly corresponding to writes, disposes, &c.) and control messages directly caused by transmitting that data. This credit is configured using the Channels/Channel/DataBandwidthLimit setting. By default, a channel is treated as if it has infinite data credit, disabling traffic shaping.

The auxiliary credit is used for everything else: asynchronous control data & retransmissions, and is configured using the Channels/Channel/AuxiliaryBandwidthLimit setting (see Section *4.7.2.1.2.1.6* on page 349).

When an auxiliary bandwidth limit has been set explicitly, or when one explicitly sets, *e.g.*, a thread priority for a thread named ‘tev.channel-name’, an independent event thread handles the generation of auxiliary data for that channel. But if neither is given, the global event thread instead handles all auxiliary data for the channel.

The global event thread has an auxiliary credit of its own, configured using Internal/AuxiliaryBandwidthLimit (see Section *4.7.2.1.5.1* on page 357). This credit applies to all discovery related traffic, as well as to all auxiliary data generated by channels without their own event thread.

Generally, it is best to simply specify both the data and the auxiliary bandwidth for each channel separately, and set Internal/AuxiliaryBandwidthLimit (see Section *4.7.2.1.5.1* on page 357) to limit the network bandwidth the discovery & liveliness protocols can consume.

2.5.5.2.3 Receive side

On the receive side, the single receive thread accepts incoming data and runs the protocol state machine. Data ready for delivery to the local data readers is queued on the delivery queue for the channel that best matches the proxy writer that wrote the data, according to the same criterion used for selecting the outgoing channel for the data writer.

The delivery queue is emptied by the delivery thread, ‘dq.channel-name’, which deserialises the data and updates the data readers. Because each channel has its own delivery thread with its own scheduling priority, once the data leaves the receive thread and is enqueued for the delivery thread, higher priority data once again takes precedence over lower priority data.

2.5.5.2.4 Discovery traffic

DDSI discovery data is always transmitted by the global timed-event thread (‘tev’), and always processed by the special delivery thread for DDSI built-in data (‘dq.builtin’). By explicitly creating a timed-event thread, one effectively separates application data from all discovery data. One way of creating such a thread is by setting properties for it (see Section 2.5.6, *Thread configuration*, on page 63), another is by setting a bandwidth limit on the auxiliary data of the channel (see Section *2.5.5.2.2* on page 60 and Section *4.7.2.1.2.1.6* on page 349).

2.5.5.2.5 On interoperability

DDSI2E channels are fully compliant with the wire protocol. One can mix & match DDSI2E with different sets of channels and with other vendors’ implementation.

2.5.5.3 Network partition configuration

2.5.5.3.1 Overview

Network partitions introduce alternative multicast addresses for data. In the DDSI discovery protocol, a reader can override the default address at which it is reachable, and this feature of the discovery protocol is used to advertise alternative multicast addresses. The DDSI writers in the network will (also) multicast to such an alternative multicast address when multicasting samples or control data.

The mapping of a DCPS data reader to a network partition is indirect: DDSI2E first matches the DCPS data reader partitions and topic against a table of ‘partition mappings’, partition/topic combinations to obtain the name of a network partition, then looks up the network partition. This makes it easier to map many different partition/topic combinations to the same multicast address without having to specify the actual multicast address many times over.

If no match is found, DDSI2E automatically defaults to standardised DDSI multicast address.

2.5.5.3.2 Matching rules

Matching of a DCPS partition/topic combination proceeds in the order in which the partition mappings are specified in the configuration. The first matching mapping is the one that will be used. The ‘*’ and ‘?’ wildcards are available for the DCPS partition/topic combination in the partition mapping.

As mentioned earlier (see Section 2.5.2.3.1, *Local discovery and built-in topics*, on page 40), DDSI2E can be instructed to ignore all DCPS data readers and writers for certain DCPS partition/topic combinations through the use of ‘IgnoredPartitions’. The ignored partitions use the same matching rules as normal mappings, and take precedence over the normal mappings.

2.5.5.3.3 Multiple matching mappings

A single DCPS data reader can be associated with a set of partitions, and each partition/topic combination can potentially map to a different network partitions. In this case, DDSI2E will use the first matching network partition. This does not affect what data the reader will receive; it only affects the addressing on the network.

2.5.5.3.4 On interoperability

DDSI2E network partitions are fully compliant with the wire protocol. One can mix & match DDSI2E with different sets of network partitions and with other vendors' implementation.

2.5.5.4 Encryption configuration

2.5.5.4.1 Overview

DDSI2E encryption support allows defining 'security profiles', named combinations of (symmetrical block) ciphers and keys. These can be associated with subsets of the DCPS data writers *via* the network partitions: data from a DCPS data writer matching a particular network partition will be encrypted if that network partition has an associated security profile.

The encrypted data will be tagged with a unique identifier for the network partition, in cleartext. The receiving nodes use this identifier to lookup the network partition & the associated encryption key and cipher.

Clearly, this requires that the definition of the encrypted network partitions must be identical on the transmitting and the receiving sides. If the network partition cannot be found, or if the associated key or cipher differs, the receiver will ignore the encrypted data. It is therefore not necessary to share keys with nodes that have no need for the encrypted data.

The encryption is performed per-packet; there is no chaining from one packet to the next.

2.5.5.4.2 On interoperability

Encryption is not yet a standardised part of DDSI, but the standard does allow vendor-specific extensions. DDSI2E encryption relies on a vendor-specific extension to marshal encrypted data into valid DDSI messages, but they cannot be interpreted by implementations that do not recognise this particular extension.

2.5.6 Thread configuration

DDSI2 creates a number of threads and each of these threads has a number of properties that can be controlled individually. The threads involved in the data path are shown in *Figure 5* on page 53 in Section 2.5.4.1, *Data path architecture*.

The properties that can be controlled are:

- stack size,
- scheduling class, and
- scheduling priority.

The threads are named and the Threads/Thread[@name] (see Section *4.7.1.1.2.1.1* on page 296) attribute is used to set the properties by thread name. Any subset of threads can be given special properties; anything not specified explicitly is left at the default value.

The following threads exist:

- *`gc`*: garbage collector, which sleeps until garbage collection is requested for an entity, at which point it starts monitoring the state of DDSI2, pushing the entity through whatever state transitions are needed once it is safe to do so, ending with the freeing of the memory.
- *`main`*: the main thread of DDSI2, which performs start-up and teardown and monitors the creation and deletion of entities in the local node using the built-in topics.
- *`recv`*: accepts incoming network packets from all sockets/ports, performs all protocol processing, queues (nearly) all protocol messages sent in response for handling by the timed-event thread, queues for delivery or, in special cases, delivers it directly to the data readers.
- *`dq.builtins`*: processes all discovery data coming in from the network.
- *`lease`*: performs internal liveliness monitoring of DDSI2 and renews the OpenSplice kernel lease if the status is satisfactory.
- *`tev`*: timed-event handling, used for all kinds of things, such as: periodic transmission of participant discovery and liveliness messages, transmission of control messages for reliable writers and readers (except those that have their own timed-event thread), retransmitting of reliable data on request (except those that have their own timed-event thread), and handling of start-up mode to normal mode transition.

and, for each defined channel:

- *`xmit.channel-name`*: takes data from the OpenSplice kernel’s queue for this channel, serialises it and forwards it to the network.
- *`dq.channel-name`*: deserialisation and asynchronous delivery of all user data.
- *`tev.channel-name`*: channel-specific ‘timed-event’ handling: transmission of control messages for reliable writers and readers and retransmission of data on request. Channel-specific threads exist only if the configuration includes an element for it or if an auxiliary bandwidth limit is set for the channel.

For DDSI2, and DDSI2E when no channels are explicitly defined, there is one channel named ‘user’.

2.5.7 Reporting and tracing

DDSI2 can produce highly detailed traces of all traffic and internal activities. It allows enabling individual categories of information, as well as having a simple verbosity level that enables fixed sets of categories and of which the definition corresponds to that of the other OpenSplice services.

The categorisation of tracing output is incomplete and hence most of the verbosity levels and categories are not of much use in the current release. This is an ongoing process and here we describe the target situation rather than the current situation.

All ‘fatal’ and ‘error’ messages are written both to the DDSI2 log and to the ospl-error.log file; similarly all ‘warning’ messages are written to the DDSI2 log and the ospl-info.log file.

The Tracing element has the following sub elements:

• Verbosity: selects a tracing level by enabled a pre-defined set of categories. The list below gives the known tracing levels, and the categories they enable:

  - none
  - severe: ‘error’ and ‘fatal’
  - warning, info: severe + ‘warning’
  - config: info + ‘config’
  - fine: config + ‘discovery’
  - finer: fine + ‘traffic’, ‘timing’ and ‘info’
  - finest: fine + ‘trace’

• EnableCategory: a comma-separated list of keywords, each keyword enabling individual categories. The following keywords are recognised:

  - fatal: all fatal errors, errors causing immediate termination
  - error: failures probably impacting correctness but not necessarily causing immediate termination.
  - warning: abnormal situations that will likely not impact correctness.
  - config: full dump of the configuration
  - info: general informational notices
  - discovery: all discovery activity
  - data: include data content of samples in traces
  - radmin: receive buffer administration

- timing: periodic reporting of CPU loads per thread
- traffic: periodic reporting of total outgoing data

In addition, the keyword ‘trace’ enables all but ‘radmin’.

- OutputFile: the file to which to write the DDSI2 log to
- AppendToFile: boolean, set to ‘true’ to append to the log instead of replacing the file.

Currently, the useful verbosity settings are ‘config’ and ‘finest’. ‘Config’ writes the full configuration to the DDSI2 log file as well as any warnings or errors, which can be a good way to verify everything is configured and behaving as expected. ‘Finest’ provides a detailed trace of everything that occurs and is an indispensable source of information when analysing problems; however, it also requires a significant amount of time and results in huge log files.

Whether these logging levels are set using the verbosity level or by enabling the corresponding categories is immaterial.

2.5.8 Compression

This section describes the options available for configuring compression of the data packets sent by the Networking Service.

In earlier OpenSplice 6.x releases, the `zlib` library was used at its default setting whenever the compression option on a network partition was enabled. In this release it is possible to configure `zlib` for less cpu usage or for more compression effort, or to select a compressor written specifically for high speed, or to plug in an alternative algorithm.

The configuration for compression in a Networking Service instance is contained in the optional top-level *Element Compression* (see Section *4.4.1.8* on page 241). These settings apply to all partitions in which compression is enabled.

2.5.8.1 Availability

The compression functionality is available on enterprise platforms (*i.e.* Linux, Windows and Solaris). On embedded platforms there are no built-in compressors included, but plugins may be used.

2.5.8.2 How to set the *level* parameter in zlib

Set the *Attribute PluginParameter* to a single digit between `0` (no compression) and `9` (maximum compression, more CPU usage). Leave the *Attribute PluginLibrary* and *Attribute PluginInitFunction* blank.

2.5.8.3 How to switch to other built-in compressors

Set the *Attribute PluginInitFunction* to the name of the initialisation function of one of the built-in compressors. These are `/ospl_comp_zlib_init/`, `/ospl_comp_lzf_init/` and `/ospl_comp_snappy_init/` for `zlib`, `lzf` and `snappy` respectively. As a convenience, the short names `zlib`, `lzf` and `snappy` are also recognized.

i Please note that not all compressors are available on all platforms. In this release `zlib` is available on Linux, Windows and Solaris; `lzf` and `snappy` are available only on RedHat Linux.

2.5.8.4 How to write a plugin for another compression library

Other compression algorithms may be used by the Networking Service. In order to do this it is necessary to build a library which maps the OpenSplice compression API onto the algorithm in question. This library may contain the actual compressor code or be dynamically linked to it.

Definitions for the compression API are provided in the include file `plugin/nw_compPlugin.h`. Five functions must be implemented.

The ***`maxsize`*** function.

This function is called when sizing a buffer into which to compress a network packet. It should therefore return the worst-case (largest) possible size of compressed data for a given uncompressed size. In most cases it is acceptable to return the uncompressed size, as the compress operation is allowed to fail if the resulting data is larger than the original (in which case the data is sent uncompressed). However, `snappy` for example will not attempt compression unless the destination buffer is large enough to take the worst possible result.

The ***`compress`*** function.

This function takes a block of data of a given size and compresses it into a buffer of a given size. It returns the actual size of the compressed data, or zero if an error ocurred (*e.g.* the destination buffer was not large enough).

The ***`uncompress`*** function.

This function takes a block of compressed data of given size and uncompresses it into a buffer also of given size. It returns the actual size of the uncompressed data, or zero if an error ocurred (*e.g.* the data was not in a valid compressed format).

The ***`exit`*** function.

This function is called at service shutdown and frees any resources used by the plugin.

The ***init*** function.

This function is called at service startup. It sets up the plugin by filling in a structure containing pointers to the four functions listed above. It also is passed the value of the *Attribute PluginParameter*. The plugin configuration structure includes a pointer to some unspecified state data which may be used to hold this parameter and/or any storage required by the compressor. This pointer is passed into the `compress` and `exit` functions.

By way of illustration, here is a simplified version of the code for `zlib`. The implementation is merely a veneer on the `zlib` library to present the required API.

```
|#include "nw_compPlugin.h"
#include "os_heap.h"
#include

unsigned long ospl_comp_zlib_maxsize (unsigned long srcsize)
{
   /* if the data can't be compressed into the same size buffer we'll send
uncompressed instead */
    return srcsize;
}

unsigned long ospl_comp_zlib_compress (void *dest, unsigned long destlen,
const void *source, unsigned long srclen, void *param)
{
    unsigned long compdsize = destlen;
    if (compress2 (dest, &compdsize, source, srclen, *(int *)param) ==
Z_OK)
    {
        return compdsize;
    }
    else
    {
        return 0;
    }
}

unsigned long ospl_comp_zlib_uncompress (void *dest, unsigned long
destlen, const void *source, unsigned long srclen)
{
    unsigned long uncompdsize = destlen;
    if (uncompress (dest, &uncompdsize, source, srclen) == Z_OK)
    {
        return uncompdsize;
    }
    else
    {
        return 0;
    }
}
```

```
void ospl_comp_zlib_exit (void *param)
{
    os_free (param);
}

void ospl_comp_zlib_init (nw_compressor *config, const char *param)
{
    /* param should contain an integer from 0 to 9 */

    int *iparam = os_malloc (sizeof (int));
    if (strlen (param) == 1)
    {
        *iparam = atoi (param);
    }
    else
    {
        *iparam = Z_DEFAULT_COMPRESSION;
    }
    config->maxfn = ospl_comp_zlib_maxsize;
    config->compfn = ospl_comp_zlib_compress;
    config->uncompfn = ospl_comp_zlib_uncompress;
    config->exitfn = ospl_comp_zlib_exit;
    config->parameter = (void *)iparam;
}
```

2.5.8.5 How to configure for a plugin

Step 1: Set *Attribute PluginLibrary* to the name of the library containing the plugin implementation.

Step 2: Set *Attribute PluginInitFunction* to the name of the initialisation function within that library.

Step 3: If the compression method is controlled by a parameter, set *Attribute PluginParameter* to configure it.

2.5.8.6 Constraints

The Networking Service packet format does not include identification of which compressor is in use. It is therefore necessary to use the *same* configuration on *all* nodes.

2.5.9 Compatibility and conformance

2.5.9.1 Conformance modes

The DDSI2 service operates in one of three modes: pedantic, strict and lax, which is configured using the Compatibility/StandardsConformance setting. The default is lax.

In pedantic mode, it strives very hard to strictly conform to the DDSI 2.1 standard. It even uses a vendor-specific extension for an essential element missing in the specification, used for specifying the GUID of a DCPS data reader or data writer in the discovery protocol; and it adheres to the specified encoding of the reliability QoS. This mode is of interest for compliancy testing but not for practical use, even though there is no application-level observable difference between an all-OpenSplice system using the DDSI2 service in pedantic mode and one operating in any of the other modes.

The second mode, strict, instead attempts to follow the intent of the specification while staying close to the letter of it. The points in which it deviates from the standard are in all probability editing errors that will be rectified in the next update. When operated in this mode, one would expect it to be fully interoperable with the other vendors' implementations, but this is not the case. The deviations in the other vendors' implementations are not required to implement DDSI 2.1, as is proven by the OpenSplice DDSI2 service, and they cannot rightly be considered 'true' implementations of the DDSI 2.1 standard.

The default mode, lax, attempts to work around (most of) the deviations of the other implementations, and provides interoperability with (at least) RTI DDS and InterCOM/Gallium DDS. For compatibility with TwinOaks CoreDX DDS, additional settings are needed. See Section 2.5.9.1.2, *Compatibility issues with TwinOaks*, on page 71 more information. In lax mode, the OpenSplice DDSI2 service not only accepts some invalid messages, but will even transmit them. The consequences for interoperability of not doing this are simply too severe.

It should be noted that if one configures two OpenSplice nodes with DDSI2 in different compliancy modes, the one in the stricter mode will complain about messages sent by the one in the less strict mode. Pedantic mode will complain about invalid encodings used in strict mode, strict mode will complain about illegal messages transmitted by the lax mode. There is nonetheless interoperability between strict and lax.

2.5.9.1.1 Compatibility issues with RTI

In lax mode, there should be no major issues with most topic types when working across a network, but within a single host there is a known problem with the way RTI DDS uses, or attempts to use, its shared memory transport to communicate with OpenSplice, which clearly advertises only UDP/IP addresses at which it is reachable. The result is an inability to reliably establish bidirectional communication between the two.

Disposing data may also cause problems, as RTI DDS leaves out the serialised key value and instead expects the reader to rely on an embedded hash of the key value. In the strict modes, the DDSI2 service requires a proper key value to be supplied, in the relaxed mode, it is willing to accept key hash, provided it is of a form that contains the key values in an unmangled form.

If an RTI DDS data writer disposes an instance with a key of which the serialised representation may be larger than 16 bytes, this problem is likely to occur. In practice, the most likely cause is using a key as string, either unbounded, or with a maximum length larger than 11 bytes. See the DDSI specification for details.

In strict mode, there is interoperation with RTI DDS, but at the cost of incredibly high CPU and network load, caused by a Heartbeats and AckNacks going back-and-forth between a reliable RTI DDS data writer and a reliable OpenSplice DCPS data reader. The problem is that once the OpenSplice reader informs the RTI writer that it has received all data (using valid AckNack message), the RTI writer immediately publishes a message listing the range of available sequence numbers and requesting an acknowledgement, which becomes an endless loop.

The best settings for interoperation appear to be:

- Compatibility/StandardsConforming: lax
- Compatibility/AckNackNumbitsEmptySet: 0

Note that the latter setting causes the DDSI2 service to generate illegal messages, and is the default when in lax mode.

2.5.9.1.2 Compatibility issues with TwinOaks

Interoperability with TwinOaks CoreDX requires:

- Compatibility/ManySocketsMode: `true`
- Compatibility/StandardsConforming: `lax`
- Compatibility/AckNackNumbitsEmptySet: `0`
- Compatibility/ExplicitlyPublishQosSetToDefault: `true`

The ManySocketsMode option needs to be changed from the default, to ensure that each domain participant has a unique locator, which is needed because TwinOaks CoreDX DDS does not include the full GUID of a reader or writer if it needs to address just one. Note the behaviour of TwinOaks CoreDX DDS is allowed by the specification.

The Compatibility/ExplicitlyPublishQosSetToDefault settings work around TwinOaks CoreDX DDS' use of incorrect default values for some of the QoS settings if they are not explicitly supplied during discovery.

2.6 The Tuner Service

The Tuner Service provides a remote interface to the monitor and control facilities of OpenSplice by means of the SOAP protocol. This enables the OpenSplice Tuner to remotely monitor and control, from any *reachable* location, OpenSplice services as well as the applications that use OpenSplice for the distribution of their data.

The exact fulfilment of these responsibilities is determined by the configuration of the Tuner Service. There is an overview of the available configuration parameters and their purpose in Section 4.4.1.7, *Element Tracing*, on page 234.

2.7 The DbmsConnect Service

The OpenSplice DbmsConnect Module is a pluggable service of OpenSplice that provides a seamless integration of the real-time DDS and the non-/near-real-time enterprise DBMS domains. It complements the advanced distributed information storage features of the OpenSplice Persistence Module (and vice versa).

Where (relational) databases play an essential role to maintain and deliver typically non- or near-real-time ‘enterprise’ information in mission systems, OpenSplice targets the real-time edge of the spectrum of distributing and providing ‘the right information at the right place at the right time’ by providing a Quality-Of-Service (QoS) aware ‘real-time information backbone’.

Changing expectations about the accessibility of information from remote/non-real-time information-stores and local/real-time sources lead to the challenge of lifting the boundaries between both domains. The DbmsConnect module of OpenSplice answers this challenge in the following ways:

- Transparently ‘connects’ the real-time DDS ‘information backbone’ to one or more ‘enterprise’ databases
- Allows both enterprise as well as embedded/real-time applications to access and share data in the most ‘natural’ way
- Allows OpenSplice to fault-tolerantly replicate enterprise information persisted in multiple relational databases in real-time
- Provides a pure ODBC/JDBC SQL interface towards real-time information via its transparent DbmsConnection
- Overcomes the lack of communication-control (QoS features controlling real-time behavior) of ‘talking’ to a remote DBMS
- Overcomes the lack of traditional 3GL and 4GL tools and features in processing information directly from a DDS backbone
- Allows for data-logging and analysis of real-time data persisted in a DBMS
- Aligns multiple and dispersed heterogeneous databases within a distributed system using the QoS-enabled data-distribution features of OpenSplice

The DbmsConnect module is unique in its dynamic configurability to achieve maximum performance:

- Dynamic DDS Partition/Topic selection and configurable content-filters to exchange exactly 'the right' information critical for performance and resource-challenged users
- Dynamic creation and mapping of DBMS database-tables and DDS topics to allow seamless data-exchange, even with legacy data models
- Selectable update-triggering per table/topic allowing for both real-time responsiveness as well as high-volume 'batch transfers'
- Works with ANY 3rd party SQL:1999 compatible DBMS system with an ODBC interface

The DbmsConnect module thus effectively eliminates traditional 'barriers' of the standalone technologies by facilitating seamless data-exchange between any ODBC compliant (SQL)database and the OpenSplice™ real-time DDS 'information-backbone'. Applications in traditionally separated mission-system domains can now exploit and leverage each other's information in a highly efficient (based upon 'current interest' as supported by the publish/subscribe paradigm of DDS), non-disruptive (obeying the QoS demands as expressed for data-items in DDS) and distributed service-oriented paradigm.

2.7.1 Usage

In order to understand the configuration and working of the DbmsConnect service, some basic concepts and use-cases will be covered in this chapter.

2.7.1.1 DDS and DBMS Concepts and Types Mapping

The concepts within DDS and DBMS are related to each other as listed in *Table 1*.

Table 1 DDS <> DBMS mapping: concepts

DDS	DBMS
Topic	Table
Type	Table structure
Instance	Primary key
Sample	Row
DataWriter.write()	`INSERT or UPDATE`
DataWriter.dispose()	`DELETE`

The primitive types available in both domains map onto each other as listed in *Table 2* below:

Table 2 DDS <> DBMS mapping: primitive types

DDS IDL type	DBMS column type (SQL:1999)
`boolean`	`BOOLEAN/TINYINT`
`short`	`SMALLINT`
`unsigned short`	`SMALLINT`
`long`	`INTEGER`
`unsigned long`	`INTEGER`
`long long`	`BIGINT`
`unsigned long long`	`BIGINT`
`float`	`REAL`
`double`	`DOUBLE`
`octet`	`BINARY(1)`
`char`	`CHAR(1)`
`wchar`	`CHAR(1)`
`string<length>`	`VARCHAR(<length>)`
`wstring<length>`	`VARCHAR(<length>)`

The mapping of complex (composite) types is as follows:

- Struct
  - Flattened out
  - Each member maps to a column with fully scoped name
- Union
  - Flattened out
  - Additional '`#DISCRIMINATOR#`' column
- Enumeration
  - An '`INTEGER`' typed column with fully scoped name
- Array and bounded sequence
  - Flattened out
  - '`[index]`' appended to fully scoped column name

2.7.1.2 General DbmsConnect Concepts

The DbmsConnect service can bridge data from the DDS domain to the DBMS domain and vice versa. In DDS, data is represented by topics, while in DBMS data is represented by tables. With DbmsConnect, a mapping between a topic and a table can be defined.

Because not all topic-table mappings have to be defined explicitly (DbmsConnect can do matching when the names are the same), namespaces can be defined. A namespace specifies or limits the context of interest and allows for easy configuration of all mappings falling (or defined in) a namespace. The context of interest for bridging data from DDS to DBMS, consists of a partition- and topicname expression. When bridging data from DBMS to DDS, the context of interest consists of a table-name expression.

A mapping thus defines the relationship of a table in DBMS with a topic in DDS and can be used to explicitly map a topic and table with different names, or define settings for a specific mapping only.

2.7.1.3 DDS to DBMS Use Case

When data in the DDS domain has to be available in the DBMS domain, the DbmsConnect service can be configured to facilitate that functionality. A topic in DDS will be mapped to a table in DBMS.

Scenario

In the DDS domain, we have topics `DbmsTopic` and `DbmsDdsTopic` that we want to make available to a database application. The database application expects the data from topic `DbmsTopic` to be available in an existing table with name `DbmsTable`. Data from the `DbmsDdsTopic` topic can be just forwarded to a table (that not yet exists) with the same name. The scenario is shown in *Figure 7*.

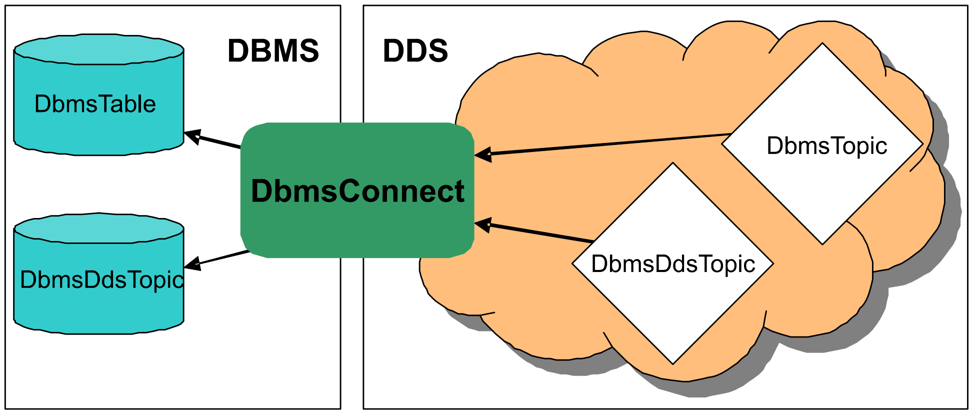


Figure 7 DDS to DBMS scenario

Configuration

The configuration for the DbmsConnect service that fulfils the needs of the scenario is given in the listing below.

```
1   ...
2   <DbmsConnectService name="dbmsconnect">
3      <DdsToDbms>
4         <NameSpace partition="*" topic="Dbms*"
5            dsn="DSN" usr="USR" pwd="PWD" odbc="ODBC">
6            <Mapping topic="DbmsTopic" table="DbmsTable"/>
7         </NameSpace>
8      </DdsToDbms>
9   </DbmsConnectService
10  ...
```

Explanation

On line *3* a `DdsToDbms` element is specified in order to configure data bridging from DDS to DBMS. On line *4*, a `NameSpace` is defined that has interest in all topics starting with `"Dbms"` in all partitions. Both the partition- and topic-expression make use of the `*`-wildcard (matching any sequence of characters). These wildcards match both topics described in the scenario, but will possibly match more. If the mapping should be explicitly limited to both topics, the topic-expression can be changed to `"DbmsTopic,DbmsDdsTopic"`.

The DbmsConnect service will implicitly map all matching topics to an equally named table in the DBMS. While this is exactly what we want for the `DbmsDdsTopic`, the database application expects the data from the `DbmsTopic` topic to be mapped to table `DbmsTable`. This is explicitly configured in the `Mapping` on line *6*. If the tables already exist and the table-definition matches the topic definition, the service will use that table. If a table does not exist, it will be created by the service. If a table exists, but doesn't match the topic definition, the mapping fails.

2.7.1.4 DBMS to DDS Use Case

When data in the DBMS domain has to become available in the DDS domain, this can be achieved by configuring the DbmsConnect service to map a table to a topic.

Scenario

In the DBMS, we have tables `DbmsTable` and `DbmsDdsTopic` that we want to make available in the `dbmsPartition` partition in DDS. The database application writes the data we want available in topic `DbmsTopic` to the table named `DbmsTable`. Data from the `DbmsDdsTopic` table can be just forwarded to the equally named topic.

When the DbmsConnect service is started, mapped tables may already contain data. For the `DbmsDdsTopic` table, we are not interested in that data. For the `DbmsTable` table however, we would like all data available to the database application to be available to the DDS applications too. This scenario is the reverse (all arrows reversed) situation of the scenario shown in *Figure 7* on page 76

Configuration

The configuration for the DbmsConnect service that fulfils the needs of the scenario is given in the listing below.

```
11 ...
12 <DbmsConnectService name="dbmsconnect">
13    <DbmsToDds publish_initial_data="false">
14       <NameSpace partition="dbmsPartition" table="Dbms*"
15          dsn="DSN" usr="USR" pwd="PWD" odbc="ODBC">
16          <Mapping topic="DbmsTopic" table="DbmsTable"
17             publish_initial_data="true"/>
18       </NameSpace>
19    </DbmsToDds>
20 </DbmsConnectService
21 ...
```

Explanation

On line *13* a `DdsToDbms` element is specified in order to configure data bridging from DBMS to DDS. On line *14*, a `NameSpace` is defined that has interest in all tables starting with `"Dbms"`. The table-expression makes use of the `*`-wildcard (matching any sequence of characters). For this scenario, a single target partition is specified. If needed, a partition expression containing multiple partitions or wildcards can be used. For example when the DDS system is divided in two partitions (to support applications running in a ‘simulation’- and a ‘real’ world) and applications in both partition need access to the data from the DBMS.

The default setting for the `publish_initial_data` attribute is `true`. Because we only want initially available data to be published for the `DbmsTable-DbmsTopic` mapping, we define the default for all namespaces to be `false` on line *13*. That setting will be inherited by all underlying elements, but can be overridden. The explicit `Mapping` specified on line *16* maps the table to the differently named topic. On line *17*, the `publish_initial_data` attribute is explicitly set to `true`, overriding the setting set at line *13*.

2.7.1.5 Replication Use Case

A specific application of data bridging from DDS to DBMS and DBMS to DDS is replication of database (tables). Replication requires some specific configuration. The basic configuration is covered in this use case.

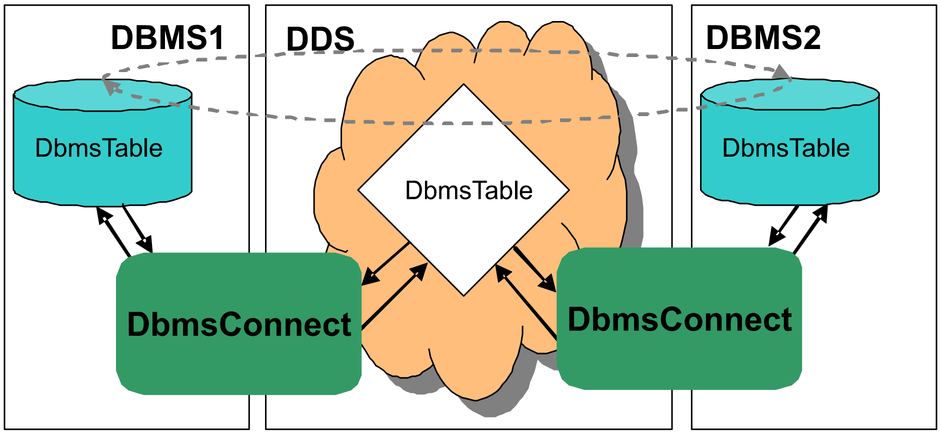


Figure 8 Replication scenario

Scenario

We have a two database servers running on different hosts. The table `DbmsTable` should be available on both database-servers and changes should be sent both ways. This scenario is shown in *Figure 8*, where the dashed arrows show the transparant role of DDS in this scenario.

Configuration

The configuration for the DbmsConnect service for both hosts, that fulfils the needs of the scenario, is given in the listing below.

```
22 ...
23 <DbmsConnectService name="dbmsconnect">
24    <DdsToDbms replication_mode="true">
25       <NameSpace partition="replication" topic="DbmsTable"
26          dsn="DSN" usr="REPLUSR" pwd="PWD" odbc="ODBC">
27       </NameSpace>
28    </DdsToDbms>
29    <DbmsToDds replication_user="REPLUSR">
30       <NameSpace partition="replication" table="DbmsTable"
31          dsn="DSN" usr="USR" pwd="PWD" odbc="ODBC">
32       </NameSpace>
33    </DbmsToDds>
34 </DbmsConnectService
35 ...
```

Explanation

The configuration for the replication scenario is symmetric in that it can be the same for both hosts. The basic idea is simple: configure a mapping from DDS to DBMS and from DBMS to DDS for the same table-topic pair within a partition (analogue to the *DDS to DBMS Use Case* on page 75 and the *DBMS to DDS Use Case* on page 77). While this (intuitive) cyclic definition would work, more configuration is needed to support this use case properly. In order to prevent modifications from keeping to cause (cyclic) updates, the DbmsConnect service needs to be able to distinguish between data that is modified as part of a replication scenario and data that is changed by other sources.

For the DDS to DBMS mapping, replication data is identified by identification information of the sending node. The `DdsToDbms` part of the service is put to replication mode in line *24*, which lets the service ignore all data transmitted by the node on which the service itself runs.

For the DBMS to DDS mapping, a special database user has to be used, that is only used by the DbmsConnect service, in order to be able to distinguish data from different sources. The database user that is used in the `DdsToDbms` mapping has to be excluded from update-cascading. This is specified on line *29* in the `replication_user` attribute. This means that all data that is inserted in the table by the user with the username specified in the `replication_user` attribute will be ignored. So the user specified at line *26* has to be the same as the user specified on line *29*.

CHAPTER

3 Tools

The OpenSplice DDS distribution contains several tools for a variety of different purposes. This chapter categorizes the various tools and gives some background about each of them. For each tool it will either refer to the appropriate manual, or provide a separate section describing a more in-depth overview of the tools' possibilities and command-line interface.

3.1 Introduction

The OpenSplice DDS tool chain is comprised of the following categories:

• Compilers/Code generators:

- ***`idlpp`*** (IDL Pre-processor): parses and validates an IDL file containing your DCPS data model. When valid, it generates a language specific representation of this data model accompanied by the corresponding DCPS accessor classes (*e.g.* DataReader, DataWriter and TypeSupport). More details about this tool can be found in the *IDL Pre-processor Guide* contained in the file `OpenSplice_PreProcessor_usermanual.pdf`.
- ***`rmipp`*** (RMI pre-processor): parses and validates an IDL file containing your RMI interface model. When valid, it generates a language specific representation of this interface accompanied by the corresponding RMI-DDS translators. More details about this tool can be found in section 3.4 of the *OpenSplice RMI over DDS Getting Started Guide* contained in the file `OpenSplice_RMI_GettingStarted.pdf`.

• Configuration Editor:

- ***`osplconf`*** (OpenSplice Configurator): a GUI based editor for the OpenSplice DDS configuration files, providing context sensitive validation and help. More details about this tool can be found in Section 3.2, *osplconf: the OpenSplice Configuration editor*, on page 82.

• Control & Monitoring Tools:

- ***`ospl`*** (OpenSplice service manager): a tool that can be used to start, stop and monitor the OpenSplice DDS Domain Service (only applicable to the Federated Deployment Mode). More details about this tool can be found in Section 3.3, *ospl: the OpenSplice service manager*, on page 85.

- ***mmstat*** (Memory Management Statistics): a tool that can display several statistics about the shared memory that is currently being used by an OpenSplice Domain Service (only applicable to the Federated Deployment Mode). More details about this tool can be found in Section 3.4, *mmstat: Memory Management Statistics*, on page 86.
- ***ospltun*** (OpenSplice Tuner): a tool that can be used to monitor and control individual DCPS entities. With this tool you can display the DCPS Entity trees of your application, watch (and possibly modify) the Qos settings of an individual DCPS entity, monitor the status flags it has currently raised, examine many more statistics about these entities, and even monitor and inject samples into your DCPS Readers/Writers. It can connect directly into the shared memory (restricted to the Federated Deployment Mode), or through a socket to a pre-configured Tuner Service using the SOAP protocol. More details about this tool can be found in the *OpenSplice DDS Tuner Guide* contained in the file `OpenSplice_Tuner_usermanual.pdf`.
- ***ospltest*** (OpenSplice Tester): an automated testing and debugging tool that can be used to receive and display messages produced in OpenSplice, and to transmit your own messages either manually or with a script. Like the Tuner, it can connect directly to the shared memory (restricted to the Federated Deployment Mode), or through a socket to a pre-configured Tuner Service using the SOAP protocol. More details about this tool can be found in the *OpenSplice Automated Testing and Debugging Tool User Guide* contained in the file `OpenSplice_Tester_usermanual.pdf`.

The following sections will provide some more details about those tools that do not have a separate manual.

3.2 osplconf: the OpenSplice Configuration editor

The OpenSplice Configuration Editor provides the following command line instruction:

`-uri=[URI]` — the domain config file that needs to be opened; *e.g.*

```
osplconf -uri=$OSPL_URI
```

When started the OpenSplice Configuration Editor can help you in several ways to tailor the deployment settings for your OpenSplice DDS system:

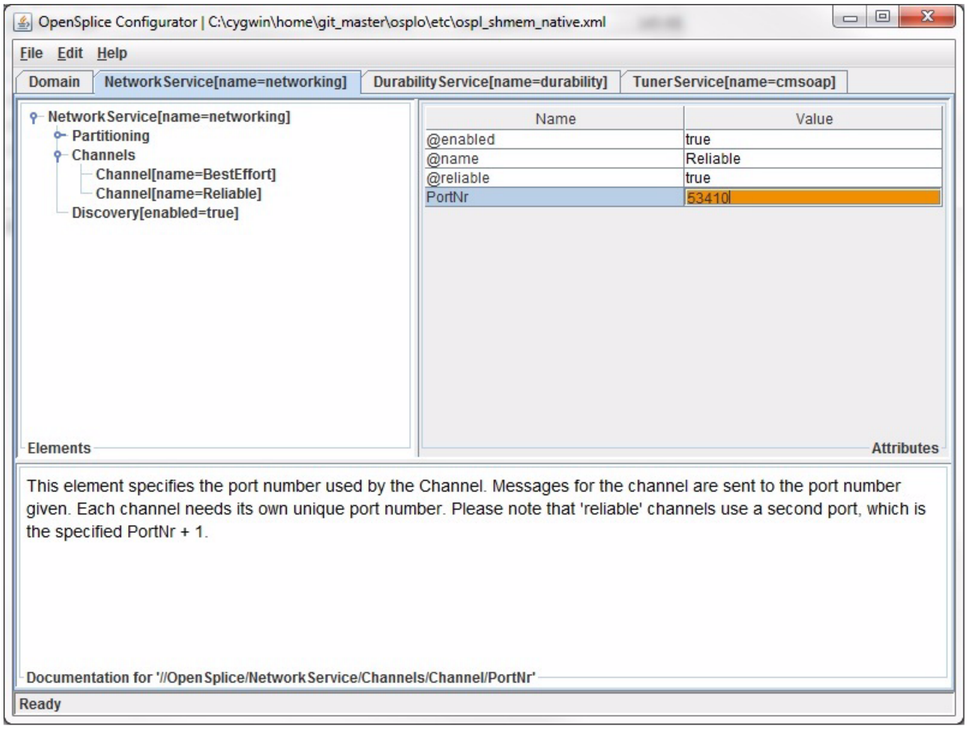
- It displays the configured services as separate tabs in a tabbed pane.
- For each service, it displays the relevant service settings in a hierarchical tree.
- For each setting, it provides a context-sensitive description in the bottom pane.

- The content of this context-sensitive help is identical to the textual information contained in Chapter 4, *Service Configuration*, starting on page 93 of this manual. However, the tables preceding each description in this manual may give some additional information regarding for example the unit of an attribute (*e.g.* Bytes per resolution tick).

• The value of each element/attribute can be edited. A context-sensitive validation algorithm will check whether your input satisfies the relevant criteria.

- When the input color is orange, you are editing the value.
- When the text field color is red, the value is unacceptable.
- When the text field color is white, the new input value has been accepted.

A typical view of the OpenSplice Configurator is displayed below:


OpenSplice Configurator | C:\cygwin\home\git_master\osplo\etc\ospl_shmem_native.xml
File Edit Help
Domain
NetworkService[name=networking]
DurabilityService[name=durability]
TunerService[name=cmsoap]
NetworkService[name=networking]
Partitioning
Channels
Channel[name=BestEffort]
Channel[name=Reliable]
Discovery[enabled=true]
Elements
Attributes
Ready

Name	Value
@enabled	true
@name	Reliable
@reliable	true
PortNr	53410

This element specifies the port number used by the Channel. Messages for the channel are sent to the port number given. Each channel needs its own unique port number. Please note that 'reliable' channels use a second port, which is the specified PortNr + 1.

Documentation for '//OpenSplice/NetworkService/Channels/Channel/PortNr'

Figure 9 Typical Configurator view

A config file is opened using the top menu bar (**File > Open**) or the keyboard shortcut Ctrl+O.

The appropriate service tab is selected.

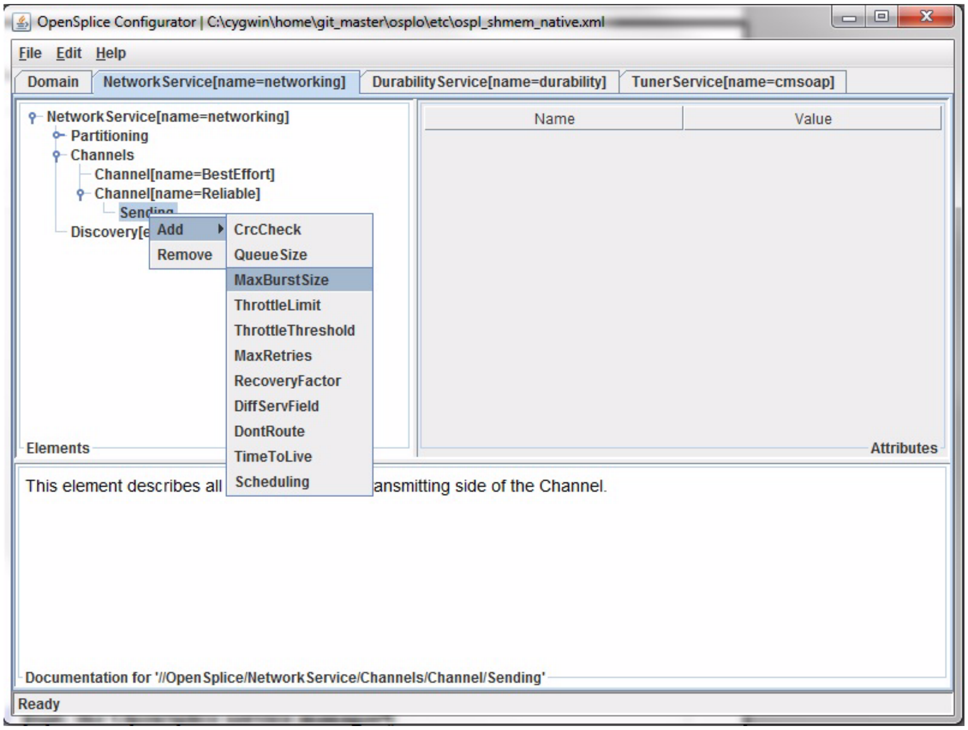
If the appropriate service is not configured, and so its tab is not visible on the top, it can be added by using the top menu-bar (**Edit > Add Service**).

The hierarchical tree on the left can be used to browse through the settings applicable to the Service and possibly modify them.

The right pane shows the settings of the currently selected tree node.

An item prefixed with a '@' represents an XML attribute. The other items represent XML elements.

If the appropriate setting is not currently configured, and therefore not visible in the tree, you can add it by right-clicking anywhere in the tree to open a context-sensitive sub-menu displaying all available settings for that particular element in the tree.



This element describes all ... ansmitting side of the Channel.

Documentation for '//OpenSplice/NetworkService/Channels/Channel/Sending'

Figure 10 Adding an element in Configurator

Once the appropriate modifications have been made, and are accepted by the Configurator, the config file can be saved using the top menu bar (**File > Save**) or the keyboard shortcut Ctrl+S.

Likewise, a config file can be written from scratch by using the top menu bar (**File > New**) or the keyboard shortcut Ctrl+N.

3.3 ospl: the OpenSplice service manager

The OpenSplice service manager (`ospl`) is a tool that monitors and controls the lifecycle of the OpenSplice Domain Service (`spliced`), which in turn monitors and controls all other OpenSplice services. This tool is only applicable to the Federated Deployment Mode, because the Single Process Deployment Mode doesn't need to run external services. Basically you can view the OpenSplice service manager as a controller around the OpenSplice Domain Service, that can be used to pass the following command-line instructions to the Domain Service:

`start [URI]` — Starts a Domain Service for the specified URI. (It looks for the environment variable `OSPL_URI` when no URI is explicitly passed.) The Domain Service will in turn parse the config file indicated by the URI and start all configured services according to their settings.

When done, the OpenSplice service manager will return one of the following exit codes:

`0` normal termination when the Domain Service has successfully started.

`1` a recoverable error has occurred (*e.g.* out of resources)

`2` an unrecoverable error has occurred (*e.g.* config file contains errors).

When also passing the `-f` flag, the OpenSplice service manager will not return the command prompt, but remain blocked until the Domain Service successfully terminates. Any termination event sent to the service manager will in that case be forwarded to the Domain Service it manages.

`stop [URI]` — Stops the Domain Service for the specified URI. (It looks for the environment variable `OSPL_URI` when no URI is explicitly passed.) The Domain Service will in turn wait for all the services it currently monitors to terminate gracefully and will then terminate itself.

When done, the OpenSplice service manager will return one of the following exit codes:

`0` normal termination when the Domain Service has successfully terminated.

`2` an unrecoverable error has occurred (*e.g.* config file cannot be resolved).

When passing the `-a` flag instead of a URI, the OpenSplice manager is instructed to terminate all Domain Services that are currently running on the local node.

`status [URI]` — Prints the status of the Domain Service for the specified URI (it looks for the environment variable `OSPL_URI` when no URI is explicitly passed.) When a Domain with the specified URI cannot be found, it prints nothing.

`list` — Lists all Domain Services by name (*i.e.* the name configured in the `OpenSplice/Domain/Name` element of the config file). This behaviour is similar to the status option, but then for all Domains that are currently running on the local node.

There are a couple of other flags that can be used to display valuable information:

`-v` — prints the version number of the current OpenSplice release.

`-h` — prints help for all command-line options.

Note that the default behaviour of `ospl` without any command-line arguments is to display help.

3.4 mmstat: Memory Management Statistics

`Mmstat` is a command-line tool that can display valuable information about the shared memory statistics of an OpenSplice Domain (this is only applicable to the Federated Deployment Mode, since the Single Process Deployment Mode does not use shared memory). The Domain to which `mmstat` must attach can be passed as a command-line parameter, and consists of a URI to the config file specifying the Domain. When no URI is passed, `mmstat` will attach to the Domain specified in the environment variable `OSPL_URI`.

Basically `mmstat` can run in four separate modes, which all display their status at regular intervals. This interval time is by default set to 3 seconds, but can be overruled by passing the `-i` flag followed by an interval value specified in milliseconds. The following modes can be distinguished using the specified flags:

`-m` The memory statistics mode (default mode)

`-M` The memory statistics difference mode

`-t` The meta-object references mode

`-T` The meta-object references difference mode

`Mmstat` will keep on displaying an updated status after every interval until the `q` key is pressed, or until the total number of iterations reaches the `sample_count` limit that can be specified by passing the `-s` flag followed by the preferred number of iterations. Intermediate status updates can be enforced by pressing the `t` key.

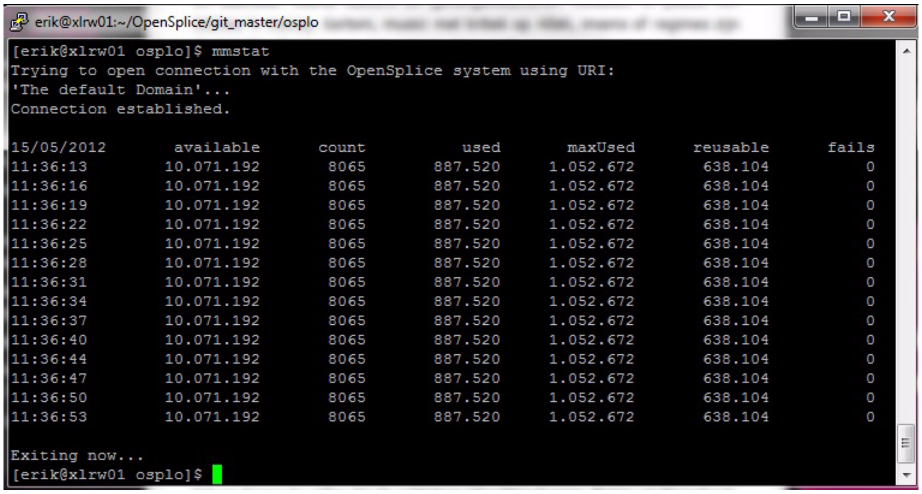
The following subsections will provide a more detailed description of the different mmstat modes presented above.

3.4.1 The memory statistics mode

In the memory statistics mode `mmstat` basically displays some general shared memory statistics that can help in correctly estimating the required size of the shared memory database in the configuration file. The numbers that will be displayed in this mode are:

- The total amount of shared memory still available (*i.e.* currently not in use).
- the number of objects currently allocated in the shared memory.
- the amount of shared memory that is currently in use by the allocated objects.
- the worstcase amount of shared memory that has been in use so far.
- the amount of shared memory that is currently marked as reuasble. (Reusable memory is memory that is conceptually available, but it might be fragmented in small chunks that cannot be allocated in bigger chunks.)

The memory statistics mode is the default mode for `mmstat`, and it is selected when no explicit mode selection argument is passed. It can also be selected explicitly by passing the `-m` flag.

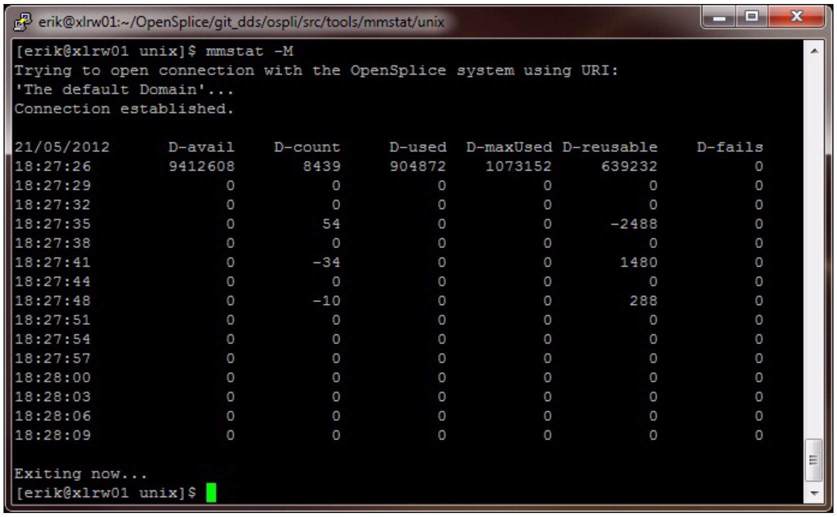


```
[erik@xlrw01 osplo]$ mmstat
Trying to open connection with the OpenSplice system using URI:
'The default Domain'...
Connection established.

15/05/2012       available      count        used      maxUsed     reusable      fails
11:36:13        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:16        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:19        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:22        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:25        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:28        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:31        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:34        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:37        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:40        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:44        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:47        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:50        10.071.192       8065     887.520    1.052.672      638.104          0
11:36:53        10.071.192       8065     887.520    1.052.672      638.104          0

Exiting now...
[erik@xlrw01 osplo]$
```

Figure 11 Typical mmstat view

3.4.2 The memory statistics difference mode

The memory statistics difference mode works very similarly to the memory statistics mode, but instead of displaying the current values of each measurement it displays the changes of each value relative to the previous measurement. This provides a good overview of the dynamics of your shared memory, such as whether it remains stable, whether it is rapidly being consumed/released, and so on.


erik@xlrw01:~/OpenSplice/git_dds/ospli/src/tools/mmstat/unix

```
[erik@xlrw01 unix]$ mmstat -M
Trying to open connection with the OpenSplice system using URI:
'The default Domain'...
Connection established.

21/05/2012       D-avail    D-count     D-used  D-maxUsed D-reusable     D-fails
18:27:26         9412608       8439     904872    1073152     639232           0
18:27:29               0          0          0          0          0           0
18:27:32               0          0          0          0          0           0
18:27:35               0         54          0          0      -2488           0
18:27:38               0          0          0          0          0           0
18:27:41               0        -34          0          0       1480           0
18:27:44               0          0          0          0          0           0
18:27:48               0        -10          0          0        288           0
18:27:51               0          0          0          0          0           0
18:27:54               0          0          0          0          0           0
18:27:57               0          0          0          0          0           0
18:28:00               0          0          0          0          0           0
18:28:03               0          0          0          0          0           0
18:28:06               0          0          0          0          0           0
18:28:09               0          0          0          0          0           0

Exiting now...
[erik@xlrw01 unix]$
```

Figure 12 Mmstat memory statistics difference mode

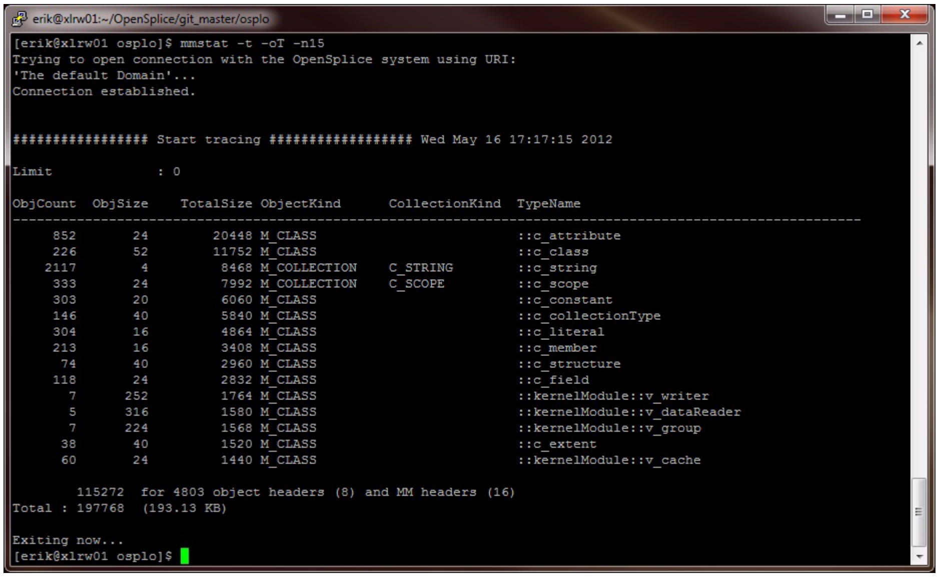
The numbers that will be displayed in this mode are:

- the difference in the amount of available shared memory relative to the previous measurement.
- the difference in the number of objects that is allocated in the shared memory relative to the previous measurement.
- the difference in the amount of shared memory that is in use by the allocated objects relative to the previous measurement.
- the difference in the worstcase amount of shared memory that has been allocated since the previous measurement. Notice that this value can only go up and so the difference can never be negative.

The memory statistics difference mode can be selected by explicitly passing the `-M` flag as a command-line parameter.

3.4.3 The meta-object references mode

In the meta-object references mode `mmstat` basically displays which objects are currently populating the shared memory. For this purpose it will iterate through all datatypes known to the Domain, and for each datatype it will display the following information:



```
[erik@xlrw01 osplo]$ mmstat -t -oT -n15
Trying to open connection with the OpenSplice system using URI:
'The default Domain'...
Connection established.


################# Start tracing ################## Wed May 16 17:17:15 2012

Limit             : 0

ObjCount  ObjSize    TotalSize ObjectKind      CollectionKind  TypeName
----------------------------------------------------------------------------------------------------------
     852       24        20448 M_CLASS                         ::c_attribute
     226       52        11752 M_CLASS                         ::c_class
    2117        4         8468 M_COLLECTION    C_STRING        ::c_string
     333       24         7992 M_COLLECTION    C_SCOPE         ::c_scope
     303       20         6060 M_CLASS                         ::c_constant
     146       40         5840 M_CLASS                         ::c_collectionType
     304       16         4864 M_CLASS                         ::c_literal
     213       16         3408 M_CLASS                         ::c_member
      74       40         2960 M_CLASS                         ::c_structure
     118       24         2832 M_CLASS                         ::c_field
       7      252         1764 M_CLASS                         ::kernelModule::v_writer
       5      316         1580 M_CLASS                         ::kernelModule::v_dataReader
       7      224         1568 M_CLASS                         ::kernelModule::v_group
      38       40         1520 M_CLASS                         ::c_extent
      60       24         1440 M_CLASS                         ::kernelModule::v_cache

        115272  for 4803 object headers (8) and MM headers (16)
Total : 197768  (193.13 KB)

Exiting now...
[erik@xlrw01 osplo]$ 
```

Figure 13 Mmstat meta-object references mode

- the number of objects currently allocated for the indicated type.
- the memory allocation footprint of a single object of the indicated type.
- the combined size taken by all objects of the indicated type.
- The kind of object (*e.g.* class, collection, etc.).
- The kind of collection (when appropriate).
- The fully scoped typename.

In normal circumstances the reference list will be so long (only the bootstrap will already inject hundreds of types into the Domain) that it will not fit on one screen. For that reason there are several ways to restrict the number of items that are displayed, by filtering out the non-interesting items:

- A filter can be specified by passing the `-f` flag, followed by a (partial) typename. This restricts the list to the only those datatypes that match the filter.
- The maximum number of items that may be displayed can be specified by passing the `-n` flag, followed by the maximum value.

  This is especially useful when combined with another flag that determines the order in which the items will be displayed. For example, when the items are sorted by memory footprint, passing `-n10` will only display the top ten datatypes that have the biggest footprint.

The order of the items in the list can be controlled by passing the `-o` flag, followed by a character specifying the ordering criterion. The following characters are supported:

C — Sort by object Count (*i.e.* the number of allocated objects from the indicated datatype).

S — Sort by object Size (*i.e.* the memory footprint of a single object from the indicated datatype).

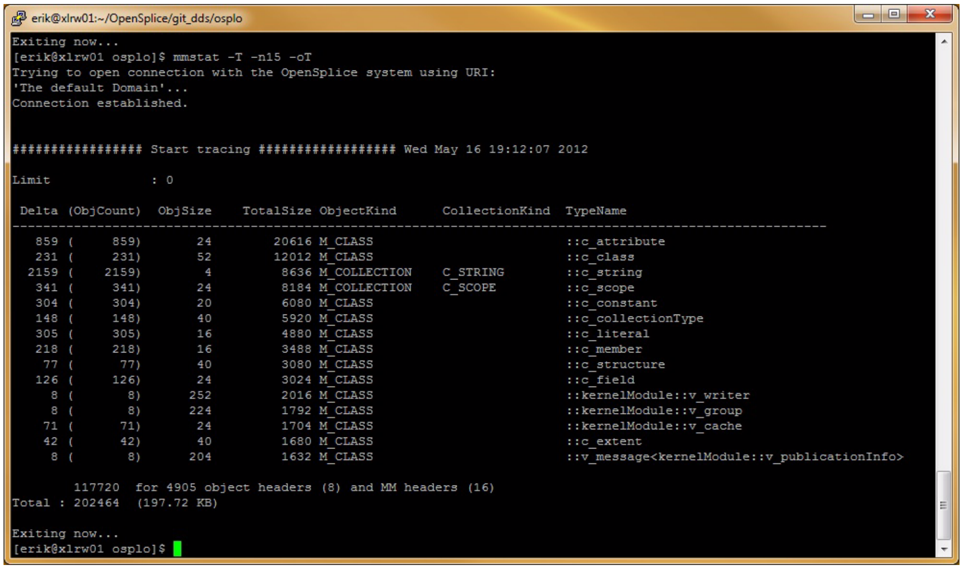
T — Sort by Total size (*i.e.* the combined memory footprint of all objects allocated from the indicated datatype).

3.4.4 The meta-object references difference mode

The meta-object references difference mode is very similar to the meta-object references mode, but instead of displaying the current values of each measurement it displays the changes of each value relative to the previous measurement. This provides a good overview of the dynamics of your shared memory, such as whether the number of objects remains stable, whether it is rapidly increasing/decreasing, and so on.

The fields that are displayed in this mode are similar to the fields displayed in the meta-object references mode, except that the numbers displayed in the first and third column are now specifying the changes relative to the previous measurement.

All the flags that are applicable to the meta-object references mode are also applicable to the meta-object references difference mode, but keep in mind that ordering (when specified) is now based on the absolute value of the difference between the current and the previous measurement. This way big negative changes will still be displayed at the top of the list.



erik@xlrw01:~/OpenSplice/git_dds/osplo

```
Exiting now...
[erik@xlrw01 osplo]$ mmstat -T -n15 -oT
Trying to open connection with the OpenSplice system using URI:
'The default Domain'...
Connection established.


################# Start tracing ################## Wed May 16 19:12:07 2012

Limit             : 0

 Delta (ObjCount)  ObjSize    TotalSize ObjectKind      CollectionKind  TypeName
----------------------------------------------------------------------------------------------------------
   859 (     859)       24        20616 M_CLASS                         ::c_attribute
   231 (     231)       52        12012 M_CLASS                         ::c_class
  2159 (    2159)        4         8636 M_COLLECTION    C_STRING        ::c_string
   341 (     341)       24         8184 M_COLLECTION    C_SCOPE         ::c_scope
   304 (     304)       20         6080 M_CLASS                         ::c_constant
   148 (     148)       40         5920 M_CLASS                         ::c_collectionType
   305 (     305)       16         4880 M_CLASS                         ::c_literal
   218 (     218)       16         3488 M_CLASS                         ::c_member
    77 (      77)       40         3080 M_CLASS                         ::c_structure
   126 (     126)       24         3024 M_CLASS                         ::c_field
     8 (       8)      252         2016 M_CLASS                         ::kernelModule::v_writer
     8 (       8)      224         1792 M_CLASS                         ::kernelModule::v_group
    71 (      71)       24         1704 M_CLASS                         ::kernelModule::v_cache
    42 (      42)       40         1680 M_CLASS                         ::c_extent
     8 (       8)      204         1632 M_CLASS                         ::v_message<kernelModule::v_publicationInfo>

        117720  for 4905 object headers (8) and MM headers (16)
Total : 202464  (197.72 KB)

Exiting now...
[erik@xlrw01 osplo]$
```

Figure 14 Mmstat meta-object references difference mode

CHAPTER 4 Service Configuration

4.1 Introduction

This chapter provides a more in-depth description of the OpenSplice DDS configuration by describing the most important configuration parameters for all available services. Each configuration parameter will be explained by means of an extensive description together with the tabular summary that contains the following information:

- *Full path* - Describes the location of the item within a complete configuration. Because the configuration is in the XML format, an XPath expression is used to point out the name and location of the configuration item.
- *Format* - Describes the format of the value of the configuration item.
- *Dimension* - Describes the unit for the configuration item (*e.g.* seconds or bytes).
- *Default value* - Describes the default value that is used by service when the configuration item is not in the configuration.
- *Valid values* - Describes the valid values for the configuration item. This can be a range or a set of values.

In case the configuration parameter is an XML attribute, the table also contains the following information:

- *Required* - Describes whether the attribute must be specified explicitly or is optional.

In case the configuration parameter is an XML element, the table also contains the following information:

- *Occurrences* - Describes the range of the possible number of occurrences of the element in the configuration by specifying the minimum and maximum number of occurrences.
- *Child-elements* - Describes the child-elements supported by the current element.
- *Required attributes* - Describes the required attributes, i.e. the attributes that must be specified inside the current element.
- Optional attributes - Describes the optional attributes, i.e. the attributes that may, but do not need to be specified inside the current element.

4.2 The Domain Service

The Domain Service is responsible for creating and initialising the DDS database which is used by the administration to manage a specific DDS Domain on a computing node. Without this administration, no other service or application is able to participate in a DDS Domain. Once the administration has been initialised, the Domain service starts the set of pluggable services. The lifecycle of the started services is under control of the Domain service, which means that it will monitor the health of all started services, take corrective actions if needed and stop the services when it is terminated.

When a shutdown of the OpenSplice Domain service is requested, it will react by announcing the shutdown using the shared administration. Applications will not be able to use DDS functionality anymore and services are requested to terminate elegantly. Once this has succeeded, the Domain service will destroy the shared administration and finally terminate itself.

Please refer to section *1.4* on page 17 for notes about applications operating in multiple domains.

`Full path`	OpenSplice/Domain
`Occurrences (min-max)`	1 - 1

`Child-elements`	*Element Id* *Element Name* *Element CPUAffinity* *Element Role* *Element Lease* *Element ServiceTerminatePeriod* *Element SingleProcess* *Element Database* *Element Service* *Element Application* *Element Listeners* *Element BuiltinTopics* *Element PriorityInheritance* *Element Statistics* *Element ReportPlugin* *Element PartitionAccess* *Element TopicAccess* *Element ResourceLimits* *Element Report* *Element Daemon* *Element GeneralWatchdog* *Element UserClockService*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.1 Element *Id*

This element specifies the domain Id of the instantiated DDS domain. If several different DDS domains are required to run simultaneously, then they all need to have their own unique domain Id.

NOTE: For maximum interoperability it is recommended that you only select a domain Id from the range `0 < n < 230`. The domain Id value is used by the DDSI2 service to derive values for the required network communiction endpoints, and service reconfiguration is required to use domain Id values outside of this range.

Please also see section 9.6.1 of the Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol specification (DDSI), v2.1, formal/2009-01-05 at *http://www.omg.org/spec/DDSI/2.1/* for further information.

Full path	OpenSplice/Domain/Id
Format	signed integer
Dimension	n/a
Default value	1
Valid values	0 - maxInt (Recommended: 1 - 229)
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.2 Element *Name*

This element specifies the name of the instantiated DDS domain. In general, it is recommended to change this name to a name that uniquely identifies the domain.

Full path	OpenSplice/Domain/Name
Format	string
Dimension	n/a
Default value	OpenSplice<version>
Valid values	any string
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.3 Element *CPUAffinity*

This element specifies which CPUs OpenSplice should be constrained to run on. It is currently only supported for VxWorks SMP kernel mode builds.

This element consists of a comma-separated list of CPU numbers that OpenSplice is to have its CPU affinity set to.

`Full path`	OpenSplice/Domain/CPUAffinity
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	Comma-separated list of CPU IDs
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.4 Element *Role*

Within a system and depending on the hosted application a Domain Service can have a specific role and interaction with other Domain Services may depend on this role.

The Role element is a user-defined string value that is communicated through the system, the behavior of other Domain Services *i.e.* how they interact with a Domain Service can be configured depend of the role by means of string matching expressions. For example, a Domain Service could limit its communication with other Domain Services by only accepting specific roles. (See also Section 4.4.1.6.2, *Attribute Scope*, on page 226).

`Full path`	OpenSplice/Domain/Role
`Format`	string
`Dimension`	n/a
`Default value`	DefaultRole
`Valid values`	any string
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.5 Element *Lease*

The Lease parameter specifies the death detection time of the Domain Service. All internal tasks performed by the Domain Service will periodically update their liveliness; when one or more tasks fail to update its liveliness the Domain will take action to either repair the failing functionality, continue in a degraded mode, or halt, depending on the configured desired behaviour.

`Full path`	OpenSplice/Domain/Lease
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element ExpiryTime*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.5.1 Element *ExpiryTime*

This element specifies the interval in which services have to announce their liveliness.

Every OpenSplice DDS service, including the Domain Service itself, has to announce its liveliness regularly. This allows corrective actions to be taken when one of the services becomes non-responsive. This element specifies the required interval. Decreasing the interval decreases the time in which non-responsiveness of a service is detected, but leads to more processing. Increasing it has the opposite effect.

`Full path`	OpenSplice/Domain/Lease/ExpiryTime
`Format`	float
`Dimension`	seconds
`Default value`	10.0
`Valid values`	0.2 - maxFloat
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute update_factor*
`Optional attributes`	<none>

4.2.5.1.1 Attribute update_factor

In case of a temporary high CPU load, the scheduling behaviour of the operating system might affect the capability of a service to assert its liveliness ‘on time’. The *update_factor* attribute introduces some elasticity in this mechanism by making the services assert their liveliness more often than required by the *ExpiryTime*. Services will report their liveliness every *ExpiryTime* multiplied by this *update_factor*.

Full path	OpenSplice/Domain/Lease/ExpiryTime[@update_factor]
Format	float
Dimension	n/a
Default value	0.2
Valid values	0.01 - 1.0
Required	yes

4.2.6 Element *ServiceTerminatePeriod*

This element specifies the amount of time the Domain Service, when instructed to terminate, should wait for the other configured Services to terminate. When this element is configured to '0' the Domain service will terminate without any wait time at all. Be aware that without any wait time the deamon will use a hard kill on any lingering service that can not terminate fast enough. This may prevent graceful termination and thus leave applications that are still attached to the DDS domain in an undefined state. Consequently the '0' value should only be used when there is some form of process management on top of OpenSplice DDS.

Full path	OpenSplice/Domain/ServiceTerminatePeriod
Format	float
Dimension	seconds
Default value	10.0
Valid values	0.0 - 60.0
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.7 Element *SingleProcess*

This element specifies whether the OpenSplice Domain and other OpenSplice services and applications are intended to be all deployed within the same process, known in OpenSplice as a *single process*.

Please note that the choice to use the single process deployment also implies the use of heap memory for the OpenSplice database management instead of shared memory that would be used otherwise. The heap memory is limited by the Operating System, so the `Database` element under `Domain` does not take effect when *`SingleProcess`* has a value of `True`.

There are two ways in which to deploy an OpenSplice application as a single process:

Single Process Application - The user starts a DDS application as a new process. In this case, the DDS `create_participant` operation will implicitly start the OpenSplice Domain Service as a thread in the existing application process. The OpenSplice Domain Service will then also implicitly start all services specified in the configuration as threads within the same process.

Single Process Application Cluster - This provides the option to co-locate multiple DDS applications into a single process. This can be done by creating application libraries rather than application executables that can be 'linked' into the single process in a similar way to how the DDS middleware services are linked into the single process. The applications that are created as libraries must be described using the *Domain/Application* (see section *4.2.10* on page 111) configuration attribute. These are started as threads within the existing process by the Domain Service *after* all the DDS services that are specified have been started as threads.

Please note that the *`Application`* elements (see section *4.2.10*) specified under *`Domain`* will only take effect for either mode when this *`SingleProcess`* attribute has a value of `True`.

`Full path`	OpenSplice/Domain/SingleProcess
`Format`	enumeration
`Dimension`	n/a
`Default value`	`False`
`Valid values`	`True, False`
`Occurrences (min-max)`	0 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.8 Element *Database*

The Database element contains information about the nodal administration (shared memory) to be used.

⚠ Please note that the Database element is only applicable in the shared memory deployment mode, *i.e.* when the *Element SingleProcess* is set to `False`, or is not specified.

`Full path`	OpenSplice/Domain/Database
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Size* *Element Threshold* *Element Address* *Element Locking*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.8.1 Element *Size*

This element specifies the size of the shared memory segment holding the database. Change this value if your system requires more memory than the default. Please note that the operating system must be configured to support the requested size. On most platforms you need 'root' or 'administrator' privileges to set large sizes.

The human-readable option lets the user postfix the value with *`K`*(ilobyte), *`M`*(egabyte) or *`G`*(igabtye). For example, *`10M`* results in 10485760 bytes.

`Full path`	OpenSplice/Domain/Database/Size
`Format`	unsigned integer, human-readable
`Dimension`	bytes
`Default value`	10485760
`Valid values`	0 – maxInt
`Occurrences (min-max)`	1 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.8.2 Element *Threshold*

This element specifies the threshold size used by OpenSplice DDS. Whenever there is less free shared memory than indicated by the threshold then no new allocations will be allowed within shared memory. Services are allowed to continue allocating shared memory until less than 50% of the threshold value is available.

The human-readable option lets the user postfix the value with *`K`*(ilobyte), *`M`*(egabyte) or *`G`*(igabtye). For example, *`10M`* results in 10485760 bytes.

It is strongly discouraged to configure a threshold value of less than the default value, but for some embedded systems it might be needed as only limited memory is available.

`Full path`	OpenSplice/Domain/Database/Threshold
`Format`	unsigned integer, human-readable
`Dimension`	bytes
`Default value`	1048576
`Valid values`	0 – maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.8.3 Element *Address*

This element specifies the start address where the nodal shared administration is mapped into the virtual memory of each process that attaches to the current Domain. The possible values are platform dependent.

`Full path`	OpenSplice/Domain/Database/Address
`Format`	(hexadecimal) memory address
`Dimension`	shared memory address

Default value	0x20000000 (Linux2.6 on x86) 0x140000000 (Linux2.6 on x86_64) 0x40000000 (Windows on x86) 0x40000000 (Windows on x86_64) 0xA0000000 (Solaris on SPARC) 0xA0000000 (AIX5.3 on POWER5+) 0x0 (VxWorks 5.5.1 on PowerPC604) 0x60000000 (VxWorks 6.x on PowerPC604) 0x20000000 (Integrity on mvme5100)
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

Change this value if the default address is already in use, for example by another Domain Service or another product.

4.2.8.4 Element *Locking*

This element specifies the locking policy of the Database, indicating whether to lock pages in physical memory or not.

With the virtual memory architecture, the operating system decides when to swap memory pages from internal memory to disc. This results in execution delays for the corresponding code because it has to be paged back into main memory. The element *Locking* can be used to avoid such swapping for the shared memory where the database resides. The user needs the appropriate privileges from the underlying operating system to be able to use this option.

The possible values are:

- **True**: lock the pages in memory.
- **False**: don't lock the pages in memory.
- **Default**: use the platform-dependent default value.

Full path	OpenSplice/Domain/Database/Locking
Format	enumeration
Dimension	n/a

`Default value`	Default
`Valid values`	True, False, Default
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9 Element *Service*

The Domain Service is responsible for starting, monitoring and stopping the pluggable services. One Service element must be specified for every service that needs to be started by the Domain Service.

- When run in shared memory mode, the Domain Service will start each service as a new process that will interface directly with the shared memory for DDS communication.
- When run in single process mode, the Domain Service will start each service as a new thread within the existing process that will have access to the heap memory for the DDS communication.

`Full path`	OpenSplice/Domain/Service
`Occurrences (min-max)`	1 - *
`Child-elements`	*Element Command* *Element MemoryPoolSize* *Element HeapSize* *Element StackSize* *Element Configuration* *Element Scheduling* *Element Locking* *Element FailureAction*
`Required attributes`	*Attribute name*
`Optional attributes`	*Attribute enabled*

4.2.9.1 Attribute name

This attribute specifies the name by which the the corresponding service is identified in the rest of the configuration file.

In the OpenSplice DDS configuration file, services and their settings are identified by a name. When the Domain Service starts a particular service, its corresponding name is passed. The service in question uses this name in order to find its own configuration settings in the rest of the configuration file. The name specified here must match the *name* attribute of the main element of the corresponding service.

`Full path`	OpenSplice/Domain/Service[@name]
`Format`	string
`Dimension`	n/a
`Default value`	durability
`Valid values`	any string
`Required`	yes

4.2.9.2 Attribute enabled

This attribute indicates whether the service is actually started or not.

`Full path`	OpenSplice/Domain/Service[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	no

Toggling a service between enabled and disabled is a quick alternative for commenting out the corresponding lines in the configuration file.

4.2.9.3 Element *Command*

This element specifies the command to be executed in order to start the service.

OpenSplice DDS comes with a set of pluggable services.

In *shared memory* mode, Command element specifies the name of the actual service executable (possibly including its path, but always including its extension, *e.g.* '.exe' on the Windows platform). When no path is included, the Domain Service will search the *PATH* environment variable for the corresponding executable. Once located, it will be started as a separate process.

In *single process* mode, Command is the name of the entry point function to be invoked *and* the name of the shared library to be dynamically loaded into the process. The signature of the entry point function is the same as `argc/argv` usually seen with `main`. The OpenSplice services are implemented in such a way that the entry point name matches that of the shared library, so (for example) specifying ‘`durability`’ is all that is required.

`Full path`	OpenSplice/Domain/Service/Command
`Format`	string
`Dimension`	executable file or entry point / shared library
`Default value`	durability
`Valid values`	The name of a service executable or entry point / shared library
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9.4 Element *MemoryPoolSize*

CAUTION: This element should only be used on the GHS Integrity platform.

This element maps directly into the integrate file for the address space for this service. Consult the GHS Integrate documentation for further information on this setting. Valid values are decimal or hexadecimal numbers and they express the number of bytes. The default setting for this element is dependent on the service for which it is configured.

`Full path`	OpenSplice/Domain/Service/MemoryPoolSize
`Format`	string
`Dimension`	decimal or hexadecimal number of bytes.
`Default value`	0xa00000 for spliced 0x280000 for durability 0x280000 for networking 0x100000 for cmsoap
`Valid values`	dependent on underlying platform
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9.5 Element *HeapSize*

CAUTION: This element should only be used on the GHS Integrity platform.

This element maps directly into the integrate file for the address space for this service. Consult the GHS Integrate documentation for further information on this setting. Valid values are decimal or hexadecimal numbers and they express the number of bytes. The default setting for this element is dependent on the service for which it is configured.

`Full path`	OpenSplice/Domain/Service/MemoryPoolSize
`Format`	string
`Dimension`	decimal or hexadecimal number of bytes.
`Default value`	0x800000 for spliced 0x240000 for durability 0x240000 for networking 0x200000 for cmsoap
`Valid values`	dependent on underlying platform
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9.6 Element *StackSize*

CAUTION: This element should only be used on the GHS Integrity platform.

This element maps directly into the integrate file for the address space for this service. Consult the GHS Integrate documentation for further information on this setting. Valid values are decimal or hexadecimal numbers and they express the number of bytes. The default setting for this element is dependent on the service for which it is configured.

`Full path`	OpenSplice/Domain/Service/StackSize
`Format`	string
`Dimension`	decimal or hexadecimal number of bytes.

`Default value`	0x10000 for spliced 0x10000 for durability 0x10000 for networking 0x10000 for cmsoap
`Valid values`	dependent on underlying platform
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9.7 Element *Configuration*

This element allows overriding of the default URI (specified in the *OSPL_URI* environment variable, or passed explicitly as command-line parameter to the *ospl* executable) with the configuration resource specified here.

When the Domain Service is started by the *ospl* executbale, by default it passes on its own URI to the services that it starts. This is valid when the configuration of the service is located in the same resource file as the configuration of the Domain Service itself. (This is a convenient situation in most cases).

If the configuration of the current service is located in a separate resource file, a separate URI identifying that particular resource file must be specified in this element.

`Full path`	OpenSplice/Domain/Service/Configuration
`Format`	string
`Dimension`	URI
`Default value`	${OSPL_URI}
`Valid values`	Any URI to a valid resource file.
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.9.8 Element *Scheduling*

This element specifies the scheduling parameters used to control the current Service.

Full path	OpenSplice/Domain/Service/Scheduling
Occurrences (min-max)	0 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.2.9.8.1 Element *Class*

This element specifies the thread scheduling class that the Domain Service will assign to the current Service when it is started. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/Domain/Service/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.9.8.2 Element *Priority*

This element specifies the thread priority that the Domain Service will assign to the current Service when it is started. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/Domain/Service/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system

Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.2.9.8.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/Domain/Service/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.2.9.9 Element *Locking*

This element specifies the locking policy of the current Service process, indicating whether pages should be locked in physical memory or not.

On platforms with a virtual memory architecture, the operating system decides when to swap memory pages from internal memory to disk. This results in execution delays for the corresponding code because it has to be paged back into main memory. The element *Locking* can be used to avoid such swapping for the current Service. The user needs the appropriate privileges from the underlying operating system to be able to use this option.

Full path	OpenSplice/Domain/Service/Locking
Format	boolean
Dimension	n/a
Default value	depends on operating system
Valid values	true, false
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.9.10 Element *FailureAction*

This element specifies what action to take at the moment that the service seems to have become non-responsive.

Each service reports its liveliness regularly using the shared administration. If the service fails to do so, the Domain Service will assume the service has become non-responsive. This element determines what action is taken by the Domain Service in case this happens.

The following actions are available:

- **skip**: Ignore the non-responsiveness and continue.
- **kill**: End the service process by force.
- **restart**: End the service process by force and restart it.
- **systemhalt**: End all OpenSplice services including the Domain Service (for the current DDS Domain on this computing node).

Full path	OpenSplice/Domain/Service/FailureAction
Format	enumeration
Dimension	n/a
Default value	skip
Valid values	skip, kill, restart, systemhalt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.10 Element *Application*

When in single process mode, the Domain service can deploy DDS applications by dynamically loading application shared libraries and starting threads within the existing process.

A user can add a multiple Application elements to the configuration when they want to ‘cluster’ multiple DDS applications within an OpenSplice DDS single process. The entry point and shared library for each Application can be specified by using the Command and Library elements that are described below.

⚠ Note that Applications only take effect when the *SingleProcess* configuration is enabled (see 4.2.7, *Element SingleProcess*, on page 100).

`Full path`	OpenSplice/Domain/Application
`Occurrences (min-max)`	0 - *
`Child-elements`	*Element Command* *Element Library* *Element Arguments*
`Required attributes`	*Attribute name*
`Optional attributes`	*Attribute enabled*

4.2.10.1 Attribute name

This attribute assigns a configuration label to the application, but it is of no further use; it can have any valid string value.

`Full path`	OpenSplice/Domain/Application[@name]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	any string
`Required`	yes

4.2.10.2 Attribute enabled

This attribute indicates whether the application is actually started or not.

`Full path`	OpenSplice/Domain/Application[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	no

Toggling an application between enabled and disabled is a quick alternative for commenting out the corresponding lines in the configuration file.

4.2.10.3 Element *Command*

Command is the name of both the entry point function to be invoked and of the shared library to be dynamically loaded into the process. The signature of the entry point function is the same as `argc/argv` usually seen with `main`.

For example, if Command is ‘`HelloWorld`’, then OpenSplice DDS will attempt to load `libHelloWorld.so` (on Unix) or `HelloWorld.dll` (on Windows) into the existing process and then invoke the ‘`HelloWorld`’ entry point to start that DDS application.

If the name of the shared library does not have the same name as the entry point, the user can override the name of the library by using the application’s Library element (see Section 4.2.10.4, *Element Library*). The shared library is located by way of the current working directory, or *via* `LD_LIBRARY_PATH` (on Unix systems) or `PATH` (on Windows systems).

Note that this has the same meaning as the Service/Command element when in the single process mode of operation.

`Full path`	OpenSplice/Domain/Application/Command
`Format`	string
`Dimension`	executable file
`Default value`	durability
`Valid values`	The name of a service executable.
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.10.4 Element *Library*

This optional attribute allows the user to override the name of the shared library if it is different from the name of the entry point specified by Command (see also Section 4.2.10.3, *Element Command*).

The shared library is located by way of the current working directory, or *via* `LD_LIBRARY_PATH` (on Unix systems) or `PATH` (on Windows systems).

`Full path`	OpenSplice/Domain/Application/Library
`Format`	string
`Dimension`	Library file
`Default value`	“” (empty string)
`Valid values`	the name of a library file
`Occurrences (min-max)`	0 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.10.5 Element *Arguments*

This optional element enables the user to specify arguments to be passed to the DDS application's entry point when it is invoked. For example, if `Command` is *`"HelloWorld"`* and `Arguments` is *`"arg1 arg2"`*, OpenSplice will invoke the `HelloWorld` function with the `argc = 3` and `argv = {"HelloWorld", "arg1", "arg2"}`.

`Full path`	OpenSplice/Domain/Application/Arguments
`Format`	string
`Dimension`	n/a
`Default value`	<none>
`Valid values`	any string
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.11 Element *Listeners*

This element specifies policies for the thread that services the listeners that the application specifies on the API-level.

`Full path`	OpenSplice/Domain/Listeners
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element StackSize*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.11.1 Element *StackSize*

This element specifies stack size of the listener thread.

`Full path`	OpenSplice/Domain/Service/Listeners/StackSize
`Format`	unsigned integer
`Dimension`	bytes
`Default value`	64000
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

⚠ Newer versions of JDK (at least 1.6u43 and 1.6u45) run out of stack space on 64-bit platforms. Using a larger default `StackSize` would impact all non-Java applications too, and is therefore undesirable. Try increasing `StackSize` to `128000` bytes if you experience problems with using listeners from Java on 64-bit platforms.

4.2.12 Element *BuiltinTopics*

This element specifies the granularity of the builtin topics.

`Full path`	OpenSplice/Domain/BuiltinTopics
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute enabled*
`Optional attributes`	<none>

4.2.12.1 Attribute enabled

This attribute enables or disables the publication of builtin topics for the existence of individual Participants/DataWriters/DataReaders. The existence of Topics will always be communicated by means of built-in topics, regardless of the value specified here.

`Full path`	OpenSplice/Domain/BuiltinTopics[@enabled]
`Format`	boolean
`Dimension`	n/a

`Default value`	true
`Valid values`	true, false
`Required`	true

4.2.13 Element *PriorityInheritance*

This element specifies the usage on Priority Inheritance on mutexes in this domain.

`Full path`	OpenSplice/Domain/PriorityInheritance
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute enabled*
`Optional attributes`	<none>

4.2.13.1 Attribute enabled

This attribute enables or disables priority inheritance for mutexes, if that is supported by the underlying Operating System.

`Full path`	OpenSplice/Domain/PriorityInheritance[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	true

4.2.14 Element *Statistics*

This element specifies the policies regarding statistics. Various statistics can be generated by OpenSplice DDS to help you analyze and tune application behaviour during application development. Since this introduces extra overhead, it is generally turned off in a runtime system.

`Full path`	OpenSplice/Domain/Statistics
`Occurrences (min-max)`	0 - 1

Child-elements	*Element Category*
Required attributes	<none>
Optional attributes	<none>

4.2.14.1 Element *Category*

This element specifies the properties for a particular category of statistics.

Full path	OpenSplice/Domain/Statistics/Category
Occurrences (min-max)	0 - *
Child-elements	<none>
Required attributes	*Attribute name*
Optional attributes	*Attribute enable*

4.2.14.1.1 Attribute name

This attribute specifies the name of a particular category of statistics.

Full path	OpenSplice/Domain/Statistics/Category[@name]
Format	string
Dimension	name of a statistics category
Default value	reader
Valid values	durability, reader, writer, networking
Required	true

4.2.14.1.2 Attribute enable

This attribute enables or disables the generation of statistics for the specified category.

Full path	OpenSplice/Domain/Statistics/Category[@enabled]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.2.15 Element *ReportPlugin*

This element allows the user to plugin a custom report facility to be used by the domain. The Domain Service is responsible for registering and unregistering any report plugins. All services and applications within the domain will use any registered report plugins.

`Full path`	OpenSplice/Domain/ReportPlugin
`Occurrences (min-max)`	0 – 10
`Child-elements`	*Element Library* *Element Initialize* *Element Report* *Element Finalize* *Element SuppressDefaultLogs*
`Required attributes`	<none>
`Optional Attributes`	<none>

4.2.15.1 Element *Library*

This element specifies the library to be loaded.

`Full path`	OpenSplice/Domain/ReportPlugin/Library
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute file_name*
`Optional attributes`	<none>

4.2.15.1.1 Attribute file_name

This attribute specifies the name of the library to be loaded. The attribute is required, if no file name is specified or a name is specified but the library cannot be loaded an error message is generated; the service will not attempt to look up any other elements and the report plugin details will not be registered.

`Full path`	OpenSplice/Domain/ReportPlugin/ Library[@file_name]
`Format`	String
`Dimension`	N/A

`Default value`	None
`Valid values`	Any valid string
`Required`	Yes

4.2.15.2 Element *Initialize*

This element specifies the library symbol that will be assigned to the report Initialize operation. This operation will be invoked initially after loading the library to perform initialization of the report facility if needed.

`Full path`	OpenSplice/Domain/ReportPlugin/Initialize
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute symbol_name*
`Optional attributes`	*Attribute argument*

4.2.15.2.1 Attribute symbol_name

This attribute specifies the name of the function to be called to initialize the report plugin. The `symbol_name` is required, if it is not specified or cannot be resolved an error message will be generated and the service will not attempt to resolve other `symbol_names` for the report plugin.

The implementation of this function must have the following signature:

```
int symbol_name (const char *argument, void **context)
```

The result value is used to return the status of the call. If `0` then the operation was successful. If not `0` then there was an error and details of the error and the result value are reported to the OpenSplice DDS default report service.

The context parameter is an out reference that can be set to plugin-specific data that will subsequently be passed to any of the other plugin functions. The value of the parameter is meaningless to the service.

`Full path`	OpenSplice/Domain/ReportPlugin/ Initialize[@symbol_name]
`Format`	String
`Dimension`	N/A
`Default value`	None
`Valid values`	Any valid string
`Required`	Yes

4.2.15.2.2 Attribute argument

The argument attribute is a string value that is passed to the function specified by the symbol_name. The string value has no meaning to the service and is used to pass any context-specific information that may be required. The argument is optional; if it is not provided then a NULL pointer is passed to the initalize function.

`Full path`	OpenSplice/Domain/ReportPlugin/ Initialize[@argument]
`Format`	String
`Dimension`	N/A
`Default value`	None
`Valid values`	Any valid string
`Required`	No

4.2.15.3 Element *Report*

This element specifies the library symbol that will be assigned to the report Report operation. This operation will be invoked on all reports performed by the DDS service.

`Full path`	OpenSplice/Domain/ReportPlugin/Report
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute symbol_name*
`Optional attributes`	<none>

4.2.15.3.1 Attribute symbol_name

This attribute specifies the name of the function to be called to pass report data to the report plugin. The `symbol_name` is required, if it is not specified or cannot be resolved an error message will be generated and the service will not attempt to resolve other `symbol_names` for the report plugin.

The implementation of this function must have the following signature:

```
int symbol_name (void *context, const char *report)
```

The result value is used to return the status of the call. If `0` then the operation was successful. If not `0` then there was an error and details of the error and the result value are reported to the OpenSplice DDS default report service.

The context parameter is a reference to the plugin-specific data retrieved from the initialize operation.

The report parameter is an XML string representation of the report data.

Below is an example of the mapping that the XML string representation will use:

```
<WARNING>
   <DESCRIPTION>The object “my_topic” not
found</DESCRIPTION>
   <CONTEXT>c_base::resolve</CONTEXT>
   <FILE>c_base.c</FILE>
   <LINE>1234</LINE>
   <CODE>0</CODE>
</WARNING>
```

Full path	OpenSplice/Domain/ReportPlugin/ Report[@symbol_name]
Format	String
Dimension	N/A
Default value	None
Valid values	Any valid string
Required	Yes

4.2.15.4 Element TypedReport

This element can be specified additionally or as an alternative to element Report (see Section 4.2.15.3, *Element Report*, on page 120). It specifies a a library symbol for a fully defined and less generic report operation that will be called by the plug-in framework for all loggable events.

Full path	OpenSplice/Domain/ReportPlugin/TypedReport
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	*Attribute symbol_name*
Optional attributes	<none>

4.2.15.4.1 Attribute symbol_name

This attribute specifies the name of the function to be called to pass report data to the report plugin.

The implementation of this function must have the following signature:

```
int symbol_name (void *context, os_reportEvent report);
```

where `os_reportEvent` is a pointer to a struct holding the data that can be logged.

The definition of this log event struct can be found in the file `$OSPL_HOME/include/sys/os_report.h` and this file should be consulted for further details. This header file should be included (`#include`) into plug-in source code and plug-ins should then be linked against the library `libddsos.so` / `ddsos.lib` if a TypedReport report operation is to be used. As with the Report element, the context parameter is a reference to the plugin-specific data retrieved from the initialize operation.

An example of implementing a plug-in with this form of log function to access this data can be found in the directory `$OSPL_HOME/examples/utilities/logging/log4cplugin`.

`Full path`	OpenSplice/Domain/ReportPlugin/ TypedReport[@symbol_name]
`Format`	String
`Dimension`	N/A
`Default value`	None
`Valid values`	Any valid string
`Required`	Yes

4.2.15.5 Element *Finalize*

This element specifies the library symbol that will be assigned to the report Finalize operation. This operation will be invoked upon process termination to perform de-initialization of the report facility if needed.

`Full path`	OpenSplice/Domain/ReportPlugin/Finalize
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute symbol_name*
`Optional attributes`	<none>

4.2.15.5.1 Attribute symbol_name

This attribute specifies the name of the function to be called to finalize the report plugin, when the domain unregisters any registered plugin. The `symbol_name` is required. If it is not specified or cannot be resolved an error message will be generated and the service will not attempt to resolve other `symbol_names` for the report plugin.

The implementation of this function must have the following signature:

```
int symbol_name (void *context)
```

The result value is used to return the status of the call. If `0` then the operation was successful. If not `0` then there was an error and details of the error and the result value are reported to the OpenSplice DDS default report service.

The context parameter is a reference to the plugin-specific data retrieved from the `initialize` operation.

`Full path`	OpenSplice/Domain/ReportPlugin/ Report[@symbol_name]
`Format`	String
`Dimension`	N/A
`Default value`	None
`Valid values`	Any valid string
`Required`	Yes

4.2.15.6 Element *SuppressDefaultLogs*

This attribute specifies whether the default logs generated by the domain are to be suppressed or not. The default value is `False`. If the value is set to `True` then ospl-error and ospl-info logs normally generated by the domain will not be generated.

`Full path`	OpenSplice/Domain/ReportPlugin/ SuppressDefaultLogs
`Format`	boolean
`Dimension`	n/a
`Default value`	False
`Valid value`	True, False
`Occurrences (min-max)`	0 – 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional Attributes`	<none>

4.2.16 Element *PartitionAccess*

This element is used to configure the partition access rights. By default all partitions have read and write access, which means that subscribers and publishers may be created for all partitions. However by changing the access level of specific partitions it is possible to prevent publishers and/or subscribers from attaching to these partitions. The access rights is Domain Service specific, each Domain Service can have its own policy.

The `PartitionAccess` element facilitates the configuration of such behavior. This is done by allowing the definition of a partition expression along with a specific access mode for the matched partitions. The `PartitionAccess` element resides as a child element within the `Domain` element. The exact definition of the `PartitionAccess` element is as follows:

Full path	OpenSplice/Domain/PartitionAccess
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	*Attribute partition_expression* *Attribute access_mode*
Optional attributes	<none>

An example demonstrates the usage of this element:

```
<OpenSplice>
   <Domain>
    <PartitionAccess partition_expression=
             ”/remote/*” access_mode=”readwrite”/>
   </Domain>
</OpenSplice>
```

4.2.16.1 Attribute partition_expression

This attribute identifies an expression that specifies the partitions for which access is being defined. The expression may use wildcards (*i.e.* the ‘*’ and ‘?’ tokens) to indicates multiple partitions that match the expression.

Full path	OpenSplice/Domain/PartitionAccess/ partition_expression
Format	string
Dimension	n.a.
Default value	*
Valid values	Any string of format [A..Z,a..z,0..9,_,/,*,?]*
Required	true

4.2.16.2 Attribute access_mode

This attribute identifies the access level for partitions specified by the `partition_expression` attribute. The following values are allowed:

`read`	Indicates domain participants can only read from this partition
`write`	Indicates domain participants can only write to this partition
`readwrite`	Indicates domain participants can read from and write to this partition
`none`	Indicates that domain participants have no access on partitions matching the partition_expression.

When multiple expressions overlap each other, the following rules are applied:

Access mode 1	Access mode 2	Resulting access mode
`read`	`write`	`readwrite`
`read`	`readwrite`	`readwrite`
`read`	`none`	`none`
`write`	`readwrite`	`readwrite`
`write`	`none`	`none`
`readwrite`	`none`	`none`

`Full path`	OpenSplice/Domain/PartitionAccess/access_mode
`Format`	string
`Dimension`	n.a.
`Default value`	readwrite
`Valid values`	read, write, readwrite, none
`Required`	true

4.2.17 Element *TopicAccess*

This element is used to configure the topic access rights. By default all topics have read and write access (built-in topics have a default access mode of read), which means that datareaders and datawriters may be created for all topics. However by changing the access level of specific topics it is possible to prevent datawriters and/or datareaders from being created for these topics. The access rights is Domain Service specific, each Domain Service can have its own policy.

The `TopicAccess` element facilitates the configuration of such behavior. This is done by allowing the definition of a topic expression along with a specific access mode for the matched topics.

The `TopicAccess` element resides as a child element within the `Domain` element. The exact definition of the `TopicAccess` element is as follows:

`Full path`	OpenSplice/Domain/TopicAccess
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute topic_expression* *Attribute access_mode*
`Optional attributes`	<none>

An example demonstrates the usage of this element:

```
<OpenSplice>
   <Domain>
      <TopicAccess topic_expression=
             ”/remote/*” access_mode=”read”/>
   </Domain>
</OpenSplice>
```

4.2.17.1 Attribute topic_expression

This attribute identifies an expression that specifies the topics for which access is being defined. The topic may use wildcards (*i.e.* the ‘`*`’ and ‘`?`’ tokens) to indicate multiple topics that match the expression.

`Full path`	OpenSplice/Domain/TopicAccess/topic_expression
`Format`	string
`Dimension`	n.a.
`Default value`	`*`
`Valid values`	Any string
`Required`	true

4.2.17.2 Attribute access_mode

This attribute identifies the access level for topics defined by the `topic_expression` attribute. The following values are allowed:

`read`	Indicates that domain participants have only read access on Topics matching the `topic_expression`.

write	Indicates that domain participants have only write access on Topics matching the `topic_expression`.
readwrite	Indicates that domain participants have read and write access on Topics matching the `topic_expression`.
none	Indicates that domain participants have no access to Topics matching the `topic_expression`.

When multiple expressions overlap each other, the following rules are applied:

Access mode 1	Access mode 2	Resulting access mode
`read`	`write`	`readwrite`
`read`	`readwrite`	`readwrite`
`read`	`none`	`none`
`write`	`readwrite`	`readwrite`
`write`	`none`	`none`
`readwrite`	`none`	`none`

`Full path`	OpenSplice/Domain/TopicAccess/access_mode
`Format`	string
`Dimension`	n.a.
`Default value`	readwrite
`Valid values`	read, write, readwrite, none
`Required`	true

4.2.18 Element *ResourceLimits*

This configuration tag allows for the specification of certain characteristics of resource limits that will be applied throughout the domain.

`Full path`	OpenSplice/Domain/ResourceLimits
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element MaxSamples* *Element MaxInstances* *Element MaxSamplesPerInstance*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.18.1 Element *MaxSamples*

This configuration tag allows for the specification of certain characteristics of the maximum samples resource limit that will be applied throughout the domain.

`Full path`	OpenSplice/Domain/ResourceLimits/MaxSamples
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element WarnAt*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.18.1.1 Element *WarnAt*

This element specifies the number of samples that, once reached, will result in a warning message printed in the info log. This is to allow the detection of excessive use of resources within the domain more easily.

`Full path`	OpenSplice/Domain/ResourceLimits/MaxSamples/ WarnAt
`Format`	unsigned integer
`Dimension`	n.a.
`Default value`	5000
`Valid values`	1-maxInt
`Occurrences (min-max)`	0 - 1
`Child Elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.18.2 Element *MaxInstances*

This configuration tag allows for the specification of certain characteristics of the maximum instances resource limit that will be applied throughout the domain

`Full path`	OpenSplice/Domain/ResourceLimits/MaxInstances
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element WarnAt*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.18.2.1 Element *WarnAt*

This element specifies the number of instances that, once reached, will result in a warning message printed in the info log. This is to allow the detection of excessive use of resources within the domain more easily.

Full path	OpenSplice/Domain/ResourceLimits/MaxInstances/WarnAt
Format	unsigned integer
Dimension	n.a.
Default value	5000
Valid values	1-maxInt
Occurrences (min-max)	0 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.18.3 Element *MaxSamplesPerInstance*

This configuration tag allows for the specification of certain characteristics of the maximum samples resource limit that will be applied throughout the domain.

Full path	OpenSplice/Domain/ResourceLimits/MaxSamplesPerInstance
Occurrences (min-max)	0 - 1
Child-elements	*Element WarnAt*
Required attributes	<none>
Optional attributes	<none>

4.2.18.3.1 Element *WarnAt*

This element specifies the number of samples per instance that, once reached, will result in a warning message printed in the info log. This is to allow the detection of excessive use of resources within the domain more easily.

Full path	OpenSplice/Domain/ResourceLimits/MaxSamplesPerInstance/WarnAt
Format	unsigned integer
Dimension	n.a.

`Default value`	5000
`Valid values`	1-maxInt
`Occurrences (min-max)`	0 - 1
`Child Elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.19 Element *Report*

The Report element controls some aspects of the OpenSplice domain logging functionality.

`Full path`	OpenSplice/Domain/Report
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute append* *Attribute verbosity*

4.2.19.1 Attribute append

This attribute determines whether logging for this domain should continue to append to the previous error and info log files when the domain is (re)started or whether the previous file should be deleted and fresh ones created.

`Full path`	OpenSplice/Domain/Report[@append]
`Format`	string
`Dimension`	n/a
`Default value`	`true`

`Valid values`	`0/no/false` *or* `1/yes/true`
	`(case-insensitive)`

`Required`	`no`

4.2.19.2 Attribute verbosity

This attribute determines what level of logging should be in effect for this domain. The levels or logging verbosity are:

```
0  DEBUG
```

```
1   INFO
2   WARNING
3   API_INFO
4   ERROR
5   CRITICAL
6   FATAL
7   REPAIRED
8   NONE
```

The level specified as this attribute is the lowest level that will be emitted to the logs. All logging can be suppressed by specifying the value `8` or `NONE`.

`Full path`	OpenSplice/Domain/Report[@verbosity]
`Format`	integer *or* string
`Dimension`	n/a
`Default value`	`INFO` *or* `1`
`Valid values`	`integer 0 - 8` *or* `DEBUG, INFO, WARNING, API_INFO, ERROR, CRITICAL, FATAL, REPAIRED, NONE` *(case-insensitive)*
`Required`	`no`

4.2.20 Element *Daemon*

Every domain is controlled by exactly one daemon: the Splice Daemon. The Splice Daemon configuration expects a root element named *`OpenSplice/Domain/Daemon`*. Within this root element, the Splice Daemon will look for several child elements. Each of these child elements is listed and explained in the following sections.

`Full path`	OpenSplice/Domain/Daemon
`Occurrences (min-max)`	0 - 1

`Child-elements`	*Element Locking* *Element KernelManager* *Element GarbageCollector* *Element ResendManager* *Element Watchdog* *Element Heartbeat*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.1 Element *Locking*

This element specifies the locking policy for the Splice Deamon process, indicating whether its pages should be locked in physical memory or not.

On platforms with a virtual memory architecture, the operating system decides when to swap memory pages from internal memory to disk. This results in execution delays for the corresponding code because it has to be paged back into main memory. The element *Locking* can be used to avoid such swapping for the Splice Daemon. The user needs the appropriate privileges from the underlying operating system to be able to use this option.

`Full path`	OpenSplice/Domain/Daemon/Locking
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.2 Element *KernelManager*

The Kernel Manager monitors Builtin Topic on status changes of DataWriters and inconsistencies between Topics and QoS policies, and it will notify all participants interested in any of these events, *i.e.* it updates status fields and wakeup blocking waitset and listener threads.

Controlling the scheduling behaviour of the Kernel Manager will therefore infuence the reactivity on detecting events, but it will not infuence the event handling itself as this is the responsibility of the participants waitset or listener thread.

Note that the Kernel Manager has no or limited value when Builtin Topics are disabled.

`Full path`	OpenSplice/Domain/Daemon/KernelManager
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.2.1 Element *Scheduling*

This element specifies the scheduling policies used to control the KernelManager thread.

`Full path`	OpenSplice/Domain/Daemon/KernelManager/ Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.2.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the KernelManager thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/Domain/Daemon/KernelManager/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.20.2.1.2 Element *Priority*

This element specifies the thread priority that will be used by the KernelManager thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/Domain/Daemon/KernelManager/Scheduling/Priority
Format	integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.2.20.2.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/Domain/Daemon/KernelManager/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.2.20.3 Element *GarbageCollector*

This element specifies the behaviour of the GarbageCollector.

The Garbage Collector is a safety mechanism and is responsible for reclaiming resources and correcting communication statusses in case an application or a remote Domain Service does not terminate properly or communication fails.

`Full path`	OpenSplice/Domain/Daemon/GarbageCollector
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.3.1 Element *Scheduling*

This element specifies the scheduling policies used to control the GarbageCollector thread.

`Full path`	OpenSplice/Domain/Daemon/GarbageCollector/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.3.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the GarbageCollector thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/Domain/Daemon/GarbageCollector/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.20.3.1.2 Element *Priority*

This element specifies the thread priority that will be used by the GarbageCollector thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/Domain/Daemon/GarbageCollector/Scheduling/Priority
Format	integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.2.20.3.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/Domain/Daemon/GarbageCollector/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.2.20.4 Element *ResendManager*

The Domain Service has a resend manager that is responsible for resending builtin Topic data that has not been sent successfully on the first attempt (for example, because of temporarily unavailable resources, such as when there is no space in the network send queue due to a temporary overload). In such a situation the data will be stored in the DataWriters history and the resend manager will periodically try to resend the data until it succeeds.

Full path	OpenSplice/Domain/Daemon/ResendManager
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.2.20.4.1 Element *Scheduling*

This element specifies the type of operating system scheduling class will be used by the thread that does local resends for the built-in participant.

Full path	OpenSplice/Domain/Daemon/ResendManager/Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.2.20.4.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the ResendManager thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/Domain/Daemon/ResendManager/Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.4.1.2 Element *Priority*

This element specifies the thread priority that will be used by the ResendManager thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/Domain/Daemon/ResendManager/Scheduling/Priority
`Format`	integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.2.20.4.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/Domain/Daemon/ResendManager/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.2.20.5 Element *Watchdog*

This element controls the scheduling characteristics of the Watchdog thread. This thread is responsible for sending domain service heartbeats, updating liveliness of the service builtin DataWriters and monitoring the health of internal services and heartbeats of remote domain services.

Full path	OpenSplice/Domain/Daemon/Watchdog
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.2.20.5.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

Full path	OpenSplice/Domain/Daemon/Watchdog/Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.2.20.5.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes. The Default value is controlled by the GeneralWatchdog settings (see section *4.2.21.1.1* on page 145).

Full path	OpenSplice/Domain/Daemon/Watchdog/ Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.20.5.1.2 Element *Priority*

This element specifies the thread priority that will be used by the watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities. The Default value is controlled by the GeneralWatchdog settings (see section *4.2.21.1.2* on page 145).

Full path	OpenSplice/Domain/Daemon/Watchdog/ Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.20.6 Element Heartbeat

The Splice Daemon uses an heartbeat mechanism to monitor the health of the remote domain services. This element allows fine-tuning of this heartbeat mechanism.

Please note this heartbeat mechanism is similar to but not the same as the service liveliness assertion.

Full path	OpenSplice/Domain/Daemon/Heartbeat
Occurrences (min-max)	0 - 1
Child-elements	*Element ExpiryTime* *Element Scheduling*
Required attributes	0
Optional attributes	*Attribute transport_priority*

4.2.20.6.1 Attribute transport_priority

This attribute controls the transport priority QoS setting that is only used by the Splice Daemon for for sending its heartbeats.

Full path	OpenSplice/Domain/Daemon/Heartbeat [@transport_priority]
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - maxInt
Required	false

4.2.20.6.2 Element *ExpiryTime*

This element specifies the maximum amount of time (in seconds) in which the Splice Daemon expects a new heartbeat of remote domain services. This is obviously also the same amount of time in which the Splice Daemon must send a heartbeat itself.

Increasing this value will lead to less networking traffic and overhead but also to less responsiveness with respect to the liveliness of the Splice Daemon. Change this value according to the needs of your system with respect to these aspects.

Full path	OpenSplice/Domain/Daemon/Heartbeat/ExpiryTime
Format	float
Dimension	seconds
Default value	10.0
Valid values	0.2 - maxFloat
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	*Attribute update_factor*
Optional attributes	<none>

4.2.20.6.2.1 Attribute update_factor

In case of a (temporary) high CPU load, the scheduling behaviour of the operating system might affect the capability of the Splice Daemon to send its heartbeat ‘on time’. This attribute introduces some elasticity in this mechanism by making the service send its heartbeat more often than required by the *ExpiryTime*.

The Splice Daemon will report its liveliness every *ExpiryTime* multiplied by this *update_factor.*

Full path	OpenSplice/Domain/Daemon/Heartbeat/ExpiryTime[@update_factor]
Format	float
Dimension	n/a
Default value	0.2
Valid values	0.1 - 0.9
Required	true

4.2.20.6.3 Element *Scheduling*

This element specifies the scheduling parameters used by the thread that periodically sends the heartbeats.

`Full path`	OpenSplice/Domain/Daemon/Heartbeat/Scheduling
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.6.3.1 Element *Class*

This element specifies the thread scheduling class that will be used by the thread that periodically sends the heartbeats. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/Domain/Daemon/Heartbeat/Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.20.6.3.2 Element *Priority*

This element specifies the thread priority that will be used by the thread that periodically sends the heartbeats. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/Domain/Daemon/Heartbeat/Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a

Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.2.20.6.3.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/Domain/Daemon/Heartbeat/ Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.2.21 Element *GeneralWatchdog*

This element controls the default scheduling characteristics of the Watchdog thread for all services. Individual services may overrule this default in their service-specific Watchdog settings.

Full path	OpenSplice/Domain/GeneralWatchdog
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.2.21.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

Full path	OpenSplice/Domain/GeneralWatchdog/Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.2.21.1.1 Element *Class*

This element specifies the default thread scheduling class that will be used by the watchdog thread of each service if not overruled by service specific settings. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/Domain/GeneralWatchdog/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.21.1.2 Element *Priority*

This element specifies the default thread priority that will be used by the watchdog thread of each service if not overruled by service specific settings. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/Domain/GeneralWatchdog/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system

Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.2.22 Element *UserClockService*

The UserClockService allows you to plug-in a custom clock library, allowing OpenSplice to read the time from an external clock source. It expects a root element named *`OpenSplice/Domain/UserClockService`*. Within this root element, the userclock will look for several child-elements. Each of these is listed and explained.

Full path	OpenSplice/Domain/UserClockService
Occurrences (min-max)	0 - 1
Child-elements	*Element UserClockModule* *Element UserClockStart* *Element UserClockStop* *Element UserClockQuery*
Required attributes	<none>
Optional attributes	<none>

4.2.22.1 Element *UserClockModule*

This element specifies the User Clock Service library file. On UNIX like and Windows platforms this will be a shared library. On VxWorks this will be a reallocatable object file. On VxWorks this tag may be empty or discarded if the functions are pre-loaded on the target.

Full path	OpenSplice/Domain/UserClockService/ UserClockModule
Format	string
Dimension	file name
Default value	n/a
Valid values	dependent on underlying operating system.
Occurrences (min-max)	1 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.2.22.2 Element *UserClockStart*

This element specifies if the user clock requires a start function to be called when the process first creates a participant.

`Full path`	OpenSplice/Domain/UserClockService/ UserClockStart
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.2.22.2.1 Attribute name

This attribute specifies the name of the start function. This start function should not have any parameters, and needs to return an int that represents 0 if there are no problems, and any other value if a problem is encountered.

`Full path`	OpenSplice/Domain/UserClockService/ UserClockStart[@name]
`Format`	string
`Dimension`	function name
`Default value`	clockStart
`Valid values`	name of any existing and accessible function
`Required`	true

4.2.22.3 Element *UserClockStop*

This element specifies if the user clock requires a stop function to be called when the process deletes the last participant.

`Full path`	OpenSplice/Domain/UserClockService/ UserClockStop
`Occurrences (min-max)`	0 - 1

Child-elements	<none>
Required attributes	*Attribute name*
Optional attributes	<none>

4.2.22.3.1 Attribute name

This attribute specifies the name of the stop function. This stop function should not have any parameters, and needs to return an int that represents 0 if there are no problems, and any other value if a problem is encountered.

Full path	OpenSplice/Domain/UserClockService/ UserClockStop[@name]
Format	string
Dimension	function name
Default value	clockStop
Valid values	name of any existing and accessible function
Required	true

4.2.22.4 Element *UserClockQuery*

This element specifies the clock query function.

Full path	OpenSplice/Domain/UserClockService/ UserClockQuery
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	*Attribute name*
Optional attributes	<none>

4.2.22.4.1 Attribute name

This attribute specifies the name of the function that gets the current time. This *clockGet* function should not have any parameters, and needs to return the current time as an *os_time* type.

The definition of the *os_time* type can be found in *os_time.h*:

```
typedef struct os_time {
    /** Seconds since 1-jan-1970 00:00 */
    os_timeSec tv_sec;
    /** Count of nanoseconds within the second */
    os_int32 tv_nsec;
    /** os_time can be used for a duration type with the following
```

```
        semantics for negative durations: tv_sec specifies the
        sign of the duration, tv_nsec is always possitive and added
        to the real value (thus real value is tv_sec+tv_nsec/10^9,
        for example { -1, 500000000 } is -0.5 seconds) */
} os_time;
```

`Full path`	OpenSplice/Domain/UserClockService/ UserClockQuery[@name]
`Format`	string
`Dimension`	function name
`Default value`	clockGet
`Valid values`	name of any existing and accessible function
`Required`	true

4.3 The Durability Service

The responsibilities of the durability service are to realize the durable properties of data in an OpenSplice system. The Durability Service looks for its configuration within the ‘OpenSplice/DurabilityService’ element. The configuration parameters that the Durability Service will look for within this element are listed and explained in the following subsections.

`Full path`	OpenSplice/DurabilityService/
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Network* *Element Persistent* *Element NameSpaces* *Element Watchdog* *Element EntityNames* *Element Tracing*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.3.1 Attribute name

This attribute uniquely identifies the configuration for the Durability Service. Multiple Durability Service configurations can be specified in one single resource so long as each has its own unique name. The actual applicable configuration is determined by the value of the *name* attribute, which matches the one specified under the attribute *OpenSplice/Domain/Service[@name]* in the configuration of the Domain Service.

Full path	OpenSplice/DurabilityService[@name]
Format	string
Dimension	n/a
Default value	durability
Valid values	any string
Required	true

4.3.2 Element *Network*

Applications need to be able to gain access to historical data in a system. When the local DDS service gets connected to a remote DDS service by means of the Network Service, (parts of) the historical data might not be consistent between the local and remote Durability Services. The Durability Service needs to be able to detect the other available Durability Services and exchange historical data with them to keep and/or restore consistency in historical data between them.

The *Network* element provides handles to fine-tune the behaviour of the communication between Durability Services on network level. These settings only apply when the Network Service is active.

Full path	OpenSplice/DurabilityService/Network
Occurrences (min-max)	0 - 1
Child-elements	*Element Heartbeat* *Element InitialDiscoveryPeriod* *Element Alignment* *Element WaitForAttachment*
Required attributes	<none>
Optional attributes	*Attribute latency_budget* *Attribute transport_priority*

4.3.2.1 Attribute latency_budget

This attribute controls the latency budget QoS setting that is used by the Durability Service for its communication with other Durability Services.

This is a hint for the service that it can delay messages for as long as the specified latency_budget to optimize bandwidth utilization. The Network Service can use this delay to pack several messages into a single network message. The default value is zero, meaning that messages are sent to the network immediately.

`Full path`	OpenSplice/DurabilityService/ Network[@latency_budget]
`Format`	float
`Dimension`	seconds
`Default value`	0.0
`Valid values`	0.0 - maxFloat
`Required`	false

4.3.2.2 Attribute transport_priority

This attribute specifies the transport priority for alignment communication between durability services. The Network Service will shedule inter-durability communication relative to other communication based on this transport priority.

For example, if the latency of timing-critical application data should not be disturbed by alignment activities between durability services, then this transport priority should be configured lower than the application policy.

`Full path`	OpenSplice/DurabilityService/ Network[@transport_priority]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - maxInt
`Required`	false

4.3.2.3 Element *Heartbeat*

During startup and at runtime, the network topology can change dynamically. This happens when OpenSplice services are started/stopped or when a network cable is plugged in/out. The Durability Services need to keep data consistency in that environment. To detect newly joining services as well as detecting services that are leaving, the Durability Service uses a heartbeat mechanism. This element allows fine-tuning of this mechanism.

Please note this heartbeat mechanism is similar to but not the same as the service liveliness assertion.

`Full path`	OpenSplice/DurabilityService/Network/Heartbeat
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element ExpiryTime* *Element Scheduling*
`Required attributes`	0
`Optional attributes`	*Attribute latency_budget* *Attribute transport_priority*

4.3.2.3.1 Attribute latency_budget

This attribute controls the latency budget QoS setting (in seconds) that is only used by the Durability Service for sending its heartbeats. It overrules the value of the *DurabilityService/Network[@latency_budget]*.

`Full path`	OpenSplice/DurabilityService/Network/ Heartbeat[@latency_budget]
`Format`	float
`Dimension`	seconds
`Default value`	0.0
`Valid values`	0.0 - maxFloat
`Required`	false

4.3.2.3.2 Attribute transport_priority

This attribute controls the transport priority QoS setting that is only used by the Durability Service for sending its heartbeats. It overrules the value of the *DurabilityService/Network[@transport_priorrity]*.

`Full path`	OpenSplice/DurabilityService/Network/ Heartbeat[@transport_priority]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - maxInt
`Required`	false

4.3.2.3.3 Element *ExpiryTime*

This element specifies the maximum amount of time in which the Durability Service expects a new heartbeat of other Durability Services. This is obviously also the same amount of time in which the Durability Service must send a heartbeat itself.

Increasing this value will lead to less network traffic and overhead but also to less responsiveness with respect to the liveliness of a Durability Service. Change this value according to the needS of your system with respect to these aspects.

`Full path`	OpenSplice/DurabilityService/Network/ Heartbeat/ExpiryTime
`Format`	float
`Dimension`	seconds
`Default value`	10.0
`Valid values`	0.2 - maxFloat
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute update_factor*
`Optional attributes`	<none>

4.3.2.3.3.1 Attribute update_factor

In case of a (temporary) high CPU load, the scheduling behaviour of the operating system might affect the capability of the Durability Service to send its heartbeat 'on time'. This attribute introduces some elasticity in this mechanism by making the service send its heartbeat more often than required by the *ExpiryTime*.

The Durability Service will report its liveliness every *ExpiryTime* multiplied by this *update_factor*.

`Full path`	OpenSplice/DurabilityService/Network/ Heartbeat/ExpiryTime[@update_factor]
`Format`	float
`Dimension`	n/a
`Default value`	0.2
`Valid values`	0.1 - 0.9
`Required`	true

4.3.2.3.4 Element *Scheduling*

This element specifies the scheduling parameters used by the thread that periodically sends the heartbeats.

`Full path`	OpenSplice/DurabilityService/Network/Heartbeat/Scheduling
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.3.4.1 Element *Class*

This element specifies the thread scheduling class that will be used by the thread that periodically sends the heartbeats. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/DurabilityService/Network/Heartbeat/Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.3.4.2 Element *Priority*

This element specifies the thread priority that will be used by the thread that periodically sends the heartbeats. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/DurabilityService/Network/Heartbeat/Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.3.2.3.4.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/DurabilityService/Network/Heartbeat/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.3.2.4 Element *InitialDiscoveryPeriod*

On startup the Durability Service needs to determine, for each namespace, if it has to align with other Durability Services in the system or if it has to load the initial state from disk (load persistent data). For this the Durability Service will publish a request for information and wait for the specified *initial discovery period* for all Durability services to respond. The Durability Service will load the persistent data from disk if no response is received within the specified initial discovery period.

This initial discovery period should be configured greater than the worst case expected discovery time which is related to underlying hardware, type of network, network configuration, and expected load. If the initial discovery period is too short the Durability Service may conclude that there is no running system and load the data from disk, which will result in conflicting states (‘split-brain syndrome’) *i.e.* two separate systems[1].

The Durabiltiy Service will wait for at least the full initial discovery period before it can continue and become operational, so for fast startup times it is important to keep the initial discovery period as small as possible.

`Full path`	OpenSplice/DurabilityService/Network/ InitialDiscoveryPeriod
`Format`	float
`Dimension`	seconds
`Default value`	3.0
`Valid values`	0.1 - 10.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.5 Element *Alignment*

The Durability Service is responsible for keeping its local cache consistent with the other available Durability caches in the system. To do this, it needs to exchange data to recover from inconsistencies. The exchange of durable data to restore consistency is called alignment. This element allows fine-tuning alignment behaviour of the Durability Service.

`Full path`	OpenSplice/DurabilityService/Network/Alignment
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element TimeAlignment* *Element AlignerScheduling* *Element AligneeScheduling* *Element RequestCombinePeriod* *Element TimeToWaitForAligner*
`Required attributes`	<none>
`Optional attributes`	*Attribute latency_budget* *Attribute transport_priority*

1. The metaphoric term ‘split-brain syndrome’ is sometimes used to highlight the results of a temporary outage of communications between two parts of a system. In such a situation, the states of the disconnected parts evolve separately and become incompatible, so that by the time communication is restored the system has become ‘schizophrenic’.

4.3.2.5.1 Attribute latency_budget

This attribute specifies the latency budget QoS setting (in seconds) that is only used by the Durability Service for the alignment of data. It overrules the value of the *DurabilityService/Network[@latency_budget]*.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment[@latency_budget]
`Format`	float
`Dimension`	seconds
`Default value`	0.0
`Valid values`	0.0 - maxFloat
`Required`	false

4.3.2.5.2 Attribute transport_priority

This attribute specifies the transport priority QoS setting that is only used by the Durability Service for the alignment of data. It overrules the value of the *DurabilityService/Network[@transport_priority]*.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment[@transport_priority]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - maxInt
`Required`	false

4.3.2.5.3 Element *TimeAlignment*

This attribute specifies whether time on all Domain Services in the domain can be considered aligned or not. This setting needs to be consistent for all durability services in the domain. In case there is no time alignment, the durability service needs to align more data to compensate possible timing gaps between different Domain Services in the domain.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment/TimeAlignment
`Format`	boolean
`Dimension`	n/a

Default value	true
Valid values	true, false
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.2.5.4 Element *AlignerScheduling*

This element specifies the scheduling parameters used to control the thread that aligns other durability services.

Full path	OpenSplice/DurabilityService/Network/ Alignment/AlignerScheduling
Occurrences (min-max)	0 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.3.2.5.4.1 Element *Class*

This element specifies the thread scheduling class that will be used by the aligner thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/DurabilityService/Network/ Alignment/AlignerScheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.2.5.4.2 Element *Priority*

This element specifies the thread priority that will be used by the aligner thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment/AlignerScheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.3.2.5.4.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/DurabilityService/Network/Alignment/ AlignerScheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.3.2.5.5 Element *AligneeScheduling*

This element specifies the scheduling parameters used to control the thread that makes sure the local service becomes and stays aligned.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment/AligneeScheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.5.5.1 Element *Class*

This element specifies the thread scheduling class that will be used by the alignee thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment/AligneeScheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.5.5.2 Element *Priority*

This element specifies the thread priority that will be used by the alignee thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/DurabilityService/Network/ Alignment/AligneeScheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system

Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.3.2.5.5.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/DurabilityService/Network/Alignment/AligneeScheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.3.2.5.6 Element *RequestCombinePeriod*

When the Durability Service detects an inconsistency with another Durability Service, it requests that service to align it. The service that receives this request will restore consistency by sending the requested information. In some cases, the Durability Service may receive alignment requests from multiple Durability Services for the same information around the same moment in time. To reduce the processing and network load in that case, the Durability Service is capable of aligning multiple Durability Services concurrently.

The RequestCombinePeriod has 2 child-elements: a setting that is used when the current Durability Service is not yet aligned with all others (*Initial*) and one for the period after that (*Operational*). These values specify the maximum amount of time the Durability service is allowed to wait with alignment after an alignment request has been received.

Increasing the value will increase the amount of time in which the Durability Service restores from inconsistencies, but will decrease the processing and network load in case multiple Durability Services need to resolve the same data around the same time. Increasing the value is useful in case more than two Domain Services for the same Domain are started at the same time.

Full path	OpenSplice/DurabilityService/Network/ Alignment/RequestCombinePeriod
Occurrences (min-max)	0 - 1
Child-elements	*Element Initial* *Element Operational*
Required attributes	<none>
Optional attributes	<none>

4.3.2.5.6.1 Element *Initial*

This element specifies the maximum amount of time the Durability Service is allowed to wait with alignment after an alignment request has been received and the service itself is not yet considered operational because it has not yet aligned itself with all other Durability Services.

Full path	OpenSplice/DurabilityService/Network/Alignment/ RequestCombinePeriod/Initial
Format	float
Dimension	seconds
Default value	0.5
Valid values	0.01 - 5.0
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.2.5.6.2 Element *Operational*

This element specifies the maximum amount of time the Durability Service is allowed to wait with alignment after an alignment request has been received and the service itself is already considered operational.

Full path	OpenSplice/DurabilityService/Network/Alignment/ RequestCombinePeriod/Operational
Format	float
Dimension	seconds
Default value	0.01

`Valid values`	0.01 - 5.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.5.7 Element TimeToWaitForAligner

When all Durability Services in the domain have configured their `aligner` element as `false` (see section *4.3.4.2.6* on page 182 for a full description of OpenSplice/DurabilityService/NameSpaces/Policy[@aligner]), none of them is able to act as an aligner for newly-started Durability Services. Therefore late-joining Durability Services will not be able to obtain historical data that is available in the domain.

This element specifies the period (in seconds) to wait until an aligner becomes available in the domain. If an aligner does not become available within the period specified by this element, the entire federation will terminate and return with error code `1` (recoverable error).

Currently only values between `0.0` and `1.0` are supported, and all non-zero values are interpreted as infinite (so basically the time-out is currently either zero or infinite). The default is `1.0`. Note that when the element `aligner` (see section *4.3.4.2.6*) is set to `true` the current Durability Service is able to act as aligner for other Durability Services with respect to the specified namespace and the federation will not terminate.

`Full path`	OpenSplice/DurabilityService/Network/Alignment/TimeToWaitForAligner
`Format`	float
`Dimension`	seconds
`Default value`	1.0
`Valid values`	0.0 - 1.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.2.6 Element *WaitForAttachment*

The Durability Service depends on the Network Service for its communication with other Durability Services. Before it starts communicating, it must make sure the Network Service is ready to send the data. This element specifies what services must be available and how long the Durability Service must wait for them to become available before sending any data.

Full path	OpenSplice/DurabilityService/Network/WaitForAttachment
Occurrences (min-max)	0 - 1
Child-elements	*Element ServiceName*
Required attributes	0
Optional attributes	*Attribute maxWaitCount*

4.3.2.6.1 Attribute maxWaitCount

This attribute specifies the number of times the Durability Service checks if the services specified in the *DurabilityService/Network/WaitForAttachment/ServiceName* elements are available before sending any data. The time between two checks is 100ms. An error is logged if one of the services still is unavailable afterwards. The service will continue after that, but this indicates a problem in the configuration and the service might not function correctly anymore.

Full path	OpenSplice/DurabilityService/Network/WaitForAttachment[@maxWaitCount]
Format	unsigned integer
Dimension	100ms per wait
Default value	200
Valid values	1 - 1000
Required	false

4.3.2.6.2 Element *ServiceName*

This element specifies the name of the service(s) that the Durability Service waits for, before starting alignment activities for a specific topic-partition combination. If (for example) the communication between Durability Services is dependent on the availability of certain local Network Services, then the Durability Service must wait until these are operational.

`Full path`	OpenSplice/DurabilityService/Network/ WaitForAttachment/ServiceName
`Format`	string
`Dimension`	name of an existing service
`Default value`	networking
`Valid values`	any valid service name
`Occurrences (min-max)`	1 - *
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3 Element *Persistent*

The Durability Service is responsible for persisting data from Persistent Topics to disk. This configuration element controls where the data is stored and how the Durability Service will perform this task.

⚠ Note that these elements are only available as part of the DDS persistence profile of OpenSplice.

`Full path`	OpenSplice/DurabilityService/Persistent
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element StoreDirectory* *Element StoreMode* *Element StoreSessionTime* *Element StoreSleepTime* *Element StoreOptimizeInterval* *Element Scheduling* *Element MemoryMappedFileStore* *Element SmpCount*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.1 Element *StoreDirectory*

This element determines the location where the persistent data will be stored on disk. The value should point to a directory on disk. If this parameter is not configured, the Durability Service will not manage persistent data.

Full path	OpenSplice/DurabilityService/Persistent/StoreDirectory
Format	string
Dimension	path to directory
Default value	/tmp/pstore
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.3.2 Element *StoreMode*

This element specifies the plugin that is used to store the persistent data on disk.

- With "XML" mode, the service will store persistent data in XML files.
- With “MMF” mode, the service will store persistent data in a Memory Mapped File that exactly represents the memory that is being used by the persistent store.
- With “KV” mode the service will store persistent data in a key-value store.

The key-value store makes use of a third-party product to store the persistent data; it currently supports SQLite and LevelDB store implementations.

CAUTION: The “MMF” store is currently only implemented on linux. For “KV” stores, SQLite is supported on linux, Windows, and Solaris; LevelDB is only supported on linux.

Full path	OpenSplice/DurabilityService/Persistent/StoreMode
Format	enumeration
Dimension	n/a
Default value	XML
Valid values	XML, MMF, KV
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.3.3 Element *StoreSessionTime*

The Durability Service has a persistency thread that periodically (in sessions) writes persistent data to disk, this element together with the *Element StoreSleepTime* can be used to optimize disk access. This element specifies the maximum session time (in seconds) for the persistency thread. After this period of time, it makes sure data is flushed to disk.

`Full path`	OpenSplice/DurabilityService/Persistent/ StoreSessionTime
`Format`	float
`Dimension`	seconds
`Default value`	20.0
`Valid values`	5.0 - 60.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.4 Element *StoreSleepTime*

This element specifies the period of time (in seconds) the persistency thread sleeps between two store sessions, also see *Element StoreSessionTime*. This allows influencing the CPU load of the persistency thread.

`Full path`	OpenSplice/DurabilityService/Persistent/ StoreSleepTime
`Format`	float
`Dimension`	seconds
`Default value`	2.0
`Valid values`	0.5 - 10.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.5 Element *StoreOptimizeInterval*

This element determines after how many write actions the persistent set for a specific partition-topic combination is optimized on disk. Persistent data is sequentially written to disk without removing data that according to key values and history policies can be removed. During a store optimize action the Durability Service will rewrite the file and thereby remove all disposable data. Note that a long interval will minimize the induced mean load but instead increases burst load.

Full path	OpenSplice/DurabilityService/Persistent/ StoreOptimizeInterval
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - 1000000000
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.3.6 Element *Scheduling*

This element specifies the scheduling parameters used to control the thread that stores persistent data on permanent storage.

Full path	OpenSplice/DurabilityService/Persistent/ Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.3.3.6.1 Element *Class*

This element specifies the thread scheduling class that will be used by the persistent thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/DurabilityService/Persistent/ Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.6.2 Element *Priority*

This element specifies the thread priority that will be used by the persistent thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/DurabilityService/Persistent/ Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.3.3.6.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/DurabilityService/Persistent/ Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.3.3.7 Element *MemoryMappedFileStore*

⚠ **CAUTION:** The “MMF” store is currently only implemented on linux.

This element specifies the memory mapped file store mode parameters. This element is only valid when the `Persistent/StoreMode` element (see Section 4.3.3.2, *Element StoreMode*, on page 166) is set to “`MMF`”. The size and the starting address of the memory file must be supplied.

`Full path`	OpenSplice/DurabilityService/Persistent/ MemoryMappedFileStore
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Size* *Element Address*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.7.1 Element *Size*

⚠ **CAUTION:** The “MMF” store is currently only implemented on linux.

This element specifies the size of the memory mapped file used to store persistent data. Change this value according to the size of your persistent data. The file should be big enough to store:

- all persistent data in your specified namespaces
- *plus* a potential backup for all persistent data in namespaces whose content may be replaced by persistent data from another master.

As a rule of thumb, the persistent store should be twice the size of your combined persistent data in your specified namespaces. Please note that the operating system must be configured to support the requested size.

The human-readable option lets the user postfix the value with `K`(ilobyte), `M`(egabyte) or `G`(igabtye). For example, `10M` results in 10485760 bytes.

`Full path`	OpenSplice/DurabilityService/Persistent/MemoryMappedFileStore/Size
`Format`	unsigned integer, human-readable
`Dimension`	bytes
`Default value`	10485760
`Valid values`	min: 1048576
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.7.2 Element *Address*

CAUTION: The “MMF” store is currently only implemented on linux.

This element specifies the start address where the file is mapped into the virtual memory. The possible values are platform dependent. Change this value if the default address is already in use, for example by another Domain Service or another product.

`Full path`	OpenSplice/DurabilityService/Persistent/MemoryMappedFileStore/Address
`Format`	(hexadecimal) memory address
`Dimension`	memory address
`Default value`	0x80000000 (Linux)
`Valid values`	depends on operating system
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.3.8 Element *SmpCount*

This element determines how many threads the Durability service will spawn to write persistent data to disk. Note that this attribute is currently only supported for MMF (memory mapped file) StoreMode.

Please also note that although technically the maximum valid value for this element is `maxInt`, the operating system may impose a lower limit, to prevent 'runaway' consumption of resources and loss of performance. It is recommended that increases of this value are carefully considered!

CAUTION: The "MMF" store is currently only implemented on linux.

Full path	OpenSplice/DurabilityService/Persistent/ SmpCount
Format	unsigned integer
Dimension	n/a
Default value	1
Valid values	1 - maxInt; may be limited by operating system
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.3.9 Element *KeyValueStore*

CAUTION: The "KV" store is currently only supported on linux (SQLite and LevelDB), Windows (SQLite), and Solaris (SQLite).

This element specifies the key-value store mode parameters. This element is only valid when the `Persistent/StoreMode` element is set to "`KV`" (see Section 4.3.3.2, *Element StoreMode*, on page 166).

It specifies which third-party software is used to implement the key-value store.

Full path	OpenSplice/DurabilityService/Persistent/ KeyValueStore
Occurrences (min-max)	0 - 1
Child-elements	*Element ConfigParameters*
Required attributes	*Attribute type*
Optional attributes	<none>

4.3.3.9.1 Attribute type

This attribute specifies the third-party product that is used to implement the KV store.

Products currently supported are SQLite and LevelDB.

`Full path`	OpenSplice/DurabilityService/Persistent/KeyValueStore[@type]
`Format`	enumeration
`Dimension`	n/a
`Default value`	sqlite
`Valid values`	sqlite, leveldb
`Required`	True

4.3.3.9.2 Element *ConfigParameters*

This element is used to set parameters that are specific to the third-party product used to implement the KV store. The element consists of a list of parameters which are separated by semicolons (‘`;`’). Each parameter is either a single name or a key-value pair where the key and the value are separated by a ‘`=`’ character.

Invalid or ‘not recognized’ values are ignored.

- When SQLite is selected as KV store implementation, refer to the Sqlite documentation for full details of the available parameters (see *http://www.sqlite.org/pragma.html*). The only exceptions are the parameters that the KV store uses itself, which are: `locking_mode`, `journal_mode`, `wal_autocheckpoint` and `synchronous`.
- When LevelDB is selected as the KV store the following parameters are available (the information below has been taken from the current LevelDB documentation—the project home page is at *http://code.google.com/p/leveldb/*):

`paranoid_checks` — boolean

If *`true`*, the implementation will do aggressive checking of the data it is processing and it will stop early if it detects *any* errors. This may have unforeseen ramifications: for example, a corruption of one database entry may cause a large number of entries to become unreadable or for the entire database to become unopenable.
Default: *`false`*

`write_buffer_size` — integer

Amount of data to build up in memory (backed by an unsorted log on disk) before converting to a sorted on-disk file. Larger values improve performance, especially during bulk loads. Up to two write buffers may be held in memory at the same time, so you may wish to adjust this parameter to control memory usage. Also, a larger write buffer will result in a longer recovery time the next time the database is opened.
Default: *`4MB`*

max_open_files — integer
Number of open files that can be used by the database. You may need to increase this if your database has a large working set (budget one open file per 2MB of working set).
Default: *1000*

block_size — integer
Approximate size of user data packed per block. Note that the block size specified here corresponds to uncompressed data. The actual size of the unit read from disk may be smaller if compression is enabled. This parameter can be changed dynamically.
Set by the KV store to *1M*.

verify_checksums — boolean
If *true*, all data read from underlying storage will be verified against corresponding checksums.
Default: *false*

fill_cache — boolean
Should the data read for this iteration be cached in memory? Callers may wish to set this field to *false* for bulk scans.
Default: *true*

Full path	OpenSplice/DurabilityService/Persistent/KeyValueStore/ConfigParameters
Format	string containing a list of parameters separated by ‘;’ characters. Each parameter is *either* a single name *or* a key-value pair where the key and the value are separated by a ‘=’ character.
Dimension	n/a
Default value	n/a
Valid values	depends on selected store implementation
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.4 Element *NameSpaces*

When a durability service wants to fulfill a particular role for some of the namespaces in a domain, it must have some way of deducing the desired behavior for those when encountered. For static, small-scale systems this can easily be solved by statically configurating this role-behavior for all relevant namespaces for each durability service in the domain.

In dynamic, large scale environments, the updating and maintaining of configurations for each durability service when new namespaces enter the domain can become quite cumbersome. Dynamic namespaces offer a solution for this problem.

Instead of specifying each namespace seperately, dynamic namespaces introduce the concept of namespace policies. A policy defines a generic role for the durability service, together with a namespace expression. This expression can contain wildcards, and is used to match against each namespace the durability service encounters in a domain. The first policy with a matching expression is then applied to the new namespace.

Specifying policies

Policies are specified in a fall-through manner, which means that the first (top) policy to match a namespace is applied. Policies specify a range of options for namespaces, which tell the durability service how to handle the data. The following items can be configured:

- Durability
- Alignee
- Aligner

In the dynamic namespace configuration, the `NameSpace` element (a child of the NameSpaces element) only supports a `name` attribute, which is mandatory. This name will be used to match against policies.

An example of dynamic namespace configuration is shown on page 182, followed by a short note about *Backwards compatibility*.

`Full path`	OpenSplice/DurabilityService/NameSpaces
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element NameSpace* *Element Policy*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.4.1 Element *NameSpace*

A NameSpace describes a dependency between data in two or more partitions by means of a partition expression. The dependency specifies that the data within one of the partitions has no right to exist separately from the data in the other partition(s). Namespaces determine which data must be managed by the Durability service. Data that does not match any of the namespaces, is ignored by the Durability service.

`Full path`	OpenSplice/DurabilityService/NameSpaces/ NameSpace
`Occurrences (min-max)`	1 - *
`Child-elements`	*Element Partition* *Element PartitionTopic*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.3.4.1.1 Attribute name

This element specifies the name for a namespace. A name is used to match a namespace with a policy.

`Full path`	OpenSplice/DurabilityService/NameSpaces/ NameSpace[@name]
`Format`	string
`Dimension`	n/a
`Default value`	“defaultNameSpace”
`Valid values`	Any valid string
`Required`	false

4.3.4.1.2 Element *Partition*

This element specifies a partition expression that matches the namespace. A namespace consists of a set of partition expressions. Together they determine the partitions that belong to the namespace. Make sure that the different name-spaces do not have an overlap in partitions. The default configuration has one namespace containing all partitions. A partition may contain the wildcards ‘*’ to match any number of characters and ‘?’ to match a single character.

`Full path`	OpenSplice/DurabilityService/NameSpaces/ NameSpace/Partition
`Format`	string
`Dimension`	partition name
`Default value`	“*”
`Valid values`	any valid partition name
`Occurrences (min-max)`	1 - *
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.4.1.3 Element *PartitionTopic*

This element specifies a partition-topic expression that matches the namespace. A group expression is a combination of a partition and a topic expression. The notation is ‘*`partition.topic`*’. A namespace consists of a set of partition-topic expressions. Together they determine the partition-topic combinations that belong to the namespace. Make sure that the different namespaces do not have an overlap in expressions. The default configuration has one namespace containing all combinations (`*.*`). A partition-topic expression may contain the wildcards ‘`*`’ to match any number of characters and ‘`?`’ to match a single character.

`Full path`	OpenSplice/DurabilityService/NameSpaces/ NameSpace/PartitionTopic
`Format`	string
`Dimension`	partition name.topic name
`Default value`	“*.*”
`Valid values`	any valid partition-topic combination
`Occurrences (min-max)`	0 - *
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.4.2 Element *Policy*

Policies are child elements of the NameSpaces element. The Policy element has a mandatory nameSpace attribute, in which a namespace expression is expected. This expression will be used to match with a namespace name.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy
Occurrences (min-max)	1 - *
Child-elements	*Element Merge*
Required attributes	*Attribute nameSpace* *Attribute durability* *Attribute alignee* *Attribute aligner*
Optional attributes	*Attribute delayedAlignment*

4.3.4.2.1 Attribute nameSpace

The element specifies an expression that matches a namespace name. A namespace may contain the wildcards ‘*****’ to match any number of characters and ‘**?**’ to match one single character.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy[@nameSpace]
Format	string
Dimension	n/a
Default value	*
Valid values	any valid string
Required	true

4.3.4.2.2 Attribute delayedAlignment

This element determines whether the durability allows delayed alignment of initial data. This can be useful for systems where there can be late-joining Durability Services with a persistent dataset, which by default are then not inserted. When this option is enabled, durability will only insert a persistent set from a late-joining service when no writers have been created in the partitions matched by the namespace.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy[@delayedAlignment]
Format	boolean
Dimension	n/a

`Default value`	false
`Valid values`	true, false
`Required`	false

4.3.4.2.3 Element *Merge*

This element specifies which action the Durability Service should take when a mismatch in namespace state is discovered. Mismatches in namespace state typically occur in the case of connectivity issues between two or more durability services. When namespace states do not match, the middleware is unable to determine which set is ‘right’. Therefore it provides the user with a number of configuration parameters which determine how to handle a state mismatch.

`Full path`	OpenSplice/DurabilityService/NameSpaces/Policy/ Merge
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute type* *Attribute scope*
`Optional attributes`	<none>

4.3.4.2.3.1 Attribute type

The `type` attribute describes the kind of action required on a namespace state mismatch.

- **Ignore**: Do nothing in case of a state mismatch. No samples are aligned, and namespace states will not be updated.
- **Merge**: Merge historical data from other namespace state. This will result in a new namespace state for the durability service that specifies this value.
- **Delete**: Dispose and delete historical data in case of a state mismatch. Immediately after successful completion of the Delete merge action no transient or persistent data will be available for late-joining readers, and all data in the reader queue of existing readers will be disposed.
- **Replace**: Dispose and delete historical data in case of a state mismatch, and merge data from another namespace state. This will result in a new namespace state for the durability service that specifies this value. Immediately after successful completion of the Replace merge action the replacement data will be available to late-joining readers, the data in the reader queue of existing readers will be disposed and replaced with the replacement data, and the generation count of the replacement data is increased.

`Full path`	OpenSplice/DurabilityService/NameSpaces/Policy/Merge [@type]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Ignore
`Valid values`	Ignore, Merge, Delete, Replace
`Required`	true

4.3.4.2.3.2 Attribute scope

The `scope` attribute describes for which scope the merge policy is valid. The scope is a role-expression in which wildcards (‘`*`’ and ‘`?`’) are allowed. Roles are matched at runtime against this expression to determine which policy applies for that role. When a role doesn’t match any policy, ‘`Ignore`’ is assumed. The order of specifying policies is important: the first scope expression that matches a role is selected for that role.

`Full path`	OpenSplice/DurabilityService/NameSpaces/Policy/Merge [@scope]
`Format`	String
`Dimension`	n/a
`Default value`	DefaultRole
`Valid values`	any valid DDS role expression
`Required`	true

4.3.4.2.4 Attribute durability

This element specifies how the durability service manages the data within the NameSpace. The original durability of the data (determined by the DataWriter that wrote it) can be ‘weakened’ (Persistent > Transient > Transient_local). This is useful to improve resource usage of the durability service in the situation where resource usage is more important then fault-tolerance. This parameter cannot be used to increase the original durability of samples. In case the value of this parameter is larger then the value a sample was published with, the sample will be handled as specified in the DataWriter durability QoS.

- **Persistent**: Maximum profile. A sample will be handled as specified in the durability Qos of the datawriter that wrote it.
- **Transient**: A sample will be stored transient at best.
- **Transient_Local**: A sample will be stored transient_local at best.

- **Durable**: Convenience value that is equal to persistent.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy [@durability]
Format	enumeration
Dimension	n/a
Default value	Durable
Valid values	Persistent, Transient, Transient_Local, Durable
Required	true

4.3.4.2.5 Attribute alignee

This element determines how the durability service manages the data that matches the namespace. Scalability of durable data is an issue in large systems. Keeping all historical data within each Durability Service may not be feasible. Often Durability Services are interested in a small part of the total system data. They are driven by both performance (boot time, memory usage, network load, CPU load) and fault tolerance (the need for replicates).

- **Initial**: The durability service requests historical data at startup and caches it locally. Historical data will be available relatively fast for new local data readers and the system is more fault-tolerant. However, caching of historical data requires a relatively large amount of resources and a long boot time.
- **Lazy**: The Durability service caches historical data after local application interest arises for the first time and a remote Durability service aligns the first data reader. Historical data is available relatively slow for the first data reader, but for every new data reader it is relatively fast. The caching resources are only used when local interest in the data arises, so it only requires resources if there is actual local interest. However, this method provides no fault-tolerance for the domain, because the local Durability service is only partly a replica and is not able to provide historical data to remote Durability service and/or remote data readers.
- **On_Request**: The Durability service will not cache historical data, but will align each separate DataReader on a request basis (in the situation where it calls wait_for_historical_data). Each new DataReader that wants historical data therefore leads to a new alignment action. This is a good setting to limit the amount of resources used, but will potentially lead to more network traffic. This method provides no fault-tolerance for the domain.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy[@alignee]
Format	enumeration
Dimension	n/a
Default value	Initial
Valid values	Initial, Lazy, On_Request
Required	true

4.3.4.2.6 Attribute aligner

This mandatory attribute determines whether the durability service can act as aligner (provide historical data) for other durability services.

Full path	OpenSplice/DurabilityService/NameSpaces/Policy[@aligner]
Format	boolean
Default value	true
Valid values	true, false
Required	true

Example

Consider the following configuration example for the durability service:

```
<NameSpaces>
  <NameSpace name="Ns01">
    <Partition>A*</Partition>
  </NameSpace>
  <Policy nameSpace="Ns*" durability="Durable" alignee="Initial"
aligner="True"/>
  <Policy nameSpace="*" durability="Transient" alignee="Lazy"
aligner="False"/>
</NameSpaces>
```

From this configuration, the following behavior can be deduced:

- One namespace with name "Ns01" which holds all partitions with prefix "A" is created and assigned with the following policy: {durabilityKind=Durable, alignmentKind=Initial, aligner=True}
- All late-joining namespaces with prefix "Ns" will be assigned with the following policy: {durabilityKind=Durable, alignmentKind=Initial, aligner=True}
- All other late joining namespaces (without prefix "Ns") will be assigned with the following policy: {durabilityKind=Transient, alignmentKind=Lazy, aligner=False}

Backwards compatibility

Although the structure of the namespace configuration file has changed, the durability service still supports the old format, where the policies for a namespace are configured on the namespace element itself. An example of an old configuration:

```
<NameSpaces>
  <NameSpace durabilityKind="Durable"
alignmentKind="Initial_and_Aligner">
    <Partition>*</Partition>
  </NameSpace>
</NameSpaces>
```

Note that this configuration is deprecated and supported only by the durability service for backwards compatibility. New users should use the new configuration and existing users should migrate to this configuration whenever possible. The configurator will by default not accept the old format anymore. When for some reason migration to the new format is not possible or desired, a user can use the configurator from an older version, or edit the configuration file manually.

When converting from old to new format, all namespace elements must be provided with a name, and a matching policy for that name must be defined. For the above example of an old configuration, this could be the equivalent in the new format:

```
<NameSpaces>
  <NameSpace name="DefaultNameSpace">
    <Partition>*</Partition>
  </NameSpace>
  <Policy nameSpace="DefaultNameSpace" durability="Durable"
aligner="True" alignee="Initial"/>
</NameSpaces>
```

4.3.5 Element *Watchdog*

This element controls the characteristics of the Watchdog thread.

`Full path`	OpenSplice/DurabilityService/Watchdog
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.5.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

Full path	OpenSplice/DurabilityService/Watchdog/Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.3.5.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/DurabilityService/Watchdog/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.5.1.2 Element *Priority*

This element specifies the thread priority that will be used by the watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/DurabilityService/Watchdog/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system

`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.3.5.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/DurabilityService/Persistent/ Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.3.6 Element *EntityNames*

This element specifies the names of the various entities used by the DurabilityService. The names specified here will be displayed in the OpenSplice DDS Tuner when viewing the DurabilityService.

`Full path`	OpenSplice/DurabilityService/EntityNames
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Publisher* *Element Subscriber* *Element Partition*
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.6.1 Element *Publisher*

This element specifies the name of the durability publisher.

Full path	OpenSplice/DurabilityService/EntityNames/Publisher
Format	string
Dimension	entity name
Default value	“durabilityPublisher”
Valid values	any string
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.6.2 Element *Subscriber*

This element specifies the name of the durability subscriber.

Full path	OpenSplice/DurabilityService/EntityNames/Subscriber
Format	string
Dimension	entity name
Default value	“durabilitySubscriber”
Valid values	any string
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.6.3 Element *Partition*

This element specifies the name of the durability partition.

Full path	OpenSplice/DurabilityService/EntityNames/Partition
Format	string
Dimension	partition name
Default value	“durabilityPartition”
Valid values	any valid partition name

`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.3.7 Element Tracing

This element controls the amount and type of information that is written into the tracing log by the Durability Service. This is useful to track the Durability Service during application development. In the runtime system it should be turned off.

`Full path`	OpenSplice/DurabilityService/Tracing
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element OutputFile* *Element Timestamps* *Element Verbosity*
`Required attributes`	<none>
`Optional attributes`	*Attribute synchronous*

4.3.7.1 Attribute synchronous

This attribute specifies whether tracing log updates are synchronous or not. A synchronous update is immediately flushed to disk: there is no buffering and therefore some performance overhead. Only use this option if you are debugging and you want to make sure all Tracing info is on disk if (when) the service crashes.

`Full path`	OpenSplice/DurabilityService/ Tracing[@synchronous]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	false

4.3.7.2 Element *OutputFile*

This option specifies where the logging is printed to. Note that "stdout" is considered a legal value that represents "standard out" and "stderr" is a legal value representing "standard error".

The default value is an empty string, indicating that all tracing is disabled.

Full path	OpenSplice/DurabilityService/Tracing/OutputFile
Format	string
Dimension	file name
Default value	“” (empty string)
Valid values	depends on operating system.
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.3.7.3 Element *Timestamps*

This element specifies whether the logging must contain timestamps.

Full path	OpenSplice/DurabilityService/Tracing/Timestamps
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute Absolute*

4.3.7.3.1 Attribute Absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

Full path	OpenSplice/DurabilityService/Tracing/ Timestamps[@absolute]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.3.7.4 Element *Verbosity*

This element specifies the verbosity level of the loggin information. The higher the level, the more (detailed) information will be logged. The value '`FINEST`' produces the most detailed log.

`Full path`	OpenSplice/DurabilityService/Tracing/Verbosity
`Format`	enumeration
`Dimension`	n/a
`Default value`	INFO
`Valid values`	SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4 The Network and the Secure Network Service

4.4.1 The Network Service

The Network Service provides a bridge between the local DDS service and a network interface.

i The OpenSplice `NetworkService` supports both Internet Protocol Versions 4 and 6 (IPv4 & IPv6) where possible. Please refer to the Release Notes (*Known Issues* section) to see if the IPv6 capability is present on your operating system.

⚠ Note that each service instance will only communicate using *one* of these protocols. It is an error to specify IPv6 ('colon-separated hexadecimal') and IPv4 ('dotted decimal') addresses in the same `NetworkService` configuration.

Multiple Network Services can exist next to each other, each serving one physical network interface.

Please refer to section Section 1.4, *Applications which operate in multiple domains*, on page 17 for notes about applications operating in multiple domains and interactions with the Network Service.

The Network Service is responsible for forwarding data to the network and for receiving data from the network. It can be configured to distinguish multiple communication channels with different QoS policies assigned to be able to schedule sending and receival of specific messages to provide optimal performance for a specific application domain.

The Network Service is selected by using the following configuration element to the Domain section of the configuration file (see Section 4.2.10, *Element Application*, on page 111).

```
<Service name="networking">
     <Command>networking</Command>
</Service>
```

The network configuration expects a root element named *`OpenSplice/NetworkService`*. Within this root element, the Network Service will look for several child-elements. Each of these is listed and explained.

`Full path`	OpenSplice/NetworkService
`Occurrences (min-max)`	0 - *
`Child-elements`	*Element Watchdog* *Element General* *Element Partitioning* *Element Channels* *Element Discovery* *Element Tracing* *Element Compression*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.4.1.1 Attribute *name*

This attribute identifies the configuration for the Network Service. Multiple Network Service configurations can be specified in one single resource. The actual applicable configuration is determined by the value of the *name* attribute, which must match the one specified under the attribute *OpenSplice/Domain/Service[@name]* in the configuration of the Domain Service.

`Full path`	OpenSplice/NetworkService[@name]
`Format`	string
`Dimension`	n/a

Default value	networking
Valid values	any string
Required	true

4.4.1.2 Element *Watchdog*

This element controls the characteristics of the Watchdog thread.

Full path	OpenSplice/NetworkService/Watchdog
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.4.1.2.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

Full path	OpenSplice/NetworkService/Watchdog/ Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.4.1.2.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/NetworkService/Watchdog/ Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1

Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.2.1.2 Element *Priority*

This element specifies the thread priority that will be used by the watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/NetworkService/Watchdog/ Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.4.1.2.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/NetworkService/Watchdog/ Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Occurrences (min-max)	0 - 1
Required	false

4.4.1.3 Element *General*

This element contains general parameters that concern the Network Service as a whole.

`Full path`	OpenSplice/NetworkService/General
`Occurrences (min-max)`	0 - *
`Child-elements`	*Element NetworkInterfaceAddress* *Element Reconnection* *Element EnableMulticastLoopback*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.3.1 Element *NetworkInterfaceAddress*

This element specifies which network interface card (NIC) should be used.

`Full path`	OpenSplice/NetworkService/General/ NetworkInterfaceAddress
`Format`	string
`Dimension`	n/a
`Default value`	"first available"
`Valid values`	"first available", any ‘dotted decimal’ IPv4 or ‘colon-separated hexadecimal’ IPv6 address, or a symbolic name of a NIC
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute forced* *Attribute ipv6*
`Remarks`	The given interface should have the required capabilities, *e.g.* broadcasting

Every Network Service is bound to only one network interface card. The card can be uniquely identified by its corresponding IP address or by its symbolic name (*e.g.* `eth0`). If the value “`first available`” is entered here, OpenSplice will try to look up an interface that has the required capabilities. If the value for this element is an Internet Protocol Version 6 (IPv6) address, then this `NetworkService` will use IPv6 for communication.

NOTE: On Integrity, IP addresses *must* be given in the `NetworkInterfaceAddress` element for nodes which have more than one ethernet interface, as it is not possible to detect NIC broadcast/multicast capabilities automatically in this environment.

4.4.1.3.1.1 Attribute forced

This attribute determines whether only the selected `NetworkInterfaceAddress` will be used or whether others can be used too.

false	Specifies that the `NetworkInterfaceAddress` will be used first, but when it is not available any other available one can be used (default).
true	Specifies that only the selected `NetworkInterfaceAddress` can be used.

`Full path`	OpenSplice/NetworkService/General/ NetworkInterfaceAddress[@forced]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Required`	false

4.4.1.3.1.2 Attribute ipv6

This attribute indicates that the Network Service should use Internet Protocol Version 6 (IPv6) for communication. This attribute is useful if the `NetworkInterfaceAddress` and/or `GlobalPartition[@Address]` are both specified as values that do not canonically indicate the interface's protocol version; *i.e.* if an interface symbolic name or '`first available`' is used for the former, and host names are used for the latter.

`Full path`	OpenSplice/NetworkService/General/ NetworkInterfaceAddress[@ipv6]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Required`	false

4.4.1.3.2 Element *Reconnection*

This element specifies the desired network-behavior with respect to the validity of restoring lost connectivity with remote Network Services. Here ‘lost connectivity’ means a prolonged inability to communicate with a known and still active remote service (typically because of network issues) that has resulted in such a service being declared ‘dead’ either by the *Topology Discovery* or lost-reliability being detected by a reliable channel’s reactivity-checking mechanism. If automatic reconnection is allowed, communication channels with the now-reachable-again service will be restored, even though reliable data might have been lost during the disconnection period.

Full path	OpenSplice/NetworkService/General/Reconnection
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	*Attribute allowed*
Optional attributes	<none>

4.4.1.3.2.1 Attribute allowed

This attribute specifies whether the network service must resume communication with another network service when it has already been seen before but has been disconnected for a while.

- `false` - Specifies that the network service must NOT resume communication.
- `true` - Specifies that the network service must resume communication.

Full path	OpenSplice/NetworkService/General/Reconnection[@allowed]
Format	boolean
Dimension	n/a
Default value	false
Valid values	true, false
Required	true

4.4.1.3.3 Element *EnableMulticastLoopback*

This element specifies whether the Network Service will allow IP multicast packets within the node to be visible to all network participants in the node, including itself. It must be `TRUE` for intra-node multicast communications, but if a node runs only a single OpenSplice Network Service and does not host any other network-capable programs, it may be set to `FALSE` for improved performance.

Full path	OpenSplice/NetworkService/General/ EnableMulticastLoopback
Format	boolean
Dimension	n/a
Default value	false
Valid values	false, true
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.4 Element *Partitioning*

The OpenSplice Network Service is capable of leveraging the network's multicast and routing capabilities. If some *a priori* knowledge about the participating nodes and their topic and partition interest is available, then the Network Services in the system can be explicitly instructed to use specific unicast or multicast addresses for its network traffic. This is done by means of so-called network partitions.

A network partition is defined by one or more unicast, multicast or broadcast IP addresses. Any Network Service that is started will read the network partition settings and, if applicable, join the required multicast groups. For every sample distributed by the Network Service, both its partition and topic name will be inspected. In combination with a set of network partition mapping rules, the service will determine which network partition the sample is written to. The mapping rules are configurable as well.

Using network configuration, nodes can be disconnected from any network partition. If a node is connected via a low speed interface, it is not capable of receiving high volume data. If the DCPS partitioning is designed carefully, high volume data is published into a specific partition, which in its turn is mapped onto a specific network partition, which is itself only connected to those nodes that are capable of handling high volume data.

Full path	OpenSplice/NetworkService/Partitioning
Occurrences (min-max)	0 - 1

`Child-elements`	*Element GlobalPartition* *Element NetworkPartitions* *Element IgnoredPartitions* *Element PartitionMappings*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.4.1 Element *GlobalPartition*

This element specifies the global or default network partition. This global network partition transports data that is either meant to be global, like discovery heartbeats, or that is not mapped onto any other network partition.

`Full path`	OpenSplice/NetworkService/Partitioning/ GlobalPartition
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute Address*
`Optional attributes`	*Attribute MulticastTimeToLive*

4.4.1.4.1.1 Attribute Address

The GlobalPartition address is a list of one or more unicast, multicast or broadcast addresses. If more than one address is specified, then the different addresses should separated by commas (`,`) semicolons (`;`) or spaces (` `). Samples for the global partition will be sent to all addresses that are specified in this list of addresses. To specify the default broadcast address, use the expression "`broadcast`". Addresses can be entered as 'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 notation or as the symbolic hostname, in which case OpenSplice will try to resolve the corresponding IP address.

If the value for this attribute is one, or more, 'colon-separated hexadecimal' Internet Protocol Version 6 (IPv6) address(es), then the `NetworkService` will be configured to use IPv6 for communication.

`Full path`	OpenSplice/NetworkService/Partitioning/GlobalPartition [@Address]
`Format`	'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 address, symbolic host name or "broadcast"
`Dimension`	n/a
`Default value`	"broadcast"

`Valid values`	"broadcast", any 'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 unicast or multicast address, or a resolvable symbolic hostname
`Required`	true
`Remarks`	The given interface should have the required capabilities, *e.g.* broadcasting or multicasting

4.4.1.4.1.2 Attribute MulticastTimeToLive

For each UDP packet sent out, the `TimeToLive` header value is set to this value for Multicast packets.

By specifying a value of '`0`', multicast traffic can be confined to the local node, and such 'loopback' performance is typically optimized by the operating system

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /GlobalPartition[@MulticastTimeToLive]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	32
`Valid values`	0 - 255
`Required`	false

4.4.1.4.2 Element *NetworkPartitions*

Network configuration can contain a set of network partitions, which are grouped under the NetworkPartitions element.

`Full path`	OpenSplice/NetworkService/Partitioning/ NetworkPartitions
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element NetworkPartition*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.4.2.1 Element *NetworkPartition*

Every NetworkPartition has a name, an address and a connected flag.

`Full path`	OpenSplice/NetworkService/Partitioning/ NetworkPartitions/NetworkPartition
`Occurrences (min-max)`	0 - *
`Child-elements`	<none>
`Required attributes`	*Attribute Address* *Attribute Connected*
`Optional attributes`	*Attribute Name* *Attribute Compression* *Attribute MulticastTimeToLive*

4.4.1.4.2.1.1 Attribute Address

The address is a list of one or more unicast, multicast or broadcast addresses. If more than one address is specified, then the different addresses should separated by commas (`,`) semicolons (`;`) or spaces ( ). Samples for this partition will be sent to all addresses that are specified in this list of addresses. To specify the default broadcast address, use the expression "`broadcast`". Addresses can be entered as 'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 notation or as the symbolic hostname, in which case OpenSplice will try to resolve the corresponding IP address.

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /NetworkPartition[@Address]
`Format`	'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 address, symbolic host name or "broadcast"
`Dimension`	n/a
`Default value`	"broadcast"
`Valid values`	"broadcast", any 'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 unicast or multicast address or resolvable symbolic hostname
`Required`	true
`Remarks`	The given interface should have the required capabilities, *e.g.* broadcasting or multicasting.

4.4.1.4.2.1.2 Attribute Connected

This attribute specifies whether the Network Partition will join its associated multicast group(s). The Network Partition will not join multicast group(s) when this attribute is set to `false` and in that case not receive data from multicast groups, the data will be filtered out either by the network interface or by multicast enabled switches. Note that this attribute has no effect on data sent and has no effect on data received *via* a unicast address.

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /NetworkPartition[@Connected]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	true

4.4.1.4.2.1.3 Attribute Name

The Name attribute identifies a Network Partition; it must be unique to create associations with *Element PartitionMappings*.

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /NetworkPartition[@Name]
`Format`	string
`Dimension`	n/a
`Default value`	string representation of the corresponding address
`Valid values`	any valid partition name
`Required`	false
`Remarks`	The name should be unique over all network partitions.

4.4.1.4.2.1.4 Attribute Compression

If the `Compression` attribute is set on a partition, outgoing data will be compressed before sending. Depending on the nature of the published data, this may improve performance, particularly when on a slow network.

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /NetworkPartition[@Connected]
`Format`	boolean
`Dimension`	n/a

`Default value`	false
`Valid values`	true, false
`Required`	false

4.4.1.4.2.1.5 Attribute MulticastTimeToLive

For each UDP packet sent out, the `TimeToLive` header value is set to this value for Multicast packets.

By specifying a value of ‘`0`’, multicast traffic can be confined to the local node, and such ‘loopback’ performance is typically optimized by the operating system

`Full path`	OpenSplice/NetworkService/Partitioning/NetworkPartitions /NetworkPartition[@MulticastTimeToLive]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	32
`Valid values`	0 - 255
`Required`	false

4.4.1.4.3 Element *IgnoredPartitions*

This element is used to group the set of *IgnoredPartition* elements.

`Full path`	OpenSplice/NetworkService/Partitioning/ IgnoredPartitions
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element IgnoredPartition*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.4.3.1 Element *IgnoredPartition*

This element can be used to create a “Local Partition” that is only available within the scope of the Domain Service on which it is specified, and therefore won't generate network-load. Any DCPS partition-topic combination specified in this element will not be distibuted by the Network Service.

Full path	OpenSplice/NetworkService/Partitioning/ IgnoredPartitions/IgnoredPartition
Occurrences (min-max)	1 - *
Child-elements	<none>
Required attributes	*Attribute DCPSPartitionTopic*
Optional attributes	<none>

4.4.1.4.3.1.1 Attribute DCPSPartitionTopic

The Network Service will match any DCPS messages to the DCPSPartitionTopic expression and determine if it matches. The PartitionExpression and TopicExpression are allowed to contain a '*' wildcard, meaning that anything matches. An exact match is considered better than a wildcard match. If a DCPS messages matches an expression it will not be send to the network.

Full path	OpenSplice/NetworkService/Partitioning/ IgnoredPartitions/IgnoredPartition [@DCPSPartitionTopic]
Format	PartitionExpression.TopicExpression
Dimension	n/a
Default value	*.*
Valid values	Expressions containing * wildcards
Required	true

4.4.1.4.4 Element *PartitionMappings*

This element is used to group a set of PartitionMapping elements.

Full path	OpenSplice/NetworkService/Partitioning/ PartitionMappings
Occurrences (min-max)	0 - 1
Child-elements	*Element PartitionMapping*
Required attributes	<none>
Optional attributes	<none>

4.4.1.4.4.1 Element *PartitionMapping*

This element specifies a mapping between a network partition and a partition-topic combination.

`Full path`	OpenSplice/NetworkService/Partitioning/ PartitionMappings/PartitionMapping
`Occurrences (min-max)`	1 - *
`Child-elements`	<none>
`Required attributes`	*Attribute DCPSPartitionTopic* *Attribute NetworkPartition*
`Optional attributes`	<none>

In order to give network partitions a meaning in the context of DCPS, mappings from DCPS partitions and topics onto network partitions should be defined. Network allows for a set of partition mappings to be defined.

4.4.1.4.4.1.1 Attribute DCPSPartitionTopic

The Network Service will match any DCPS messages to the DCPSPartitionTopic expression and determine if it matches. The PartitionExpression and TopicExpression are allowed to contain a ‘*’ wild card, meaning that anything matches. An exact match is considered better than a wild card match. For every DCPS message, the best matching partition is determined and the data is sent over the corresponding network partition as specified by the matching *NetworkPartition* element.

`Full path`	OpenSplice/NetworkService/Partitioning/PartitionMappings /PartitionMapping[@DCPSPartitionTopic]
`Format`	PartitionExpression.TopicExpression
`Dimension`	n/a
`Default value`	`*.*`
`Valid values`	Expressions containing * wildcards
`Required`	true

4.4.1.4.4.1.2 Attribute NetworkPartition

The NetworkPartition attribute of a partition mapping defines that network partition that data in a specific DCPS partition of a specific DCPS topic should be sent to.

Full path	OpenSplice/NetworkService/Partitioning/PartitionMappings /PartitionMapping[@NetworkPartition]
Format	string
Dimension	n/a
Default value	n/a
Valid values	Any name of a previously defined network partition
Required	true

4.4.1.5 Element *Channels*

This element is used to group a set of Channels.

Full path	OpenSplice/NetworkService/Channels
Occurrences (min-max)	1 - 1
Child-elements	*Element Channel* *Element AllowedPorts*
Required attributes	<none>
Optional attributes	<none>

The set of channels define the behaviour of the ‘network’ concerning aspects as priority, reliability and latency budget. By configuring a set of channels, the Network Service is able to function as a ‘scheduler’ for the network bandwidth. It achieves this by using the application-defined DDS QoS policies of the data to select the most appropriate channel to send the data.

4.4.1.5.1 Element *Channel*

This element specifies all properties of an individual Channel.

Full path	OpenSplice/NetworkService/Channels/Channel
Occurrences (min-max)	1 - 42

Child-elements	*Element PortNr* *Element AllowedPorts* *Element FragmentSize* *Element Resolution* *Element AdminQueueSize* *Element Sending* *Element Receiving*
Required attributes	*Attribute name* *Attribute reliable* *Attribute enabled*
Optional attributes	*Attribute default* *Attribute priority*

The Network Service will make sure messages with a higher priority precede messages with a lower priority and it uses the latency budget to assemble multiple messages into one UDP packet where possible, to optimise the bandwidth usage. Of course, its performance depends heavily on the compatibility of the configured channels with the used DDS QoS policies of the applications.

4.4.1.5.1.1 Attribute name

The name uniquely identifies the channel.

Full path	OpenSplice/NetworkService/Channels/Channel[@name]
Format	string
Dimension	n/a
Default value	n/a
Valid values	any string
Required	true

4.4.1.5.1.2 Attribute reliable

If this attribute is set to true, the channel sends all messages reliably. If not, data is sent only once (fire-and-forget).

Full path	OpenSplice/NetworkService/Channels/Channel[@reliable]
Format	boolean
Dimension	n/a
Default value	false

`Valid values`	true, false
`Required`	true
`Remarks`	This setting should be consistent over all nodes in the system

The specific channel a message is written into depends on the attached quality of service. Once a message has arrived in a channel, it will be transported with the quality of service attached to the channel. If the reliable attribute happens to be set to true, the message will be sent over the network using a reliability protocol.

4.4.1.5.1.3 Attribute enabled

This attribute toggles a channel on or off.

`Full path`	OpenSplice/NetworkService/Channels/Channel[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	true

Toggling a channel between enabled and disabled is a quick alternative for commenting out the corresponding lines in the configuration file.

4.4.1.5.1.4 Attribute default

This attribute indicates whether the channel is selected as the default channel in case no channel offers the quality of service requested by a message.

The network channels should be configured corresponding to the quality of service settings that are expected to be requested by the applications. It might happen, however, that none of the available channels meets the requested quality of service for a specific message. In that case, the message will be written into the default channel.

`Full path`	OpenSplice/NetworkService/Channels/Channel[@default]
`Format`	boolean
`Dimension`	n/a
`Default value`	false

`Valid values`	true, false
`Required`	false
`Remarks`	Only one channel is allowed to have this attribute set to true

4.4.1.5.1.5 Attribute priority

This attribute sets the transport priority of the channel.

Messages sent to the network have a transport_priority quality of service value. Selection of a network channel is based on the priority requested by the message and the priority offered by the channel. The priority settings of the different channels divide the priority range into intervals. Within a channel, messages are sorted in order of priority.

`Full path`	OpenSplice/NetworkService/Channels/Channel[@priority]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - maxInt
`Required`	false
`Remarks`	The specified priority value has no relation to the operating system threading priority.

4.4.1.5.1.6 Element *PortNr*

This element specifies the port number used by the Channel. Messages for the channel are sent to the port number given. Each channel needs its own unique port number. Please note that ‘reliable’ channels use a second port, which is the specified *PortNr* + 1.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ PortNr
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	n/a
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.7 Element AllowedPorts

This element specifies the port numbers available for the network service to be used by the reliable network channels. The network channel is configured with a unique port number. However, the reliable network channels require a second port number to provide the reliable communication service. For this second port number each reliable network channel will select a free port from the AllowedPorts.

When the AllowedPorts is not specified for a particular channel then the default AllowedPorts which is configured on the Channels element is used. Also when the default AllowedPorts is not specified each reliable network channel will first try to use the configured *portNr* + 1 as the second port or when this port number is already in use will determine a port number dynamically.

The AllowedPorts is a list of entries where an entry is a port number or a port number range.

When the AllowedPorts contains more than one entry then these entries must be seperated by commas (`,`). A port number range consists of the lower and the upper bound of the port number range, where the lower and the upper bound are seperated by a minus (`-`).

`Full path`	OpenSplice/NetworkService/Channels/Channel/AllowedPorts
`Format`	string containing ports or port ranges seperated by commas. A port range consists of a lower bound and an upper bound separated by a minus (`-`)
`Dimension`	n/a
`Default value`	n/a
`Valid values`	depends on operating system
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.8 Element *FragmentSize*

This element specifies the size of the fragments into which a large message will be divided by the channel before it is sent to the UDP stack. Operating system settings determine the maximum datagram size.

The human-readable option lets the user postfix the value with `K`(ilobyte), `M`(egabyte) or `G`(igabtye). For example, `10M` results in 10485760 bytes.

`Full path`	OpenSplice/NetworkService/Channels/Channel/FragmentSize
`Format`	unsigned integer, human-readable
`Dimension`	bytes
`Default value`	1300
`Valid values`	200 - 65536 (if operating system allows it)
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.9 Element *Resolution*

The resolution indicates the number of milliseconds that this channel sleeps between two consecutive resend or packing actions. Latency budget values are truncated to a multiple of *Resolution* milliseconds.

It is considered good practice to specify the Resolution consistently throughout the system.

`Full path`	OpenSplice/NetworkService/Channels/Channel/Resolution
`Format`	unsigned integer
`Dimension`	milliseconds
`Default value`	10
`Valid values`	1 - MaxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.10 Element *AdminQueueSize*

For reliable channels the receiving side needs to keep the sending side informed about the received data and the received control messages.

This is done by means of an "AdminQueue". This setting determines the size of this queue, and it must be greater than the maximum number of reliable messages send or received during each "Resolution" milliseconds.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ AdminQueueSize
`Format`	unsigned integer
`Dimension`	messages
`Default value`	4000
`Valid values`	400 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11 Element *Sending*

This element describes all properties for the transmitting side of the Channel.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Sending
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element CrcCheck* *Element QueueSize* *Element MaxBurstSize* *Element ThrottleLimit* *Element ThrottleThreshold* *Element MaxRetries* *Element RecoveryFactor* *Element DiffServField* *Element DontRoute* *Element TimeToLive* *Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.1 Element *CrcCheck*

This configuration element has been added in order to protect OpenSplice network packets from malicious attacks. CRCs (Cyclic Redundancy Checks) are specifically designed to protect against common types of errors on communication channels. When enabled, the integrity of delivered network packets from one DDS process to another is assured. There is a small performance cost to using this feature due to the addtional overhead of carrying out the CRC calculations.

When the sending side is enabled the network packet will contain a valid `crc` field.

`Full Path`	OpenSplice/NetworkService/Channel/Channels/Sending/CrcCheck
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.2 Element *QueueSize*

This element specifies the number of messages the network queue can contain.

`Full path`	OpenSplice/NetworkService/Channels/Channel/Sending/QueueSize
`Format`	unsigned integer
`Dimension`	messages
`Default value`	400
`Valid values`	1 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

The channel queue decouples DataWriters from the actual network processing, DataWriters will write messages into the queue and the network service will read the messages from the queue and marshal them onto the network according to the desired quality of service. The size of the queue will determine the amount of messages the channel can temporarily buffer in overload situations before blocking DataWriters. Note that a large queue size also increases the worst-case latency, so in general there is a trade off when blocking a DataWriter in favour of buffering more messages.

4.4.1.5.1.11.3 Element *MaxBurstSize*

Amount in bytes to be sent at maximum every 'Resolution' milliseconds. The default value is set to 1GB per resolution tick. This can be regarded as effectively unlimited, as it far exceeds the capacity of current physical networks.

The human-readable option lets the user postfix the value with *K*(ilobyte), *M*(egabyte) or *G*(igabtye). For example, *10M* results in 10485760 bytes.

Full path	OpenSplice/NetworkService/Channels/Channel/ Sending/MaxBurstSize
Format	unsigned integer, human-readable
Dimension	bytes/(resolution interval)
Default value	1073741823
Valid values	1024 - 1073741823
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.4 Element *ThrottleLimit*

Throttling is specific to reliable channels and will enable you to further limit (below MaxBurstSize) the amount of data that is sent every Resolution interval. This happens if a receiving Network Service indicates that it has trouble processing all incoming data. This value is the lower boundary of the range over which the throttling can adapt the limit. If this value is set to the same value as (or higher than) MaxBurstSize throttling is disabled.

The ThrottleLimit value is not allowed be smaller than the FragmentSize. If a lower value is provided, then the value of FragmentSize is used as ThrottleLimit.

Full path	OpenSplice/NetworkService/Channels/Channel/Sending/ThrottleLimit
Format	unsigned integer
Dimension	bytes/(resolution interval)
Default value	10240
Valid values	0 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.5 Element *ThrottleThreshold*

This is the number of unprocessed network fragments that a channel will store before it will inform the senders that it has trouble processing the incoming data. Those senders can use this information to adjust their throttle values, effectively reducing the amount of incoming data in case of a temporary overflow, and increasing again when the channel is able to catch up.

It is considered good practice to specify the ThrottleTreshold consistently throughout the system.

Full path	OpenSplice/NetworkService/Channels/Channel/Sending/ThrottleThreshold
Format	unsigned integer
Dimension	fragments
Default value	50
Valid values	2 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.6 Element *MaxRetries*

⚠ This element is only applicable for reliable channels.

A reliable channel implements a reliability protocol in which it builds a list of connected remote services. This protocol expects all connected services to acknowledge messages within a specific period of time, otherwise messages will be resent. This element specifies the number of retransmissions the service has to execute before considering that the addressed service has become unresponsive. When this happens the remote service will be removed from the reliability protocol and the channel will no longer expect messages to be acknowledged.

`Full path`	OpenSplice/NetworkService/Channels/Channel/Sending/MaxRetries
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	100
`Valid values`	1 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.7 Element *RecoveryFactor*

A reliable channel implements a reliability protocol in which it builds a list of connected remote services. This protocol expects all connected services to acknowledge messages within a specific period of time otherwise messages will be resent. The expected period of time is specified by this attribute as the number of resolution ticks. (See also Section 4.4.1.5.1.9, *Element Resolution*, on page 209.) The lost message is resent after Resolution * RecoveryFactor milliseconds.

`Full path`	OpenSplice/NetworkService/Channels/Channel/Sending/RecoveryFactor
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	3
`Valid values`	2 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.8 Element *DiffServField*

This element describes the DiffServ setting the channel will apply to all its network messages.

`Full path`	OpenSplice/NetworkService/Channels/Channel/Sending/DiffServField
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 255
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.9 Element *DontRoute*

The IP `DONTROUTE` socket option is set to the value specified.

`Full Path`	OpenSplice/NetworkService/Channels/Channel/Sending/DontRoute
`Format`	boolean
`Dimension`	n.a
`Default value`	true
`Valid values`	true, false
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.11.10 Element *TimeToLive*

For each UDP packet sent out, the IP Time To Live header value is set to the value specified.

Full Path	OpenSplice/NetworkService/Channels/Channel/Sending/TimeToLive
Format	unsigned integer
Dimension	n.a
Default value	0
Valid values	0-255
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.11 Element *Scheduling*

This element specifies the scheduling policies used to control the transmitter thread of the current Channel.

Full path	OpenSplice/NetworkService/Channels/Channel/Sending/Scheduling
Occurrences (min-max)	0 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.11.1 Element *Class*

This element specifies the thread scheduling class that will be used by the transmitter thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/NetworkService/Channel/Channels/Sending/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default

Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.11.11.2 Element *Priority*

This element specifies the thread priority that will be used by the transmitter thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/NetworkService/Channel/Channels/ Sending/Scheduling/Priority
Format	integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.4.1.5.1.11.11.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/NetworkService/Channel/Channels/ Sending/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.4.1.5.1.12 Element *Receiving*

This element describes all properties for the receiving side of the Channel.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Receiving
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element CrcCheck* *Element ReceiveBufferSize* *Element Scheduling* *Element DefragBufferSize* *Element SMPOptimization* *Element MaxReliabBacklog* *Element PacketRetentionPeriod* *Element ReliabilityRecoveryPeriod*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.1 Element *CrcCheck*

This configuration element protects OpenSplice network packets from malicious attacks. CRCs (Cyclic Redundancy Checks) are specifically designed to protect against common types of errors on communication channels. When enabled, the integrity of delivered network packets from one DDS process to another is assured. There is a small performance cost associated with the use of this feature due to the addtional overhead of carrying out the CRC calculations.

When the receiving side is enabled only network packets that contain a valid `crc` field are accepted.

`Full Path`	OpenSplice/NetworkService/Channel/Channels/ Receiving/CrcCheck
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.2 Element *ReceiveBufferSize*

The UDP receive buffer of the Channel socket is set to the value given. If many message are lost, the receive buffer size has to be increased.

The human-readable option lets the user postfix the value with `K`(ilobyte), `M`(egabyte) or `G`(igabtye). For example, `10M` results in 10485760 bytes.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Receiving/ReceiveBufferSize
`Format`	unsigned integer, human-readable
`Dimension`	bytes
`Default value`	1000000
`Valid values`	depends on operating system
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.3 Element *Scheduling*

This element specifies the scheduling policies used to control the receiver thread of the current Channel.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Receiving/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.3.1 Element *Class*

This element specifies the thread scheduling class that will be used by the receiver thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/NetworkService/Channels/Channel/Receiving/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.12.3.2 Element *Priority*

This element specifies the thread priority that will be used by the receiver thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/NetworkService/Channel/Channels/Receiving/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.4.1.5.1.12.3.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/NetworkService/Channel/Channels/ Receiving/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.4.1.5.1.12.4 Element *DefragBufferSize*

The maximum number of Fragment buffers that will be allocated for this channel. These buffers are used to store incoming fragments waiting to be processed, as well as fragments that are being processed.

With respect to very large messages be aware that the number of buffers times the fragment size must be sufficient to process the messages otherwise they will be dropped. (See also Section 4.4.1.5.1.8, *Element FragmentSize*, on page 209.)

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Receiving/DefragBufferSize
`Format`	unsigned integer
`Dimension`	fragments
`Default value`	5000
`Valid values`	500 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.5 Element *SMPOptimization*

This option will distribute the processing done for incoming fragements over multiple threads, which will lead to an improved throughput on SMP nodes.

`Full path`	OpenSplice/NetworkService/Channels/Channel/ Receiving/SMPOptimization
`Occurrences (min-max)`	0 - 1

Child-elements	<none>
Required attributes	*Attribute enabled*
Optional attributes	<none>

4.4.1.5.1.12.5.1 Attribute enabled

This attribute toggles the Optimization on or off.

Full path	OpenSplice/NetworkService/Channel/Channels/ Receiving/SMPOptimization/[@enabled]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	true

4.4.1.5.1.12.6 Element *MaxReliabBacklog*

This element specifies the maximum number of received fragments maintained in the channel from a single sender for the purpose of order preservation because an earlier fragment from that sender is missing. A sender is disconnected and all maintained fragments are discarded when this number is exceeded. Future fragments from this sender are only accepted after a disconnect if reconnection is set to true (see Section 4.4.1.3.2, *Element Reconnection*, on page 195).

Full path	OpenSplice/NetworkService/Channels/Channel/ Receiving/MaxReliabBacklog
Format	unsigned integer
Dimension	fragments
Default value	1000
Valid values	100 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.1.12.7 Element *PacketRetentionPeriod*

This element specifies how long received packets are retained by the network service for its so-called ‘reliability-under-publisher-crash’ extended reliability protocol. This protocol ensures that a consistent or *aligned* data-set is received by all alive (receiving) Network Services, even though some might not have received some packets at the moment a sender disappears (for whatever reason). The protocol implies that each node retains sufficient received data so that it can be (re-)distributed if a sender disappears before all receiving Network Services ‘up-to-date’. When the `PacketRetentionPeriod` element is set to `0` (the default value), the alignment amongst receiving Network Services will not occur. To activate full realibility, this setting must be configured to a time period that exceeds the worst-case death-detection time as configured for the discovery protocol of the set of distributed Network Services in the system.

`Full Path`	OpenSplice/NetworkService/Channel/Channels/Receiving/PacketRetentionPeriod
`Format`	unsigned integer
`Dimension`	ms
`Default value`	0 ms
`Valid values`	0 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.5.1.12.8 Element *ReliabilityRecoveryPeriod*

This element specifies a timeout period for the alignment phase of the extended reliability protocol, as explained in section 4.4.1.5.1.12.7, *Element PacketRetentionPeriod*. It only has an effect when the related `PacketRetentionperiod` is set to a non-zero value. After the specified `reliabilityRecoveryPeriod` timeout, any data retained for the purpose of alignment of receiving Network Services (following the disappearance or crash of a sending Network Service) will be discarded. The value of this timeout period must be sufficient to allow for the worst-case alignment-time of any ‘missed’ data by individual receiving Network Services following the disappearance of a sending Network Services in the system.

Full Path	OpenSplice/NetworkService/Channel/Channels/Receiving/ReliabilityRecoverPeriod
Format	unsigned integer
Dimension	ms
Default value	1000 ms
Valid values	0 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.5.2 Element AllowedPorts

See also Section 4.4.1.5.1.7, *Element AllowedPorts*, on page 208.

The AllowedPorts specified on the channels element applies to those channels that do not have an AllowedPorts element specified.

Full path	OpenSplice/NetworkService/Channels/AllowedPorts
Format	string containing ports or port ranges seperated by commas. A port range consists of a lower bound and an upper bound separated by a minus (-)
Dimension	n/a
Default value	n/a
Valid values	depends on operating system
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.6 Element *Discovery*

This element controls various parameters of the Network Services Discovery mechanism.

Discovery reduces the Network Service configuration and minimizes network traffic. Without Discovery, data is always sent to the network and all Networking Services need to configure the addresses[1] of all Network Services they need to communicate with. With Discovery, data is only sent to where interest exists and connectivity is discovered based on a minimum configuration[1] (see Section 4.4.1.6.4, *Element ProbeList*, on page 227).

Discovery is based on a heartbeat mechanism to advertize the service's availability. The Network Service starts by announcing its existence by sending heartbeats to the Global Partition[2] which is initially filled with the addresses specified in the ProbeList; remote Network Services receiving the heartbeat will start sending heartbeats in return. All Network Services that discover new heartbeats will automatically request address information that match their Scope (see Section 4.4.1.6.2, *Attribute Scope*, on page 226) from the Network Service sending the heartbeat, and add the retrieved address information to their Global Partition. Currently only uni-cast addresses are exchanged. Addresses are removed from the Global Partition when a remote Network Service stops and heartbeats are no longer received.

`Full path`	OpenSplice/NetworkService/Discovery
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element PortNr* *Element ProbeList* *Element Sending* *Element Receiving*
`Required attributes`	<none>
`Optional attributes`	*Attribute enabled* *Attribute Scope*

4.4.1.6.1 Attribute enabled

This element can be used to enable or disable Discovery. In case Discovery is disabled, entities will only detect each others presence implicitly once messages are received for the first time.

1. This can be multicast addresses and/or uni-cast addresses, especially in an uni-cast environment with many nodes the configuration of the Network Service's lists can be cumbersome.
1. Only a subset of addresses of nodes are initially specified, these nodes are assumed to be available as a discovery source, all nodes will make themselves known to these discovery nodes and thereby making its existence and address available for all other nodes.
2. The Global Partition contains all the addresses that the Network Service communicate with.

Full path	OpenSplice/NetworkService/Discovery[@enabled]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.4.1.6.2 Attribute Scope

This attribute implements a comma-separated list of string-matching expressions allowed to contain wild-card symbols.

Network Services that are not preconfigured but are discovered by receiving their heartbeats can dynamically join the Global Partition and participate in the communication space if the Role of the discovered Network Service matches one of the string expressions of this attribute (see Section 4.2.4, *Element Role*, on page 97).

Full path	OpenSplice/NetworkService/Discovery[@Scope]
Format	String
Dimension	n/a
Default value	n/a
Valid values	any string
Required	false

4.4.1.6.3 Element *PortNr*

This element specifies the Port number used by Discovery.

Full path	OpenSplice/NetworkService/Discovery/PortNr
Format	unsigned integer
Dimension	n/a
Default value	3369
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.6.4 Element *ProbeList*

This element contains a list of addresses that is added to the Global Partition at startup. So discovery heartbeats are sent to these addresses and any Network Services available at these addreses will subsequently exchange discovered addresses according to the Discovery mechanism (see Section 4.4.1.6, *Element Discovery*, on page 224). Multiple ProbeList addresses can be entered by separating them with commas (,), semicolons (;), or spaces ( ). The addresses can be entered as 'dotted decimal' IPv4 or 'colon-separated hexadecimal' IPv6 notation or as the symbolic `hostname`, in which case the middleware will try to resolve the corresponding IP address

`Full path`	OpenSplice/NetworkService/Discovery/ProbeList
`Format`	String
`Dimension`	n/a
`Default value`	
`Valid values`	any string
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5 Element *Sending*

This element describes all properties for transmitting Discovery heartbeats.

`Full path`	OpenSplice/NetworkService/Discovery/Sending
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Interval* *Element SafetyFactor* *Element SalvoSize* *Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5.1 Element *Interval*

This element specifies the interval at which heartbeats will be sent. This interval is also communicated as part of a heartbeat to remote Network Services so they know when to expect the next heartbeat.

`Full path`	OpenSplice/NetworkService/Discovery/Sending/Interval
`Format`	unsigned integer
`Dimension`	milliseconds
`Default value`	333
`Valid values`	100 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5.2 Element *SafetyFactor*

The SafetyFactor is used to set a margin (< 1) on the expected heartbeat interval. The actual interval at which the heartbeats are sent is the specified interval multiplied by this factor, so the actual interval will be equal to or smaller than the specified value. This can be used to avoid timing issues such as those caused by typical scheduling or network latencies.

`Full path`	OpenSplice/NetworkService/Discovery/Sending/SafetyFactor
`Format`	float
`Dimension`	n/a
`Default value`	0.9
`Valid values`	0.2 - 1.0
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5.3 Element *SalvoSize*

The reactivity of discovery depends on the heartbeat frequency, a higher heartbeat frequency gives a faster reactivity but also imposes a higher network load, which is not desirable. Ideally the heartbeat frequency must be kept as low as possible but from a startup (and shutdown) perspective a high reactivity is often desired. So the Network Service has the capability to send an additional salvo of heartbeats at startup and shutdown at ten times the normal heartbeat speed to maximize reactivity during these phases without requiring a continuous high heartbeat frequency. The SalvoSize sets the number of messages to send during these phases.

`Full path`	OpenSplice/NetworkService/Discovery/Sending/SalvoSize
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	3
`Valid values`	1 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5.4 Element *Scheduling*

This element specifies the scheduling policies used to control the Discovery transmit thread.

`Full path`	OpenSplice/NetworkService/Discovery/Sending/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.5.4.1 Element *Class*

This element specifies the thread scheduling class that will be used by the Discovery transmit thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/NetworkService/Discovery/Sending/Scheduling/Class
Format	enumeration
Dimension	n/a
Default value	Default
Valid values	Timeshare, Realtime, Default
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.6.5.4.2 Element *Priority*

This element specifies the thread priority that will be used by the Discovery transmit thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/NetworkService/Discovery/Sending/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.4.1.6.5.4.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/NetworkService/Discovery/Sending/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.4.1.6.6 Element *Receiving*

The Receiving element describes all Discovery properties for processing received heartbeats.

`Full path`	OpenSplice/NetworkService/Discovery/Receiving
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element DeathDetectionCount* *Element ReceiveBufferSize* *Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.6.1 Element *DeathDetectionCount*

A Network Service has a configured heartbeat interval period meaning that after that period a new heartbeat will be sent (see Section 4.4.1.6.5.1, *Element Interval*, on page 228). This element specifies the number of times the interval period must pass without receiving a new heartbeat to consider the remote Domain Service to be dead[1].

`Full path`	OpenSplice/NetworkService/Discovery/Receiving/DeathDetectionCount
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	6
`Valid values`	1 - maxInt

1. A Domain Service is also considered dead if communication fails and the Domain Service is no longer visible.

`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.6.2 Element *ReceiveBufferSize*

The UDP receive buffer of Discovery is set to the value given. If many message are lost, the receive buffer size has to be increased.

`Full path`	OpenSplice/NetworkService/Discovery/Receiving/ReceiveBufferSize
`Format`	unsigned integer
`Dimension`	bytes
`Default value`	1000000
`Valid values`	depends on operating system
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.6.3 Element *Scheduling*

This element specifies the scheduling policies used to control the receiver thread of Discovery.

`Full path`	OpenSplice/NetworkService/Discovery/Receiving/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.6.3.1 Element *Class*

This element specifies the thread scheduling class that will be used by the receiver thread of Discovery heartbeats. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/NetworkService/Discovery/Receiving/ Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.6.6.3.2 Element *Priority*

This element specifies the thread priority that will be used by the receiver thread of Discovery hearbeats. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/NetworkService/Discovery/Receiving/ Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.4.1.6.6.3.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/NetworkService/Discovery/Receiving/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.4.1.7 Element *Tracing*

This element controls the amount and type of information that is written into the tracing log by the Network Service. This is useful to track the Network Service during application development. In the runtime system it should be turned off.

Full path	OpenSplice/NetworkService/Tracing
Occurrences (min-max)	0 - 1
Child-elements	*Element OutputFile* *Element Timestamps* *Element Categories*
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.1 Element *OutputFile*

This option specifies where the logging is printed to. Note that “stdout” is considered a legal value that represents “standard out”.

Full path	OpenSplice/NetworkService/Tracing/OutputFile
Format	string
Dimension	file name
Default value	networking.log
Valid values	depends on operating system.
Occurrences (min-max)	0 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.2 Element *Timestamps*

This element specifies whether the logging must contain timestamps.

`Full path`	OpenSplice/NetworkService/Tracing/Timestamps
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute Absolute*

4.4.1.7.2.1 Attribute Absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

`Full path`	OpenSplice/NetworkService/Tracing/Timestamps[@absolute]
`Format`	boolean
`Dimension`	
`Default value`	true
`Valid values`	true, false
`Required`	false

4.4.1.7.3 Element *Categories*

This element contains the logging properties for various categories.

Full path	OpenSplice/NetworkService/Tracing/Categories
Occurrences (min-max)	0 - 1
Child-elements	*Element Default* *Element Configuration* *Element Construction* *Element Destruction* *Element Mainloop* *Element Groups* *Element Send* *Element Receive* *Element Throttling* *Element Test* *Element Discovery*
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.3.1 Element *Default*

This element specifies the tracing level used for categories that are not explicitly specified. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

Full path	OpenSplice/NetworkService/Tracing/Categories /Default
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - 6
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.3.2 Element *Configuration*

This element specifies the tracing level for the *Configuration* category. This includes the processing of all NetworkService parameters in the config file. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Configuration
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.3 Element *Construction*

This element specifies the tracing level for the *Construction* category. This includes the creation of all internal processing entities. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Construction
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.4 Element *Destruction*

This element specifies the tracing level for the *Destruction* category. This includes the destruction of all internal processing entities when the NetworkService terminates. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Destruction
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.5 Element *Mainloop*

This element specifies the tracing level for the *Mainloop* category. This includes information about each of the threads spawned by the NetworkService. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Mainloop
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.6 Element *Groups*

This element specifies the tracing level for the *Groups* category. This includes the management of local groups (partition-topic combinations). Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

Full path	OpenSplice/NetworkService/Tracing/Categories/Groups
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - 6
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.3.7 Element *Send*

This element specifies the tracing level for the *Send* category. This includes information about outgoing data. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

Full path	OpenSplice/NetworkService/Tracing/Categories/Send
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - 6
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.3.8 Element *Receive*

This element specifies the tracing level for the *Receive* category. This includes information about incoming data. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Receive
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.9 Element *Throttling*

This element specifies the tracing level for the *Throttling* category. This includes information about incoming data. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

`Full path`	OpenSplice/NetworkService/Tracing/Categories /Throttling
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 6
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.1.7.3.10 Element *Test*

This element specifies the tracing level for the *Test* category. This is a reserved category used for testing purposes. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

Full path	OpenSplice/NetworkService/Tracing/Categories/Test
Format	unsigned integer
Dimension	
Default value	0
Valid values	0 - 6
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.7.3.11 Element *Discovery*

This element specifies the tracing level for the *Discovery* category. This includes all activity related to Discovery. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

Full path	OpenSplice/NetworkService/Tracing/Categories/Discovery
Format	unsigned integer
Dimension	n/a
Default value	0
Valid values	0 - 6
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.4.1.8 Element *Compression*

This element contains configuration for compression in the Network Service.

Full path	OpenSplice/NetworkService/Compression
Occurrences (min-max)	0 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute PluginLibrary* *Attribute PluginInitFunction* *Attribute PluginParameter*

4.4.1.8.1 Attribute PluginLibrary

This attribute specifies a library to load at run time, in which a compression plugin may be found.

`Full path`	OpenSplice/NetworkService/Compression[@PluginLibrary]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	any string
`Required`	false

4.4.1.8.2 Attribute PluginInitFunction

This attribute specifies the name of a compression plugin initialisation function, or the name of one of the built-in compressors.

`Full path`	OpenSplice/NetworkService/Compression[@PluginInitFunction]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	any string
`Required`	false

4.4.1.8.3 Attribute PluginParameter

This attribute specifies a parameter that is to be passed to a compressor. The interpretation of this may vary from one compressor to another.

`Full path`	OpenSplice/NetworkService/Compression[@PluginParameter]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	any string
`Required`	false

4.4.2 The Secure Network Service

The Secure Network Service is an extended version of the Network Service (see Section 4.4.1, *The Network Service*, on page 189) which adds security capabilities to the communication between DDS service instances. The Secure Network Service provides for secure communication between the DDS service instances by using encryption of the data that is being exchanged by using a configurable cipher. Besides encryption the Secure Network Service provides authentication and access control of the data that is being exchanged between the DDS service instances.

The Secure Network Service is selected by using the following configuration element to the Domain section of the configuration file (see Section 4.2.10, *Element Application*, on page 111).

```
<Service name="snetworking">
      <Command>snetworking</Command>
</Service>
```

All the configuration settings that apply to the Network Service also apply to the Secure Network Service. Thus all the configuration parameters that are discussed in Section 4.4.1, *The Network Service*) also apply to the Secure Network Service.

This section will only discuss the *additional* configuration parameters that are applicable to the Secure Network Service.

The Secure Network Service configuration expects a root element named *`OpenSplice/SNetworkService`*, so when configuring the Secure Network Service you must use this root element.

4.4.2.1 Element *Partitioning*

The OpenSplice Secure Network Service is capable of leveraging the network’s multicast and routing capabilities. If some *a priori* knowledge about the participating nodes and their topic and partition interest is available, then the Network Services in the system can be explicitly instructed to use specific unicast or multicast addresses for its network traffic. This is done by means of so-called network partitions.

A network partition is defined by one or more unicast, multicast or broadcast IP addresses. Any Secure Network Service that is started will read the network partition settings and, if applicable, join the required multicast groups. For every sample distributed by the Secure Network Service, both its partition and topic name will be inspected. In combination with a set of network partition mapping rules, the service will determine which network partition the sample is written to. The mapping rules are configurable as well.

Using network configuration, nodes can be disconnected from any network partition. If a node is connected via a low speed interface, it is not capable of receiving high volume data. If the DCPS partitioning is designed carefully, high volume data is published into a specific partition, which in its turn is mapped onto a specific network partition, which is itself only connected to those nodes that are capable of handling high volume data.

`Full path`	OpenSplice/SNetworking/Partitioning
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element GlobalPartition* *Element NetworkPartitions*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.2.1.1 Element *GlobalPartition*

In the context of the Secure Network Service the *GlobalPartition* element (see Section 4.4.1.4.1, *Element GlobalPartition*, on page 197) has an additional optional attribute named *SecurityProfile*.

`Full path`	OpenSplice/SNetworking/Partitioning/ GlobalPartition
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	*Attribute Address*
`Optional attributes`	*Attribute SecurityProfile*

4.4.2.1.1.1 Attribute Address

See Section 4.4.1.4.1.1, *Attribute Address*, on page 197 for a full description of this attribute.

Full path	OpenSplice/SNetworking/Partitioning/GlobalPartition[@Address]
Format	‘dotted decimal’ IPv4 or ‘colon-separated hexadecimal’ IPv6 address, symbolic host name or “broadcast”
Dimension	n/a
Default value	"broadcast"
Valid values	"broadcast", any ‘dotted decimal’ IPv4 or ‘colon-separated hexadecimal’ IPv6 unicast or multicast address, or a resolvable symbolic hostname
Required	true
Remarks	The given interface should have the required capabilities, *e.g.* broadcasting or multicasting

4.4.2.1.1.2 Attribute SecurityProfile

In the context of the Secure Network Service, the *GlobalPartition* element provides support for the attribute *SecurityProfile*. The attribute references a security profile declared in the context of the Security element. If the given reference is invalid, the global partition configuration is invalid. In this case, the partition will be blocked to prevent unwanted information leaks. A configuration error message will be logged to the `ospl-error.log` file. If the security feature has been enabled, but no profile is declared, then the `NULL` profile is used by default; this means that no security is added to the transport.

Full path	OpenSplice/SNetworking/Partitioning/NetworkPartitions/GlobalPartition[@SecurityProfile]
Format	string
Dimension	n/a
Default value	nullProfile
Valid values	any valid Security Profile name
Required	false

4.4.2.1.2 Element *NetworkPartitions*

Network configuration can contain a set of network partitions, which are grouped under the NetworkPartitions element.

Full path	OpenSplice/SNetworking/Partitioning/NetworkPartitions
Occurrences (min-max)	0 - 1
Child-elements	*Element NetworkPartition*
Required attributes	<none>
Optional attributes	<none>

4.4.2.1.2.1 Element *NetworkPartition*

In the context of the Secure Network Service the *NetworkPartition* element (see Section 4.4.1.4.2.1, *Element NetworkPartition*, on page 198) has an additional optional attribute named *SecurityProfile*.

Full path	OpenSplice/SNetworking/Partitioning/NetworkPartitions/NetworkPartition
Occurrences (min-max)	0 - *
Child-elements	<none>
Required attributes	*Attribute Address* *Attribute Connected*
Optional attributes	*Attribute SecurityProfile*

4.4.2.1.2.1.1 Attribute Address

See Section 4.4.1.4.2.1.1, *Attribute Address*, on page 199 for a full description of this attribute.

Full path	OpenSplice/SNetworking/Partitioning/NetworkPartitions/NetworkPartition[@Address]
Format	‘dotted decimal’ IPv4 or ‘colon-separated hexadecimal’ IPv6 address, symbolic host name or "broadcast"
Dimension	n/a
Default value	"broadcast"
Valid values	"broadcast", any ‘dotted decimal’ IPv4 or ‘colon-separated hexadecimal’ IPv6 unicast or multicast address or resolvable symbolic hostname
Required	true
Remarks	The given interface should have the required capabilities, *e.g.* broadcasting or multicasting.

4.4.2.1.2.1.2 Attribute Connected

See Section 4.4.1.4.2.1.2, *Attribute Connected*, on page 199 for a full description of this attribute.

`Full path`	OpenSplice/SNetworking/Partitioning/NetworkPartitions/ NetworkPartition[@Connected]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	true

4.4.2.1.2.1.3 Attribute SecurityProfile

In the context of the Secure Network Service, the *NetworkPartition* element provides support for the attribute SecurityProfile. The attribute references a securityprofile declared in the context of the Security element. If the given reference is invalid, the network partition configuration is invalid. In this case the partition will be blocked to prevent unwanted information leaks. A configuration error message will be logged to the `ospl-error.log` file. If the security feature has been enabled but no profile is declared, the `NULL` profile will be used by default. The ordering of network partition declarations in the OSPL configuration file must be the same for all nodes within the OpenSplice domain. If certain nodes shall not use one of the network partitions, the network partition in question must be declared as `connected = 'false'`. In this case the declared security profile would not be evaluated or initialized, and the associated secret cipher keys need not to be defined for the OpenSplice node in question.

`Full path`	OpenSplice/SNetworking/Partitioning/NetworkPartitions/N etworkPartition[@SecurityProfile]
`Format`	string
`Dimension`	n/a
`Default value`	nullProfile
`Valid values`	any valid Security Profile name
`Required`	false

4.4.2.2 Element *Security*

This element is only relevant for the Secure Network Service and the normal (non-secure) Network Service will ignore this element. Note that the Secure Network Service also will behave like the normal Network Service if no *Security* element or an empty *Security* element is specified.

`Full path`	OpenSplice/SNetworking/Security
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element SecurityProfile* *Element AccessControl* *Element Authentication*
`Required attributes`	<none>
`Optional attributes`	*Attribute enabled*

4.4.2.2.1 Attribute enabled

This is an optional attribute. If not defined it defaults to *true* and all network partitions, if not specified otherwise, will be encoded using the NULL cipher. The NULL cipher does not provide for any level of integrity or confidentiality; message items will be sent unencrypted. If enabled = '`false`' the security feature will not be activated, and the Network Service acts like any other OpenSplice node not being security aware. Security profiles defined in the configuration file will not take effect, but will cause the system to log warnings.

`Full path`	OpenSplice/SNetworking/Security[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	false

4.4.2.2.2 Element ***SecurityProfile***

This element defines the security profile which can be applied to one or more network partitions. This element is optional.

`Full path`	OpenSplice/SNetworking/Security/ SecurityProfile
`Occurrences (min-max)`	0 - *

Child-elements	<none>
Required attributes	*Attribute Name* *Attribute Cipher* *Attribute cipherKey*
Optional attributes	<none>

4.4.2.2.2.1 Attribute Name

This is a mandatory attribute. The name must be unique for all Security Profiles being declared. If the name is not specified, the security profile will be ignored as it cannot be referenced anyway.

Full path	OpenSplice/SNetworking/Security/SecurityProfile[@Name]
Format	string
Dimension	n/a
Default value	“aSecurityProfile”
Valid values	any string
Required	true

4.4.2.2.2.2 Attribute Cipher

This is a mandatory attribute. Depending on the declared cipher, the cipher key must have a specific length (128 bits, 192 bits, or 256 bits) or none at all. The following case-insensitive values are supported by the current implementation:

- ***aes128***, implements AES cipher with a 128-bit cipher-key (16 Bytes, 32 hexadecimal characters). This cipher will occupy 34 bytes of each UDP packet being sent.
- ***aes192***, implements the AES cipher with a 192-bit cipher-key (24 Bytes, 48 hexadecimal characters). This cipher will occupy 34 bytes of each UDP packet being sent.
- ***aes256***, implements the AES cipher with a 256-bit cipher-key (32 Bytes, 64 hexadecimal characters. This cipher will occupy 34 bytes of each UDP packet being sent.
- ***blowfish***, implements the Blowfish cipher with a 128-bit cipher-key (16 Bytes, 32 hexadecimal characters). This cipher will occupy 26 bytes of each UDP packetb eing sent.
- ***null***, implements the NULL cipher. The only cipher that does not require a cipher-key. This cipher will occupy 4 bytes of each UDP packet being sent.

All ciphers except for the NULL cipher are combined with SHA1 to achieve data integrity. Also, the *rsa-* prefix can be added to the ciphers. In this case, digital signatures using RSA will be available.

`Full path`	OpenSplice/SNetworking/Security/ SecurityProfile[@Cipher]
`Format`	string
`Dimension`	n/a
`Default value`	“null”
`Valid values`	aes128, aes192, aes256, blowfish, rsa-aes128, rsa-aes192, rsa-aes256, rsa-blowfish, null
`Required`	true

4.4.2.2.2.3 Attribute cipherKey

The cipherKey attribute is used to define the secret key required by the declared cipher. The value can be a URI referencing an external file containing the secret key, or the secret key can be defined in-place directly as a string value.

The key must be defined as a hexadecimal string, each character representing 4 bits of the key, for example. `1ABC` represents the 16-bit key `0001 1010 1011 1100`. The key must not follow a well-known pattern and must match exactly the key length required by the chosen cipher. In case of malformed cipher-keys, the security profile in question will be marked as invalid. Moreover, each network partition referring to the invalid Security Profile will not be operational and thus traffic will be blocked to prevent information leaks. As all OpenSplice applications require read access to the XML configuration file, for security reasons it is recommended to store the secret key in an external file in the file system, referenced by the URI in the configuration file. The file must be protected against read and write access from other users on the host. Verify that access rights are not given to any other user or group on the host.

Alternatively, storing the secret key in-place in the XML configuration file will give read/write access to all DDS applications joining the same OpenSplice node. Because of this, the ‘in-place’ method is strongly discouraged.

`Full path`	OpenSplice/SNetworking/Security/ SecurityProfile[@cipherKey]
`Format`	string
`Dimension`	n/a

`Default value`	“” (empty string)
`Valid values`	any string of the correct length
`Required`	true

4.4.2.2.3 Element *AccessControl*

The optional AccessControl element defines settings for access control enforcement and which access control module shall be used.

`Full path`	OpenSplice/SNetworking/Security/ AccessControl
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element AccessControlModule*
`Required attributes`	<none>
`Optional attributes`	*Attribute enabled* *Attribute policy*

4.4.2.2.3.1 Attribute enabled

The access control feature will be activated when enabled = ‘true’.

`Full path`	OpenSplice/SNetworking/Security/ AccessControl[@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Required`	false

4.4.2.2.3.2 Attribute policy

The policy attribute references a file containing the access control policy.

`Full path`	OpenSplice/SNetworking/Security/ AccessControl[@policy]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	any valid policy name
`Required`	false

4.4.2.2.3.3 Element *AccessControlModule*

The AccessControlModule element determines which access control module(s) will be used. More than one module may be defined. All defined and enabled modules will be used to determine whether access should be granted.

Full path	OpenSplice/SNetworking/Security/ AccessControl/AccessControlModule
Occurrences (min-max)	0 - *
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute enabled* *Attribute type*

4.4.2.2.3.3.1 Attribute enabled

The module specified in the type attribute is used to evaluate access control rules when enabled = ‘true’.

Full path	OpenSplice/SNetworking/Security/AccessControl/ AccessControlModule[@enabled]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.4.2.2.3.3.2 Attribute type

The type attribute defines the access control model type. Currently, OpenSplice only supports mandatory access control; accordingly the only valid value for this attribute is ‘MAC’.

Full path	OpenSplice/SNetworking/Security/AccessControl/ AccessControlModule[@type]
Format	string
Dimension	n/a
Default value	“none”
Valid values	“MAC”
Required	false

4.4.2.2.4 Element *Authentication*

The optional Authentication element defines whether additional sender authorization shall be performed. Enabling Authentication requires that a cipher, including RSA (such as *rsa-aes256*), is used.

Full path	OpenSplice/SNetworking/Security/ Authentication
Occurrences (min-max)	0 - 1
Child-elements	*Element X509Authentication*
Required attributes	<none>
Optional attributes	*Attribute enabled*

4.4.2.2.4.1 Attribute enabled

Authentication is performed when enabled is set to ‘true’.

Full path	OpenSplice/SNetworking/Security/ Authentication[@enabled]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.4.2.2.4.2 Element *X509Authentication*

The X509Authentication element defines where keys and certificates required for X509 authentication may be found.

Full path	OpenSplice/SNetworking/Security/ Authentication/X509Authentication
Occurrences (min-max)	0 - 1
Child-elements	*Element Credentials* *Element TrustedCertificates*
Required attributes	<none>
Optional attributes	<none>

4.4.2.2.4.2.1 Element *Credentials*

The Credentials element is an optional element. If it is missing, then the node does not sign messages (in other words, it does not send credentials).

`Full path`	OpenSplice/SNetworking/Security/ Authentication/X509Authentication/Credentials
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Key* *Element Cert*
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.2.2.4.2.1.1 Element *Key*

⚠ The Key element references the file containing the key. It is recommended that the absolute path is used. A relative path will be interpreted relative to the directory from which the OpenSplice daemon is started.

`Full path`	OpenSplice/SNetworking/Security/ Authentication/X509Authentication/ Credentials/Key
`Occurrences (min-max)`	1 - 1
`Child-elements`	
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.2.2.4.2.1.2 Element *Cert*

⚠ The Cert element references the file containing the certificate. It is recommended that the absolute path is used. A relative path will be interpreted relative to the directory from which the OpenSplice daemon is started.

`Full path`	OpenSplice/SNetworking/Security/ Authentication/X509Authentication/ Credentials/Cert
`Occurrences (min-max)`	1 - 1
`Child-elements`	
`Required attributes`	<none>
`Optional attributes`	<none>

4.4.2.2.4.2.2 Element *TrustedCertificates*

⚠ The TrustedCertificates element references a file containing the trusted certificates. It is recommended that the absolute path is used. A relative path will be interpreted relative to the directory from which the OpenSplice daemon is started. Since the file is looked for on the local file system, it will accept a file name in the notation that is supported by that particular Operating System.

`Full path`	OpenSplice/SNetworking/Security/ Authentication/X509Authentication/ TrustedCertificates
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	valid path to certificate file(s)
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5 The Tuner Service

The TunerService configuration determines how the Tuner Service handles the incoming client connections. It expects a root element named *`OpenSplice/TunerService`*, in which several child-elements may be specified. Each of these are listed and explained.

`Full path`	OpenSplice/TunerService
`Occurrences (min-max)`	0 - *
`Child-elements`	*Element Client* *Element Server* *Element GarbageCollector* *Element LeaseManagement* *Element Watchdog*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.5.1 Attribute *name*

This attribute identifies a configuration for the Tuner Service by name. Multiple Tuner Service configurations can be specified in one single resource file. The actual applicable configuration is determined by the value of the *name* attribute, which must match the one specified under the attribute *OpenSplice/Domain/Application[@name]* in the configuration of the Domain Service.

Full path	OpenSplice/TunerService[@name]
Format	string
Dimension	n/a
Default value	cmsoap
Valid values	any string
Required	true

4.5.2 Element *Client*

This element determines how the Tuner service handles the incoming client connections.

Full path	OpenSplice/TunerService/Client
Occurrences (min-max)	0 - 1
Child-elements	*Element LeasePeriod* *Element MaxClients* *Element MaxThreadsPerClient* *Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.5.2.1 Element *LeasePeriod*

This element determines the maximum amount of time in which a connected client must update its lease. This can be done implicitly by calling any function or explicitly by calling the update lease function. The Tuner tool will automatically update its lease when it is connected to the Tuner Service. This ensures that all resources are cleaned up automatically if the client fails to update its lease within this period.

Full path	OpenSplice/TunerService/Client/LeasePeriod
Format	float
Dimension	seconds
Default value	15.0
Valid values	10.0 - maxFloat
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.5.2.2 Element *MaxClients*

This element determines the maximum allowed number of clients that are allowed to be concurrently connected to the Tuner Service. Clients are identified by IP-address.

Full path	OpenSplice/TunerService/Client/MaxClients
Format	unsigned integer
Dimension	n/a
Default value	2
Valid values	1 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.5.2.3 Element *MaxThreadsPerClient*

This element specifies the maximum number of threads that the Tuner Service will create for one specific client. The number of threads determines the maximum number of concurrent requests for a client.

Full path	OpenSplice/TunerService/Client/ MaxThreadsPerClient
Format	unsigned integer
Dimension	n/a
Default value	2

`Valid values`	1 - maxInt
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.2.4 Element *Scheduling*

This element specifies the scheduling policies used to control the threads that handle the client requests to the Tuner Service.

`Full path`	OpenSplice/TunerService/Client/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.2.4.1 Element *Class*

This element specifies the thread scheduling class that will be used by the threads that handle client requests to the Tuner Service. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/TunerService/Client/Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.2.4.2 Element *Priority*

This element specifies the thread priority that will be used by the threads that handle client requests to the Tuner Service. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/TunerService/Client/Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.5.2.4.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/TunerService/Client/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Required`	false

4.5.3 Element *Server*

This element determines the server side behaviour of the Tuner Service.

Full path	OpenSplice/TunerService/Server
Occurrences (min-max)	0 - 1
Child-elements	*Element Backlog* *Element PortNr* *Element Verbosity*
Required attributes	<none>
Optional attributes	<none>

4.5.3.1 Element *Backlog*

This element determines the maximum number of client requests that are allowed to be waiting when the maximum number of concurrent requests is reached.

Full path	OpenSplice/TunerService/Server/Backlog
Format	unsigned integer
Dimension	n/a
Default value	5
Valid values	0 - maxInt
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.5.3.2 Element *PortNr*

This element specifies the port number that the TunerService will use to listen for incoming requests. This port number must also be used by the Tuner tool to connect to this service.

When using the single process option set this value to `Auto`.

Full path	OpenSplice/TunerService/Server/PortNr
Format	unsigned integer
Dimension	n/a
Default value	8000
Valid values	`1 - 65535` (depends on operating system) *or* `Auto`
Occurrences (min-max)	0 - 1

`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.3.3 Element *Verbosity*

This element specifies the verbosity level of the logging of the service.

`Full path`	OpenSplice/TunerService/Server/Verbosity
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	0
`Valid values`	0 - 5
`Occurrences (min-max)`	0 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.4 Element *GarbageCollector*

This element specifies the behaviour of the garbage collection thread of the service.

`Full path`	OpenSplice/TunerService/GarbageCollector
`Occurrences (min-max)`	0 - 1
`Child-elements`	*Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.4.1 Element *Scheduling*

This element specifies the scheduling policies used to control the garbage collection thread of the Tuner Service.

`Full path`	OpenSplice/TunerService/GarbageCollection/ Scheduling
`Occurrences (min-max)`	1 - 1

`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.4.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the garbage collection thread of the Tuner Service. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/TunerService/GarbageCollection/ Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.4.1.2 Element *Priority*

This element specifies the thread priority that will be used by the garbage collection thread of the Tuner Service. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/TunerService/GarbageCollection/ Scheduling/ Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1

Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.5.4.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/TunerService/GarbageCollection/Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.5.5 Element *LeaseManagement*

This element specifies the behaviour of the lease management thread of the Tuner Service.

Full path	OpenSplice/TunerService/LeaseManagement
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.5.5.1 Element *Scheduling*

This element specifies the scheduling policies used to control the lease management thread of the Tuner Service.

Full path	OpenSplice/TunerService/LeaseManagement/Scheduling
Occurrences (min-max)	1 - 1

`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.5.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the lease management thread of the Tuner Service. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/TunerService/LeaseManagement/ Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.5.1.2 Element *Priority*

This element specifies the thread priority that will be used by the lease management thread of the Tuner Service. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/TunerService/LeaseManagement/ Scheduling/ Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1

Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute priority_kind*

4.5.5.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

Full path	OpenSplice/TunerService/LeaseManagement/ Scheduling/Priority[@priority_kind]
Format	enumeration
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.5.6 Element *Watchdog*

This element controls the characteristics of the Watchdog thread.

Full path	OpenSplice/TunerService/Watchdog
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.5.6.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

Full path	OpenSplice/TunerService/Watchdog/ Scheduling
Occurrences (min-max)	1 - 1
Child-elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.5.6.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/TunerService/Watchdog/ Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.5.6.1.2 Element *Priority*

This element specifies the thread priority that will be used by the watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/TunerService/Watchdog/ Scheduling/Priority
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	depends on operating system
`Valid values`	depends on operating system
`Occurrences (min-max)`	1 - 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind*

4.5.6.1.2.1 Attribute priority_kind

This attribute specifies whether the specified *Priority* is a relative or absolute priority.

`Full path`	OpenSplice/TunerService/Watchdog/Scheduling/Priority[@priority_kind]
`Format`	enumeration
`Dimension`	n/a
`Default value`	Relative
`Valid values`	Relative, Absolute
`Occurrences (min-max)`	0 - 1
`Required`	false

4.6 The DbmsConnect Service

The DbmsConnect Service configuration is responsible for DDS to DBMS bridging and expects a root element named *`OpenSplice/DbmsConnectService`*. Within this root element, the DbmsConnect Service will look for several child-elements. Each of these is listed and explained.

`Full path`	OpenSplice/DbmsConnectService
`Occurrences (min-max)`	0 - *
`Child-elements`	*Element DdsToDbms* *Element DbmsToDds* *Element Tracing* *Element Watchdog*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.6.1 Attribute *name*

This attribute identifies the configuration for the DBMS Service by name. Multiple DBMS Service configurations can be specified in one single resource file. The actual applicable configuration is determined by the value of the *name* attribute, which must match the one specified under the attribute *OpenSplice/Domain/Application[@name]* in the configuration of the Domain Service.

`Full path`	OpenSplice/DBMSConnectService[@name]
`Format`	string
`Dimension`	n/a

Default value	n/a
Valid values	any string
Required	true

4.6.2 Element *DdsToDbms*

This element specifies the configuration properties concerning DDS to DBMS bridging.

Full path	OpenSplice/DBMSConnectService/DdsToDbms
Occurrences (min-max)	0 - 1
Child-elements	*Element NameSpace*
Required attributes	<none>
Optional attributes	*Attribute replication_mode*

4.6.2.1 Attribute replication_mode

This attribute specifies the default replication mode for all NameSpaces in the *DdsToDbms* element.

When replicating databases through DDS, the *NameSpace* elements in the *DbmsToDds* and *DdsToDbms* elements map a Table and Topic circularly. To prevent data-modifications from continuously cascading, modifications made by the DBMSConnect service itself should not trigger new updates in the DBMS nor in the DDS.

In replication mode, the DbmsConnect service ignores samples that were published by itself. (Currently this means that everything that is published on the same node as the DBMSConnect Service is considered to be of DBMSConnect origin and therefore ignored). That way, replication of changes that were copied from Dbms to DDS back into Dbms is avoided.

WARNING: This setting does not avoid replication of changes that were copied from DDS to Dbms back into DDS. For that purpose, the *replication_user* attribute of the *DbmsToDds* or *DbmsToDds/NameSpace* elements should be set appropriately as well!

Full path	OpenSplice/DbmsConnectService/ DdsToDbms[@replication_mode]
Format	boolean
Dimension	n/a
Default value	false

Valid values	true, false
Remarks	none
Required	no

4.6.2.2 Element *NameSpace*

This element specifies the responsibilities of the service concerning the bridging of particular data from DDS to DBMS. At least one *NameSpace* element has to be present in a *DdsToDbms* element.

Full path	OpenSplice/DBMSConnectService/DdsToDbms/ NameSpace
Occurrences (min-max)	1 - *
Child-elements	*Element Mapping*
Required attributes	*Attribute dsn* *Attribute usr* *Attribute pwd*
Optional attributes	*Attribute name* *Attribute partition* *Attribute topic* *Attribute schema* *Attribute catalog* *Attribute replication_mode* *Attribute update_frequency* *Attribute odbc*

4.6.2.2.1 Attribute dsn

Represents the ODBC Data Source Name, that represents the DBMS where the service must bridge the DDS data to.

Full path	OpenSplice/DbmsConnectService/DdsToDbms/ NameSpace[@dsn]
Format	string
Dimension	Data source name
Default value	n/a
Valid values	Any valid DSN
Required	true

4.6.2.2.2 Attribute usr

Represents the user name that is used when connecting to the DBMS.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@usr]
`Format`	string
`Dimension`	n/a
`Default value`	n/a
`Valid values`	Any valid username
`Required`	true

4.6.2.2.3 Attribute pwd

Represents the password that is used when connecting to the DBMS.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@pwd]
`Format`	string
`Dimension`	n/a
`Default value`	n/a
`Valid values`	Any valid password
`Required`	true

4.6.2.2.4 Attribute name

The name of the namespace. If not specified, the namespace will be named “(nameless)”.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@name]
`Format`	string
`Dimension`	n/a
`Default value`	(nameless)
`Valid values`	Any valid string
`Required`	false

4.6.2.2.5 Attribute partition

This attribute specifies an expression that represents one or more DDS partitions. It is allowed to use wildcards in the expression: a '`*`' represents any sequence of characters and a '`?`' represents a single character.

This expression is used to specify the partitions from which DDS samples must be 'bridged' to the DBMS domain.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@partition]
`Format`	string
`Dimension`	n/a
`Default value`	`*`
`Valid values`	Any valid DDS partition expression
`Required`	false

4.6.2.2.6 Attribute topic

This attribute specifies an expression that represents one or more DDS topics. It is allowed to use wildcards in the expression: a '`*`' represents any sequence of characters and a '`?`' represents a single character.

This expression is used to specify the topics from which DDS samples must be *bridged* to the DBMS domain. For every matching topic encountered in one or more of the specified partitions, it creates a separate table in the DBMS. The table name will match that of the topic, unless specified otherwise in the *table* attribute of a *Mapping* element.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@topic]
`Format`	string
`Dimension`	n/a
`Default value`	`*`
`Valid values`	Any valid DDS Topic expression
`Required`	false

4.6.2.2.7 Attribute schema

This attribute represents the schema that is used when communicating with the DBMS. The exact schema content may be dependent on the DBMS that is being used, so consult your DBMS documentation for more details on this subject.

Full path	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@schema]
Format	string
Dimension	n/a
Default value	“” (empty string)
Valid values	Any valid string
Required	false

4.6.2.2.8 Attribute catalog

Represents the catalog that is used when communicating with the DBMS. The exact catalog content may be dependent on the DBMS that is being used, so consult your DBMS documentation for more details on this subject.

Full path	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@catalog]
Format	string
Dimension	n/a
Default value	“” (empty string)
Valid values	Any valid string
Required	false

4.6.2.2.9 Attribute replication_mode

This attribute specifies the replication mode for the current *NameSpace* element. If not specified, the value defaults to `false`.

When replicating databases through DDS, the *NameSpace* elements in the *DbmsToDds* and *DdsToDbms* elements map a Table and Topic circularly. To prevent data-modifications from continuously cascading, modifications made by the DBMSConnect service itself should not trigger new updates in the DBMS.

In replication mode, the DbmsConnect service ignores samples that were published by itself. (Currently this means that everything that is published on the same node as the DBMSConnect Service is considered to be of DBMSConnect origin and therefore ignored). That way, replication of changes that were copied from Dbms to DDS back into Dbms is avoided.

WARNING: This setting does not avoid replication of changes that were copied from DDS to Dbms back into DDS. For that purpose, the *replication_user* attribute of the *DbmsToDds* or *DbmsToDds/NameSpace* elements should be set appropriately as well!

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@replication_mode]
`Format`	boolean
`Dimension`	
`Default value`	false
`Valid values`	true, false
`Required`	false

4.6.2.2.10 Attribute update_frequency

This attribute specifies the frequency (in Hz) at which the service will automatically update the DBMS domain with DDS data. The default value of `0.0` means that updates are only performed when new DDS data arrives (*i.e.* updating is ‘event-based’).

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@update_factor]
`Format`	float
`Dimension`	frequency (Hz)
`Default value`	0.0
`Valid values`	0.0 - maxFloat
`Required`	false

4.6.2.2.11 Attribute odbc

The service dynamically loads an ODBC library at runtime. This attribute specifies the name of the ODBC library to be loaded. Platform specific pre- and postfixes and extensions are automatically added.

If this attribute is not provided, the service will attempt to load the generic ODBC library. The resulting behaviour is dependent on the platform on which the DbmsConnect Service is running.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@odbc]
`Format`	string
`Dimension`	Library name
`Default value`	Platform dependent
`Valid values`	Any valid library name
`Required`	false

4.6.2.2.12 Element *Mapping*

This element specifies a modification to the way that the service handles a pre-configured set of data within the specified NameSpace. Its attributes are used to configure the responsibilities of the service concerning the bridging of data from DDS to DBMS.

Full path	OpenSplice/DBMSConnectService/DdsToDbms/NameSpace/Mapping
Occurrences (min-max)	0 - *
Child-elements	<none>
Required attributes	*Attribute topic*
Optional attributes	*Attribute table* *Attribute query* *Attribute filter*

4.6.2.2.12.1 Attribute topic

This attribute specifies the name of the topic where the Mapping applies to. If you specify no particular topic, it will create tables for all topics.

Full path	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace/Mapping[@topic]
Format	string
Dimension	Topic name
Default value	*
Valid values	Any valid DDS Topic name
Required	true

4.6.2.2.12.2 Attribute table

This attribute specifies an alternative name for the table that must be associated with the Topic.

Full path	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace/Mapping[@table]
Format	string
Dimension	Table name

`Default value`	“” (empty string)
`Valid values`	Any valid DBMS table name
`Required`	false

4.6.2.2.12.3 Attribute query

This attribute specifies an SQL query expression. Only DDS data that matches the query will be bridged to the DBMS domain. This is realized by means of a DCPS query condition. The default value is an empty string, representing all available samples of the selected topic.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/ NameSpace/Mapping[@query]
`Format`	string
`Dimension`	DDS query expression
`Default value`	“” (empty string representing all data of the specified topic)
`Valid values`	WHERE clause of an SQL expression
`Required`	false

4.6.2.2.12.4 Attribute filter

This attribute specifies an SQL content filter. Only DDS data that matches the filter will be bridged to the DBMS domain. This is realized by means of a DCPS ContentFilteredTopic. The default value is an empty string, representing all available samples of the selected topic.

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/ NameSpace/Mapping[@filter]
`Format`	string
`Dimension`	DDS content filter
`Default value`	“” (empty string representing all data of the specified topic)
`Valid values`	WHERE clause of an SQL expression
`Required`	false

4.6.3 Element *DbmsToDds*

This element specifies the configuration properties concerning DDS to DBMS bridging.

Full path	OpenSplice/DBMSConnectService/DbmsToDds
Occurrences (min-max)	0 - 1
Child-elements	*Element NameSpace*
Required attributes	<none>
Optional attributes	*Attribute event_table_policy* *Attribute publish_initial_data* *Attribute replication_user* *Attribute trigger_policy*

4.6.3.1 Attribute event_table_policy

This attribute specifies the default setting of the event table policy for all *NameSpace* elements in the current *DbmsToDds* element.

An event table (sometimes referred to as ‘change table’ or ‘shadow table’) is a support-table that is slaved to an application-table, adding some status flags that are under the control of a trigger mechanism that responds to creation/modification/deletion events in the application-table.

The following policies are currently supported:

- **FULL**: (default) An ‘event table’ will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the table if it already exists.
- **LAZY**: An ‘event table’ will only be created if it is not available when the service connects, and it will not be deleted when the service disconnects.
- **NONE**: An ‘event table’ will neither be created nor deleted by the service. For each specified *NameSpace*, the service will poll for the existence of a consistent table with a frequency specified in the coresponding *update_frequency* attribute. It will start using the table as soon as it is available. With this policy set, no initial data will be published regardless of the value of the applicable *publish_initial_data* attribute.

Full path	OpenSplice/DbmsConnectService/DbmsToDds[@event_table_policy]
Format	enumeration
Dimension	n/a
Default value	FULL
Valid values	FULL, LAZY, NONE
Required	no

4.6.3.2 Attribute publish_initial_data

This attribute specifies the default behaviour with respect to publishing initially available data in the DBMS to the DDS for all *NameSpace* elements in the current *DbmsToDds* element. If not specified, it defaults to `true`. The value of this attribute is ignored when the corresponding *event_table_policy* is set to NONE.

`Full path`	OpenSplice/DbmsConnectService/ DbmsToDds[@publish_initial_data]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	no

4.6.3.3 Attribute replication_user

This attribute specifies the default replication user for all *NameSpace* elements in the current *DdsToDbms* element.

When replicating databases through DDS, the *NameSpace* elements in the *DbmsToDds* and *DdsToDbms* elements map a Table and Topic circularly. To prevent data-modifications from continuously cascading, modifications made by the service itself should not trigger new updates in the DBMS nor in the DDS.

To distinguish between DBMS updates coming from an application and DBMS updates coming from DDS, an extra database user (the replication user) has to be introduced that differs from the application users. That way, replication of changes that were copied from DDS to Dbms back into DDS is avoided. The *replication_user* attribute specifies the user name of that replication user. An empty string (default value) indicates that there is no separate replication user.

WARNING: This setting does not avoid replication of changes that were copied from Dbms to DDS back into Dbms. For that purpose, the *replication_mode* attribute of the *DssToDbms* or *DssToDbms/NameSpace* elements should be set appropriately as well!

`Full path`	OpenSplice/DbmsConnectService/ DbmsToDds[@replication_user]
`Format`	string
`Dimension`	n/a

Default value	“” (empty string indicating no replication user)
Valid values	Any valid SQL user name
Required	no

4.6.3.4 Attribute trigger_policy

This attribute specifies the default trigger policy for all *NameSpace* elements in the current *DbmsToDds* element.

Triggers are used to to update the event table in case of creation/modification/deletion events on the application-table.

The following policies are currently supported:

- **FULL**: (default) Triggers will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the triggers if they already exist.
- **LAZY**: Triggers will only be created if they are not available when the service connects, and will not be deleted when the service disconnects.
- **NONE**: Triggers will neither be created nor deleted by the service. This allows you to build your own custom triggering mechanism.

Full path	OpenSplice/DbmsConnectService/DbmsToDds[@trigger_policy]
Format	enumeration
Dimension	n/a
Default value	FULL
Valid values	FULL, LAZY, NONE
Required	no

4.6.3.5 Element *NameSpace*

This element specifies the responsibilities of the service concerning the bridging of data from DBMS to DDS. At least one *NameSpace* element has to be present in a *DbmsToDds* element.

Full path	OpenSplice/DBMSConnectService/DbmsToDds/NameSpace
Occurrences (min-max)	1 - *

`Child-elements`	*Element Mapping*
`Required attributes`	*Attribute dsn* *Attribute usr* *Attribute pwd*
`Optional attributes`	*Attribute name* *Attribute partition* *Attribute table* *Attribute schema* *Attribute catalog* *Attribute force_key_equality* *Attribute event_table_policy* *Attribute publish_initial_data* *Attribute replication_user* *Attribute trigger_policy* *Attribute update_frequency* *Attribute odbc*

4.6.3.5.1 Attribute dsn

Represents the ODBC Data Source Name, that represents the DBMS where the service must bridge the DDS data from.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@dsn]
`Format`	string
`Dimension`	Data source name
`Default value`	n/a
`Valid values`	Any valid DSN
`Required`	true

4.6.3.5.2 Attribute usr

Represents the user name that is used when connecting to the DBMS.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@usr]
`Format`	string
`Dimension`	Username

`Default value`	n/a
`Valid values`	Any valid username
`Required`	true

4.6.3.5.3 Attribute pwd

Represents the password that is used when connecting to the DBMS.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@pwd]
`Format`	string
`Dimension`	Password
`Default value`	n/a
`Valid values`	Any valid password
`Required`	true

4.6.3.5.4 Attribute name

The name of the namespace. If not specified, the namespace will be named "(nameless)".

`Full path`	OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@name]
`Format`	string
`Dimension`	n/a
`Default value`	(nameless)
`Valid values`	Any valid string
`Required`	false

4.6.3.5.5 Attribute partition

This attribute specifies an expression that represents one or more DDS partitions. It is allowed to use wildcards in the expression: a ‘`*`’ represents any sequence of characters and a ‘`?`’ represents a single character.

This expression is used to specify the DDS partition(s) where DBMS records will be written to as DDS samples by the service.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@partition]
`Format`	string
`Dimension`	n/a
`Default value`	*
`Valid values`	Any valid DDS partition expression
`Required`	false

4.6.3.5.6 Attribute table

This attribute specifies an expression that represents one or more DBMS tables. It is allowed to use wildcards in the expression: a '`*`' represents any sequence of characters and a '`?`' represents a single character.

This expression is used to specify the tables from which DBMS data must be 'bridged' to the DDS domain.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@table]
`Format`	string
`Dimension`	n/a
`Default value`	*
`Valid values`	Any Table expression
`Required`	false

4.6.3.5.7 Attribute schema

This attribute represents the schema that is used when communicating with the DBMS. The exact schema content may be dependent on the DBMS that is being used, so consult your DBMS documentation for more details on this subject.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@schema]
`Format`	string
`Dimension`	n/a
`Default value`	"" (empty string)
`Valid values`	Any valid string
`Required`	false

4.6.3.5.8 Attribute catalog

Represents the catalog that is used when communicating with the DBMS. The exact catalog content may be dependent on the DBMS that is being used, so consult your DBMS documentation for more details on this subject.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@catalog]
Format	string
Dimension	n/a
Default value	“” (empty string)
Valid values	Any valid string
Required	false

4.6.3.5.9 Attribute force_key_equality

This attribute specifies the default setting for *Mapping* elements in the current *NameSpace* element with regard to the enforcement of key equality between table and topic definitions. If true, key definitions from the table and topic must match, otherwise key definitions may differ.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@force_key_equality]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	no

4.6.3.5.10 Attribute event_table_policy

This attribute specifies the default setting of the event table policy for all *Mapping* elements in the current *NameSpace* element. If not specified, the value defaults to FULL.

An event table (sometimes referred to as ‘change table’ or ‘shadow table’) is a support-table that is slaved to an application-table, adding some status flags that are under the control of a trigger mechanism that responds to creation/modification/deletion events in the application-table.

The following policies are currently supported:

- **FULL**: An ‘event table’ will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the table if it already exists.
- **LAZY**: An ‘event table’ will only be created if it is not available when the service connects, and it will not be deleted when the service disconnects.
- **NONE**: An ‘event table’ will neither be created nor deleted by the service. For each specified *NameSpace*, the service will poll for the existence of a consistent table with a frequency specified in the coresponding *update_frequency* attribute. It will start using the table as soon as it is available. With this policy set, no initial data will be published regardless of the value of the applicable *publish_initial_data* attribute.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@event_table_policy]
`Format`	enumeration
`Dimension`	n/a
`Default value`	FULL
`Valid values`	FULL, LAZY, NONE
`Required`	no

4.6.3.5.11 Attribute publish_initial_data

This attribute specifies the default behaviour with respect to publishing initially available data in the DBMS to the DDS for all *Mapping* elements in the current *NameSpace* element. If not specified, the value defaults to `true`. The value of this attribute is ignored when the corresponding *event_table_policy* is set to NONE.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@publish_initial_data]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	no

4.6.3.5.12 Attribute replication_user

This attribute specifies the default replication user for all *Mapping* elements in the current *NameSpace* element. If not specified, the value defaults to an empty string.

When replicating databases through DDS, the *NameSpace* elements in the *DbmsToDds* and *DdsToDbms* elements map a Table and Topic circularly. To prevent data-modifications from continuously cascading, modifications made by the service itself should not trigger new updates in the DBMS nor in the DDS.

To distinguish between DBMS updates coming from an application and DBMS updates coming from DDS, an extra database user (the replication user) has to be introduced that differs from the application users. That way, replication of changes that were copied from DDS to Dbms back into DDS is avoided. The *replication_user* attribute specifies the user name of that replication user. An empty string (default value) indicates that there is no separate replication user.

WARNING: This setting does not avoid replication of changes that were copied from Dbms to DDS back into Dbms. For that purpose, the *replication_mode* attribute of the *DssToDbms* or *DssToDbms/NameSpace* elements should be set appropriately as well!

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace[@replication_user]
`Format`	string
`Dimension`	n/a
`Default value`	“” (empty string)
`Valid values`	Any valid SQL user name
`Required`	no

4.6.3.5.13 Attribute trigger_policy

This attribute specifies the default trigger policy for all *Mapping* elements in the current *NameSpace* element. If not specified, the value defaults to FULL.

Triggers are used to to update the event table in case of creation/modification/ deletion events on the application-table.

The following policies are currently supported:

- **FULL**: Triggers will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the triggers if they already exist.
- **LAZY**: Triggers will only be created if they are not available when the service connects, and will not be deleted when the service disconnects.
- **NONE**: Triggers will neither be created nor deleted by the service. This allows you to build your own custom triggering mechanism.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace[@trigger_policy]
Format	enumeration
Dimension	n/a
Default value	FULL
Valid values	FULL, LAZY, NONE
Required	no

4.6.3.5.14 Attribute update_frequency

This attribute specifies the frequency (in Hz) at which the service will update the DDS domain with DBMS data. The default value is 2.0Hz. Event-based updates are not supported. If 0.0Hz is specified, the default of 2.0Hz will be used.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace[@update_factor]
Format	float
Dimension	frequency (Hz)
Default value	2.0
Valid values	0.0 - maxFloat
Required	false

4.6.3.5.15 Attribute odbc

The service dynamically loads an ODBC library at runtime. This attribute specifies the name of the ODBC library to be loaded. Platform specific pre- and postfixes and extensions are automatically added.

If this attribute is not provided, the service will attempt to load the generic ODBC library. The resulting behaviour is dependent on the platform on which the DbmsConnect Service is running.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace[@odbc]
Format	string
Dimension	Library name

Default value	Platform dependent
Valid values	Any valid library name
Required	false

4.6.3.5.16 Element *Mapping*

This element specifies a modification to the way that the service handles a pre-configured set of data within the specified NameSpace. Its attributes are used to configure the responsibilities of the service concerning the bridging of data from DBMS to DDS.

Full path	OpenSplice/DBMSConnectService/DbmsToDds/ NameSpace/Mapping
Occurrences (min-max)	0 - *
Child-elements	<none>
Required attributes	*Attribute table*
Optional attributes	*Attribute topic* *Attribute query* *Attribute force_key_equality* *Attribute event_table_policy* *Attribute publish_initial_data* *Attribute trigger_policy*

4.6.3.5.16.1 Attribute table

This attribute specifies the name of the table where the Mapping applies to.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace/Mapping[@table]
Format	string
Dimension	Table name
Default value	n/a
Valid values	Any valid DBMS table name
Required	true

4.6.3.5.16.2 Attribute topic

This attribute specifies an alternative name for the topic that must be associated with the table. By default the topic name is equal to the table name.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@topic]
`Format`	string
`Dimension`	Topic name
`Default value`	The name of the matching table
`Valid values`	Any valid DDS Topic name
`Required`	false

4.6.3.5.16.3 Attribute query

Optional SQL query expression. Only DBMS data that matches the query will be bridged to the DDS domain. This is realized by means of a SQL query. The default value is an empty string, representing all available data in the selected table.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@query]
`Format`	string
`Dimension`	SQL expression
`Default value`	“” (empty string representing all data in the specified table)
`Valid values`	WHERE clause of an SQL expression
`Required`	false

4.6.3.5.16.4 Attribute force_key_equality

This attribute specifies the enforcement of key equality between table and topic definitions. If true, key definitions from the table and topic must match, otherwise key definitions may differ. If not specified, the value defaults to true.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@force_key_equality]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	no

4.6.3.5.16.5 Attribute event_table_policy

This attribute specifies the event table policy in the current *Mapping* element. If not specified, the value defaults to FULL.

An event table (sometimes referred to as ‘change table’ or ‘shadow table’) is a support-table that is slaved to an application-table, adding some status flags that are under the control of a trigger mechanism that responds to creation/modification/deletion events in the application-table.

The following policies are currently supported:

- **FULL**: An ‘event table’ will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the table if it already exists.
- **LAZY**: An ‘event table’ will only be created if it is not available when the service connects, and it will not be deleted when the service disconnects.
- **NONE**: An ‘event table’ will neither be created nor deleted by the service. For the specified table, the service will poll with a frequency specified in the coresponding *update_frequency* attribute of the parent *NameSpace*. It will start using the table as soon as it is available. With this policy set, no initial data will be published regardless of the value of the applicable *publish_initial_data* attribute.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@event_table_policy]
`Format`	enumeration
`Dimension`	n/a
`Default value`	FULL
`Valid values`	FULL, LAZY, NONE
`Required`	no

4.6.3.5.16.6 Attribute publish_initial_data

This attribute specifies the behaviour with respect to publishing the initially available data specified in the current *Mapping* element from DBMS to DDS. If not specified, the value defaults to `true`. The value of this attribute is ignored when the corresponding *event_table_policy* is set to NONE.

`Full path`	OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@publish_initial_data]
`Format`	boolean
`Dimension`	n/a

Default value	true
Valid values	true, false
Required	no

4.6.3.5.16.7 Attribute trigger_policy

This attribute specifies the trigger policy for the current *Mapping* element. If not specified, the value defaults to FULL.

Triggers are used to to update the event table in case of creation/modification/ deletion events on the application-table.

The following policies are currently supported:

- **FULL**: Triggers will always be created when the service connects, and will always be deleted when the service disconnects. In this mode, the service will replace the triggers if they already exist.
- **LAZY**: Triggers will only be created if they are not available when the service connects, and will not be deleted when the service disconnects.
- **NONE**: Triggers will neither be created nor deleted by the service. This allows you to build your own custom triggering mechanism.

Full path	OpenSplice/DbmsConnectService/DbmsToDds/ NameSpace/Mapping[@trigger_policy]
Format	enumeration
Dimension	n/a
Default value	FULL
Valid values	FULL, LAZY, NONE
Required	no

4.6.4 Element *Tracing*

This element controls all tracing aspects of the DbmsConnect Service.

Full path	OpenSplice/DbmsConnectService/Tracing
Occurrences (min-max)	0 - 1
Child-elements	*Element OutputFile* *Element Timestamps* *Element Verbosity*
Required attributes	<none>
Optional attributes	<none>

4.6.4.1 Element *OutputFile*

This element specifies where the tracing log is printed to. Note that "stdout" and "stderr" are considered legal values that represent "standard out" and "standard error" respectively. The default value is dbmsconnect.log.

Full path	OpenSplice/DbmsConnectService/Tracing/OutputFile
Format	string
Dimension	file name
Default value	dbmsconnect.log
Valid values	depends on operating system.
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.6.4.2 Element *Timestamps*

This element specifies whether the logging must contain timestamps.

Full path	OpenSplice/DbmsConnectService/Tracing/Timestamps
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	*Attribute Absolute*

4.6.4.2.1 Attribute Absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

Full path	OpenSplice/DbmsConnectService/Tracing/Timestamps[@absolute]
Format	boolean
Dimension	n/a
Default value	true
Valid values	true, false
Required	false

4.6.4.3 Element *Verbosity*

This element specifies the verbosity level of the logging.

Full path	OpenSplice/DbmsConnectService/Tracing/Verbosity
Format	enumeration
Dimension	n/a
Default value	INFO
Valid values	SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
Occurrences (min-max)	0 - 1
Child-elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.6.5 Element *Watchdog*

This element controls the characteristics of the Watchdog thread.

Full path	OpenSplice/DbmsConnectService/Watchdog
Occurrences (min-max)	0 - 1
Child-elements	*Element Scheduling*
Required attributes	<none>
Optional attributes	<none>

4.6.5.1 Element *Scheduling*

This element specifies the scheduling parameters used to control the watchdog thread.

`Full path`	OpenSplice/DbmsConnectService/Watchdog/Scheduling
`Occurrences (min-max)`	1 - 1
`Child-elements`	*Element Class* *Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.6.5.1.1 Element *Class*

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

`Full path`	OpenSplice/DbmsConnectService/Watchdog/Scheduling/Class
`Format`	enumeration
`Dimension`	n/a
`Default value`	Default
`Valid values`	Timeshare, Realtime, Default
`Occurrences (min-max)`	1 - 1
`Child Elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.6.5.1.2 Element *Priority*

This element specifies the thread priority that will be used by the watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/DbmsConnectService/Watchdog/Scheduling/Priority
Format	unsigned integer
Dimension	n/a
Default value	depends on operating system
Valid values	depends on operating system
Occurrences (min-max)	1 - 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7 The DDSI2 and DDSI2 Enhanced Networking Service

4.7.1 The DDSI2 Networking Service

4.7.1.1 Element DDSI2Service

The DDSI2 Networking configuration expects a root element named *OpenSplice/DDSI2Service*. Within this root element, the networking daemon will look for several child-elements.

Full path	OpenSplice/DDSI2Service
Occurrences (min-max)	0 – *
Child elements	*Element Threads* *Element Sizing* *Element Compatibility* *Element Discovery* *Element Tracing* *Element Internal* *Element Watchdog* *Element General* *Element TCP* *Element ThreadPool*
Required attributes	*Attribute name*
Optional attributes	<none>

The DDSI2 Networking Service is selected by using the following xml in your configuration file:

```
<Service name="ddsi2">
       <Command>ddsi2</Command>
</Service>
```

DDSI2 Formats and Units

The descriptions of the DDSI2 Networking Service settings often refer to one of three 'formats' for the way a value should be presented. These formats all consist of a number (either integer or floating-point) followed by a unit specification. The following formats are used:

- '***time***' is used to describe durations or intervals:

  - unit is '***s***' for seconds

  which may be prefixed with a multiplier:

  - '***n***' for 10^{-9}, '***u***' for 10^{-6}, '***m***' for 10^{-3}
  - '***min***' for 60, '***hr***' for 3600

  As an example, the participant discovery interval, Discovery/SPDPInterval (see Section *4.7.1.1.5.2* on page 306) takes a value in the *time* format To set it to `30s` (thirty seconds), all of the following entries are equivalent and will have the desired effect if one allows for a rounding error of a few nanoseconds in the fourth case:

  ```
  -30s
  -0.5min
  -30e9ns
  -8.333e-3hr
  ```

- '***memory size***' is used to describe buffer capacities:

  - unit is '***B***' for bytes

  which may be prefixed with a multiplier:

  - '***k***' or '***Ki***' for 2^{10}, '***M***' or '***Mi***' for 2^{20}, '***G***' or '***Gi***' for 2^{30}

  (Note that interpreting the SI prefixes as powers of 2 is traditional in memory size specifications.)

- '***bandwidth***' is used to describe network bandwidths:

  - unit '***bps***' or '***b/s***' for bits/second
  - unit '***Bps***' or '***B/s***' for bytes/second

  and in both cases may be prefixed with a multiplier:

  - '***k***' for 10^3, '***M***' for 10^6 or '***G***' for 10^9

- ‘***Ki***’ for 2^{10}, ‘***Mi***’ for 2^{20} or ‘***Gi***’ for 2^{30}.

(Note that interpreting the SI prefixes as powers of 10 is traditional in bandwidth specifications.)

A bandwidth may also be specified as ‘***inf***’ (no unit), to indicate that the DDSI2 Service should not attempt to artificially limit the bandwidth.

4.7.1.1.1 Attribute name

This attribute identifies the configuration for the DDSI2 Service. Multiple DDSI2 service configurations can be specified in one single resource. The actual applicable configuration is determined by the value of the name attribute, which must match the specified under the element `OpenSplice/Service[@name]` in the Domain Service configuration.

`Full path`	OpenSplice/DDSI2Service/name
`Format`	String
`Default value`	
`Required`	false

4.7.1.1.2 Element Threads

This element is used to set thread properties

`Full path`	OpenSplice/DDSI2Service/Threads
`Occurrences (min-max)`	0 – 1
`Child elements`	Element Thread
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.2.1 Element Thread

This element specifies thread properties, such as scheduling parameters.

`Full path`	OpenSplice/DDSI2Service/Threads/Thread
`Occurrences (min-max)`	0 – 1000

Child elements	*Element StackSize* *Element Scheduling*
Required attributes	<none>
Optional attributes	*Attribute name*

4.7.1.1.2.1.1 Attribute name

The name of the thread.

Full path	OpenSplice/DDSI2Service/Threads/Thread[@name]
Format	String
Default value	
Required	false

4.7.1.1.2.1.2 Element StackSize

This element specifies the stack size for this thread (or ‘default’).

Full path	OpenSplice/DDSI2Service/Threads/Thread/StackSize
Format	memory size
Default value	default
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.2.1.3 Element Scheduling

Configures the scheduling properties of the thread.

Full path	OpenSplice/DDSI2Service/Threads/Thread/Scheduling
Occurrences (min-max)	0 – 1
Child elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.2.1.3.1 Element Class

This element specifies the thread scheduling class (‘realtime’, ‘timeshare’ or ‘default’). The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/DDSI2Service/Threads/Thread/Scheduling/Class
Format	Enumeration
Default value	default
Valid values	realtime, timeshare, default
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.2.1.3.2 Element Priority

This element specifies the thread priority (decimal integer or ‘`default`’). Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/DDSI2Service/Threads/Thread/Scheduling/Priority
Format	String
Default value	default
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attribute	<none>
Optional attributes	<none>

4.7.1.1.3 Element Sizing

The Sizing element specifies a variety of configuration settings dealing with expected system sizes, buffer sizes, *etc.*.

Full path	OpenSplice/DDSI2Service/Sizing
Occurrences (min-max)	0 – 1
Child elements	*Element EndpointsInSystem* *Element NetworkQueueSize* *Element ReceiveBufferSize* *Element ReceiveBufferChunkSize* *Element LocalEndpoints*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.3.1 Element EndpointsInSystem

This endpoint specifies the expected maximum number of endpoints in the network. Underestimating this number will have a significant performance impact, but will not affect correctness; signficantly overestimating it will cause more memory to be used than necessary.

Full path	OpenSplice/DDSI2Service/Sizing/ EndpointsInSystem
Format	Integer
Default value	20000
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.3.2 Element NetworkQueueSize

This element specifies the maximum number of samples in the network queue. Write/dispose operations add samples to this queue, the DDSI2 service drains it. Larger values allow large bursts of writes to occur without forcing synchronization between the application and the DDSI2 service, but do introduce the potential for longer latencies and increase the maximum amount of memory potentially occupied by messages in the queue.

`Full path`	OpenSplice/DDSI2Service/Sizing/ NetworkQueueSize
`Format`	Integer
`Default value`	1000
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.3.3 Element ReceiveBufferSize

Size of a single receive buffer. Many receive buffers may be needed; this size must be greater than `ReceiveBufferChunkSize` (see Section *4.7.1.1.3.4* below) by a modest amount.

`Full path`	OpenSplice/DDSI2Service/Sizing/ ReceiveBufferSize
`Format`	memory size
`Default value`	1 MiB
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.3.4 Element ReceiveBufferChunkSize

Size of one allocation unit in the receive buffer. Must be greater than the maximum packet size by a modest amount (too large packets are dropped). These allocations shrink when possible.

Full path	OpenSplice/DDSI2Service/Sizing/ReceiveBufferChunkSize
Format	memory size
Default value	128 KiB
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.3.5 Element LocalEndpoints

This element specifies the expected maximum number of endpoints local to one DDSI2 service. Underestimating this number will have a significant performance impact, but will not affect correctness; significantly overestimating it will cause more memory to be used than necessary.

Full path	OpenSplice/DDSI2Service/Sizing/LocalEndpoints
Format	Integer
Default value	1000
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.4 Element Compatibility

The Compatibility elements allows specifying various settings related to compatibility with standards and with other DDSI implementations.

`Full path`	OpenSplice/DDSI2Service/Compatibility
`Occurrences (min-max)`	0 – 1
`Child elements`	*Element ArrivalOfDataAssertsPpAndEpLiveliness* *Element AckNackNumbitsEmptySet* *Element RespondToRtiInitZeroAckWithInvalidHeartbeat* *Element AssumeRtiHasPmdEndpoints* *Element StandardsConformance* *Element ExplicitlyPublishQosSetToDefault* *Element ManySocketsMode*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.1 Element ArrivalOfDataAssertsPpAndEpLiveliness

Ignored (for backwards compatibility).

`Full path`	OpenSplice/DDSI2Service/Compatibility/ ArrivalOfDataAssertsPpAndEpLiveliness
`Format`	Boolean
`Default value`	true
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.2 Element AckNackNumbitsEmptySet

This element governs the representation of an acknowledgement message that does not also negatively-acknowledge some samples.

If set to `0`, the generated acknowledgements have an invalid form and will be rejected by `strict` mode, but several other implementations require this setting for smooth interoperation.

If set to `1`, all acknowledgements sent by DDSI2 adhere to the form of acknowledgement messages allowed by the standard, but this causes problems when interoperating with these other implementations.

The `strict` and `pedantic` standards conformance modes always overrule `AckNackNumbitsEmptySet = 0` to prevent the transmission of invalid messages.

`Full path`	OpenSplice/DDSI2Service/Compatibility/AckNackNumbitsEmptySet
`Format`	Integer
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.3 Element RespondToRtiInitZeroAckWithInvalidHeartbeat

This option allows a closer mimicking of the behaviour of some other DDSI implementations, albeit at the cost of generating even more invalid messages. Setting it to `true` ensures a Heartbeat can be sent at any time when a remote node requests one, setting it to `false` delays it until a valid one can be sent. The latter is fully compliant with the specification, and no adverse effects have been observed. It is the default.

`Full path`	OpenSplice/DDSI2Service/Compatibility/RespondToRtiInitZeroAckWithInvalidHeartbeat
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.4 Element AssumeRtiHasPmdEndpoints

This option assumes `ParticipantMessageData` endpoints required by the liveliness protocol are present in RTI participants even when not properly advertised by the participant discovery protocol.

`Full path`	OpenSplice/DDSI2Service/Compatibility/ AssumeRtiHasPmdEndpoints
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.5 Element StandardsConformance

Selects the level of standards conformance of this instance of the DDSI2 Service. Stricter conformance typically means less interoperability with other implementations.

Currently three modes are defined:

`pedantic` – Very strictly conforms to the specification, ultimately for compliancy testing, but currently of little value because it adheres even to what will most likely turn out to be editing errors in the DDSI standard. Arguably, as long as no errata have been published it is the current text that is in effect, and that is what `pedantic` currently does.

`strict` – Takes a slightly less strict view of the standard than `pedantic`: it follows the established behaviour where the standard is obviously in error.

`lax` – Simply tries to provide the smoothest possible interoperability, anticipating future revisions of elements in the standard in areas that other implementations do not adhere to, even though there is no good reason not to.

The default setting is `lax`.

`Full path`	OpenSplice/DDSI2Service/Compatibility/StandardsConformance
`Format`	Enumeration
`Default value`	lax
`Valid values`	lax, strict, pedantic
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.6 Element ExplicitlyPublishQosSetToDefault

This option specifies whether QoS settings set to default values are explicitly published in the discovery protocol. Implementations are to use the default value for QoS settings not published, which allows a signficant reduction of the amount of data that needs to be exchanged for the discovery protocol, but this requires all implementations to adhere to the default values specified by the specifications. When interoperability is required with an implementation that does not follow the specifications in this regard, setting this option to `true` will help.

`Full path`	OpenSplice/DDSI2Service/Compatibility/ExplicitlyPublishQosSetToDefault
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.4.7 Element ManySocketsMode

This option specifies whether a network socket will be created for each domain participant on a host. The specification seems to assume that each participant has a unique address, and setting this option to `true` will ensure that this is the case.

The default setting of `false` slightly improves performance and reduces network traffic somewhat.

`Full path`	OpenSplice/DDSI2Service/Compatibility/ManySocketsMode
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.5 Element Discovery

The Discovery element allows specifying various parameters related to the discovery of peers.

`Full path`	OpenSplice/DDSI2Service/Discovery
`Occurrences (min-max)`	0 – 1
`Child elements`	*Element SPDPMulticastAddress* *Element SPDPInterval* *Element DomainId* *Element ParticipantIndex* *Element Ports* *Element Peers*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.5.1 Element SPDPMulticastAddress

This element specifies the multicast address that is used as the destination for the participant discovery packets.

`Full path`	OpenSplice/DDSI2Service/Discovery/ SPDPMulticastAddress
`Format`	String
`Default value`	239.255.0.1
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.5.2 Element SPDPInterval

This element specifies the interval between spontaneous transmissions of participant discovery packets.

`Full path`	OpenSplice/DDSI2Service/Discovery/ SPDPInterval
`Format`	time
`Default value`	10 s
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.5.3 Element DomainId

This element allows for the overriding of the DDS Domain Id that is used for this DDSI2 service.

`Full path`	OpenSplice/DDSI2Service/Discovery/DomainId
`Format`	String
`Default value`	default
`Occurrences (min-max`	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.4 Element ParticipantIndex

This element specifies the DDSI participant id used by this instance of the DDSI2 service for discovery purposes. Only one such participant id is used, independent of the number of actual DomainParticipants on the node. It is either ‘`AUTO`’ (the default), which will attempt to automatically determine an available participant index, or a non-negative integer less than 120, or ‘`NONE`’, which causes it to use arbitrary port numbers for unicast sockets which entirely removes the constraints on the participant index but makes unicast discovery impossible.

Full path	OpenSplice/DDSI2Service/Discovery/ParticipantIndex
Format	String
Default value	auto
Valid values	auto, none, 0 – 120
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5 Element Ports

The Port element allows specifying various parameters related to the port numbers used for discovery. These all have default values specified by the DDSI 2.1 specification.

Full path	OpenSplice/DDSI2Service/Discovery/Ports
Occurrences (min-max)	0 – 1
Child elements	*Element MulticastMetaOffset* *Element UnicastMetaOffset* *Element MulticastDataOffset* *Element UnicastDataOffset* *Element Base* *Element DomainGain* *Element ParticipantGain*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.1 Element MulticastMetaOffset

This element specifies the port number for multicast meta traffic (please refer to the DDSI 2.1 specification, section 9.6.1, constant d0).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/ MulticastMetaOffset
Format	Integer
Default value	0
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.2 Element UnicastMetaOffset

This element specifies the port number for unicast meta traffic (please refer to the DDSI 2.1 specification, section 9.6.1, constant d1).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/ UnicastMetaOffset
Format	Integer
Default value	10
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.3 Element MulticastDataOffset

This element specifies the port number for multicast meta traffic (please refer to the DDSI 2.1 specification, section 9.6.1, constant d2).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/ MulticastDataOffset
Format	Integer
Default value	1
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.4 Element UnicastDataOffset

This element specifies the port number for unicast meta traffic (please refer to the DDSI 2.1 specification, section 9.6.1, constant d3).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/ UnicastDataOffset
Format	Integer
Default value	11
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.5 Element Base

This element specifies the base port number (please refer to the DDSI 2.1 specification, section 9.6.1, constant PB).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/Base
Format	Integer
Default value	7400
Valid values	1 – 65535
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.6 Element DomainGain

This element specifies the domain gain, relating domain ids to sets of port numbers (please refer to the DDSI 2.1 specification, section 9.6.1, constant DG).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/DomainGain
Format	Integer
Default value	250
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.5.7 Element ParticipantGain

This element specifies the participant gain, relating participant index to sets of port numbers (please refer to the DDSI 2.1 specification, section 9.6.1, constant PG).

Full path	OpenSplice/DDSI2Service/Discovery/Ports/ParticipantGain
Format	Integer
Default value	2
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.6 Element Peers

This element statically configures addresses of discovery.

Full path	OpenSplice/DDSI2Service/Discovery/Peers
Occurrences (min-max)	0 – 1
Child elements	*Element Peer*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.5.6.1 Element Peer

This element statically configures addresses of discovery.

Full path	OpenSplice/DDSI2Service/Discovery/Peers/Peer
Occurrences (min-max)	0 – *
Child elements	<none>
Required attributes	<none>
Optional attributes	*Attribute address*

4.7.1.1.5.6.1.1 Attribute address

This element specifies an IP address to which discovery packets must be sent, in addition to the default multicast address if multicasting is enabled (please refer to General/AllowMulticast (see Section *4.7.1.1.9.2* on page 336)). Multiple Peers may be specified.

Full path	OpenSplice/DDSI2Service/Discovery/Peers/Peer/address
Format	String
Default value	
Required	false

4.7.1.1.6 Element Tracing

The Tracing element controls the amount and type of information that is written into the tracing log by the DDSI service. This is useful to track the DDSI service during application development.

Full path	OpenSplice/DDSI2Service/Tracing
Occurrences (min-max)	0 – 1
Child elements	*Element Timestamps* *Element AppendToFile* *Element PacketCaptureFile* *Element EnableCategory* *Element Verbosity* *Element OutputFile*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.6.1 Element Timestamps

This element specifies whether the logging must contain timestamps.

Full path	OpenSplice/DDSI2Service/Tracing/Timestamps
Format	Boolean
Default value	true
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	*Attribute absolute*

4.7.1.1.6.1.1 Attribute absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

Full path	OpenSplice/DDSI2Service/Tracing/Timestamps/absolute
Format	Boolean
Default value	true
Valid values	false, true
Required	false

4.7.1.1.6.2 Element AppendToFile

This option specifies whether the output is to be appended to an existing log file.

Full path	OpenSplice/DDSI2Service/Tracing/AppendToFile
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.6.3 Element PacketCaptureFile

This option specifies the file to which received and sent packets will be logged in the `pcap` format suitable for analysis using common networking tools, such as WireShark. IP and UDP headers are fictitious, in particular the destination address of received packets. The `TTL` may be used to distinguish between sent and received packets: it is `255` for sent packets and `128` for received ones.

Full path	OpenSplice/DDSI2Service/Tracing/PacketCaptureFile
Format	String
Default value	
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.6.4 Element EnableCategory

This element enables individual logging categories. These are enabled in addition to those enabled by Tracing/Verbosity.

Full path	OpenSplice/DDSI2Service/Tracing/ EnableCategory
Format	String
Default value	
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.6.5 Element Verbosity

This element enables standard groups of categories, based on a desired verbosity level. This is in addition to the categories enabled by the Tracing/EnableCategory setting.

Full path	OpenSplice/DDSI2Service/Tracing/Verbosity
Fomat	Enumeration
Default value	none
Valid values	config, info, warning, severe, none, finest, finer, fine
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.6.6 Element OutputFile

This option specifies where the logging is printed to. Note that `stdout` and `stderr` are treated as special values, representing 'standard out' and 'standard error' respectively. No file is created unless logging categories are enabled using the Tracing/Verbosity or Tracing/EnabledCategory settings.

`Full path`	OpenSplice/DDSI2Service/Tracing/OutputFile
`Format`	String
`Default value`	ddsi2.log
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7 Element Internal

The `Internal` elements deal with a variety of settings that are in no way supported and may change without notice.

Full path	OpenSplice/DDSI2Service/Internal
Occurrences (min-max)	0 – 1
Child elements	*Element ScheduleTimeRounding* *Element ResponsivenessTimeout* *Element PrimaryReorderMaxSamples* *Element RetransmitMerging* *Element SecondaryReorderMaxSamples* *Element DDSI2DirectMaxThreads* *Element RetransmitMergingPeriod* *Element DefragUnreliableMaxSamples* *Element SquashParticipants* *Element DefragReliableMaxSamples* *Element BuiltinEndpointSet* *Element AggressiveKeepLast1Whc* *Element ConservativeBuiltinReaderStartup* *Element MaxQueuedRexmitBytes* *Element MeasureHbToAckLatency* *Element LegacyFragmentation* *Element MaxQueuedRexmitMessages* *Element SuppressSpdpMulticast* *Element SpdpResponseMaxDelay* *Element WriterLingerDuration* *Element UnicastResponseToSpdpMessages* *Element MinimumSocketReceiveBufferSize* *Element SynchronousDeliveryPriorityThreshold* *Element SynchronousDeliveryLatencyBound* *Element MaxMessageSize* *Element NackDelay* *Element MaxParticipants* *Element FragmentSize* *Element PreEmptiveAckDelay* *Element AccelerateRexmitBlockSize* *Element DeliveryQueueMaxSamples* *Element MaxSampleSize* *Element Test* *Element Watermarks*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.1 Element ScheduleTimeRounding

INTERNAL

Allows for the timing of scheduled events to be rounded up so that more events can be handled in a single cycle of the event queue. The default is `0` so that events are not rounded at all, *i.e.* scheduled exactly, whereas a value of `10ms` (‘`10 ms`’ is also valid, as embedded white space is allowed for readability) would mean that events are rounded up to the nearest 10 milliseconds.

`Full path`	OpenSplice/DDSI2Service/Internal/ScheduleTimeRounding
`Format`	time
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.2 Element ResponsivenessTimeout

INTERNAL

Time an unresponsive reader is allowed to block the transmit thread until it is considered unresponsive and degraded to `unreliable`. It will be restored to reliable service once it resumes `ACK`’ing samples.

`Full path`	OpenSplice/DDSI2Service/Internal/ResponsivenessTimeout
`Format`	time
`Default value`	1 s
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.3 Element PrimaryReorderMaxSamples

INTERNAL

Maximum size in samples of a primary re-order admin (one per proxy-writer).

Full path	OpenSplice/DDSI2Service/Internal/PrimaryReorderMaxSamples
Format	Integer
Default value	64
Occurrences (min-max)	0 - 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.4 Element RetransmitMerging

INTERNAL

Setting controlling the addressing and timing of retransmits. Possible values are:

never – Retransmit only to the `NACK`-ing reader.

adaptive – Attempt to combine proper retransmits for reliability, but send historical transient-local data to the requesting reader only.

always – Do not distinguish between different causes, always try to merge.

The default is `adaptive`.

Full path	OpenSplice/DDSI2Service/Internal/RetransmitMerging
Format	Enumeration
Default value	adaptive
Valid values	never, adaptive, always
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.5 Element SecondaryReorderMaxSamples

INTERNAL

Maximum size in samples of a secondary re-order admin (one per out-of-sync reader).

`Full path`	OpenSplice/DDSI2Service/Internal/ SecondaryReorderMaxSamples
`Format`	Integer
`Default value`	16
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.6 Element DDSI2DirectMaxThreads

INTERNAL

Maximum number of extra threads for experimental direct mode.

`Full path`	OpenSplice/DDSI2Service/Internal/ DDSI2DirectMaxThreads
`Format`	Integer
`Default value`	1
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.7 Element RetransmitMergingPeriod

INTERNAL

Period during which future `NACK`s are considered to be covered by a retransmit.

`Full path`	OpenSplice/DDSI2Service/Internal/ RetransmitMergingPeriod
`Format`	time
`Default value`	5 ms

Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.8 Element DefragUnreliableMaxSamples

INTERNAL

Maximum number of samples being defragmented simultaneously for best-effort writers.

Full path	OpenSplice/DDSI2Service/Internal/ DefragUnreliableMaxSamples
Format	Integer
Default value	4
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.9 Element SquashParticipants

INTERNAL

Advertise only one domain participant in DDSI (the one corresponding to the DDSI2 process), making it the virtual owner of all readers and writers of all domain participants, reducing discovery work.

Full path	OpenSplice/DDSI2Service/Internal/ SquashParticipants
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.10 Element DefragReliableMaxSamples

INTERNAL

Maximum number of samples being defragmneted simultaneously for reliable writers.

Full path	OpenSplice/DDSI2Service/Internal/ DefragReliableMaxSamples
Format	Integer
Default value	16
Occurrences (min-max)	0 - 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.11 Element BuiltinEndpointSet

INTERNAL

Controls which participants will have which built-in endpoints for the discovery and liveliness protocols. Valid values are:

`full` – All participants have all endpoints.

`writers` – All participants have the writers, but just one has the readers.

`minimal` – Only one participant has built-in endpoints.

The default is `writers`, as this is thought to be compliant and reasonably efficient, `minimal` may or may not be compliant but is most efficient, and `full` is inefficient but certain to be compliant.

Full path	OpenSplice/DDSI2Service/Internal/ BuiltinEndpointSet
Full path	Enumeration
Default value	writers

Valid values	full, writers, minimal
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.12 Element AggressiveKeepLast1Whc

INTERNAL

Whether to drop a reliable sample from a `WHC` before all readers have `ACK`’d it as soon as a later sample becomes available.

Full path	OpenSplice/DDSI2Service/Internal/ AggressiveKeepLast1Whc
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.13 Element ConservativeBuiltinReaderStartup

INTERNAL

Let all built-in readers request all historical data, instead of just one.

Full path	OpenSplice/DDSI2Service/Internal/ ConservativeBuiltinReaderStartup
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1

`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.14 Element MaxQueuedRexmitBytes

INTERNAL

Maximum number of bytes queued for retransmission, default value of `0` is unlimited unless `AuxiliaryBandwidthLimit` is in effect, in which case it is `NackDelay * AuxiliaryBandwidthLimit`. Must be large enough to contain the largest sample that may need to be re-transmitted.

`Full path`	OpenSplice/DDSI2Service/Internal/ MaxQueuedRexmitBytes
`Format`	memory size
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.15 Element MeasureHbToAckLatency

INTERNAL

Measure heartbeat-to-ACK latency by prepending timestamps to heartbeats and `ACK`s and calculating round trip times.

`Full path`	OpenSplice/DDSI2Service/Internal/ MeasureHbToAckLatency
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.16 Element LegacyFragmentation

INTERNAL

⚠ This option enables a backwards-compatible, non-compliant setting and interpretation of the control flags in fragmented data messages. To be enabled *ONLY* when requiring interoperability between compliant and non-compliant versions of DDSI2 for large messages.

Full path	OpenSplice/DDSI2Service/Internal/ LegacyFragmentation
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.17 Element MaxQueuedRexmitMessages

INTERNAL

Maximum number of messages queued for retransmission, default value is arbitrarily chosen. Must be large enough to contain the largest sample that may need to be retransmitted.

Full path	OpenSplice/DDSI2Service/Internal/ MaxQueuedRexmitMessages
Format	Integer
Default value	200
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.18 Element SuppressSpdpMulticast

INTERNAL

Disable even the mandatory multicasting of the participant discovery packets.

Full path	OpenSplice/DDSI2Service/Internal/SuppressSpdpMulticast
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.19 Element SpdpResponseMaxDelay

INTERNAL

Maximum pseudo-random delay between discovering a remote participant and responding to it.

Full path	OpenSplice/DDSI2Service/Internal/SpdpResponseMaxDelay
Format	time
Default value	0
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.20 Element WriterLingerDuration

INTERNAL

Maximum duration for which actual deletion of a reliable writer with unacknowledged data in its history will be delayed to provide proper reliable transmission.

Full path	OpenSplice/DDSI2Service/Internal/ WriterLingerDuration
Format	time
Default value	1 s
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.21 Element UnicastResponseToSpdpMessages

INTERNAL

Repond to newly-discovered participants with a unicasted SPDP packet instead of rescheduling the periodic multicasted one.

Full path	OpenSplice/DDSI2Service/Internal/ UnicastResponseToSpdpMessages
Format	Boolean
Default value	true
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.22 Element MinimumSocketReceiveBufferSize

INTERNAL

Minimum size of socket receive buffers. This setting can only increase the size of the receive buffer, if the operating system by default creates a larger buffer, it is left unchanged.

Full path	OpenSplice/DDSI2Service/Internal/MinimumSocketReceiveBufferSize
Format	memory size
Default value	64 KiB
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.23 Element SynchronousDeliveryPriorityThreshold

INTERNAL

Messages with latency budget <= SynchronousDeliveryLatencyBound and transport priority >= SynchronousDeliveryPriorityThreshold will be delivered synchronously, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of bandwidth.

Full path	OpenSplice/DDSI2Service/Internal/SynchronousDeliveryPriorityThreshold
Format	Integer
Default value	2147483647
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.24 Element SynchronousDeliveryLatencyBound

INTERNAL

Messages with latency budget <= SynchronousDeliveryLatencyBound and transport priority >= SynchronousDeliveryPriorityThreshold will be delivered synchronously, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of bandwidth. The special value '`inf`' can be used to indicate that there is no limit on the latency budget.

`Full path`	OpenSplice/DDSI2Service/Internal/ SynchronousDeliveryLatencyBound
`Format`	time *or* '`inf`'
`Default value`	`0`
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.25 Element MaxMessageSize

INTERNAL

This element specifies the maximum size UDP payload DDSI will use. DDSI will try to maintain this limit, but will overrun it if it cannot otherwise fit the data in. For the current version, this means that when `MaxMessageSize < FragmentSize + 108` bytes (for reliable data, if there is only best-effort data, the delta is 92 bytes), UDP payloads larger than `MaxMessageSize` may be observed. On some networks it may be necessary to set this item to keep the packetsize below the MTU to prevent IP fragmentation.

`Full path`	OpenSplice/DDSI2Service/Internal/ MaxMessageSize
`Format`	memory size
`Default value`	4 KiB
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.26 Element NackDelay

INTERNAL

Delay between receipt of a `HEARTBEAT` indicating missing samples and a `NACK` (ignored when the `HEARTBEAT` requires an answer).

`Full path`	OpenSplice/DDSI2Service/Internal/ NackDelay
`Format`	time
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.27 Element MaxParticipants

INTERNAL

Maximum number of participants one DDSI2 service instance is willing to service. `0` is unlimited.

`Full path`	OpenSplice/DDSI2Service/Internal/ MaxParticipants
`Format`	Integer
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.28 Element FragmentSize

INTERNAL

Samples larger than `FragmentSize` are broken into fragments of `FragmentSize` bytes each, except the last one, which may be smaller. The DDSI specification mandates a minimum fragment size of 1025 bytes, but DDSI2 will do whatever size is requested, accepting fragments of which the size is at least the minimum of 1025 and `FragmentSize`.

`Full path`	OpenSplice/DDSI2Service/Internal/ FragmentSize
`Format`	memory size
`Default value`	1.25 KiB
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.29 Element PreEmptiveAckDelay

INTERNAL

The delay between the registration of an entity and the sending of the pre-emptive `ACKNACK`.

`Full path`	OpenSplice/DDSI2Service/Internal/ PreEmptiveAckDelay
`Format`	time
`Default value`	10 ms
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.30 Element AccelerateRexmitBlockSize

INTERNAL

Proxy readers that are assumed to sill be retrieving historical data get this many samples retransmitted when they `NACK` something, even if some of these samples have sequence numbers outside the set covered by the `NACK`.

`Full path`	OpenSplice/DDSI2Service/Internal/ AccelerateRexmitBlockSize
`Format`	Integer
`Default value`	0
`Occurrences (min-max)`	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.31 Element DeliveryQueueMaxSamples

INTERNAL

Maximum size of a delivery queue in samples for incoming samples to be accepted.

Full path	OpenSplice/DDSI2Service/Internal/DeliveryQueueMaxSamples
Format	Integer
Default value	256
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.32 Element MaxSampleSize

INTERNAL

Maximum allowed serialized size of a sample.

Full path	OpenSplice/DDSI2Service/Internal/MaxSampleSize
Format	Memory size
Default value	2147483647 B
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.33 Element Test

INTERNAL

Testing options.

Full path	OpenSplice/DDSI2Service/Internal/Test
Occurrences (min-max)	0 – 1
Child elements	*Element XmitLossiness*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.33.1 Element XmitLossiness

INTERNAL

Fraction of outgoing packets to drop, specified as ‘parts per thousand’.

Full path	OpenSplice/DDSI2Service/Internal/Test/ XmitLossiness
Format	Integer
Default value	0
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.34 Element Watermarks

INTERNAL

Watermarks for flow-control

Full path	OpenSplice/DDSI2Service/ Internal/Watermarks
Occurrences (min-max)	0 – 1
Child elements	*Element WhcLow* *Element WhcHigh*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.7.34.1 Element WhcLow

INTERNAL

Low watermark for the `WHC`s in samples, transmitting resumes when `WHC` shrinks to this size.

`Full path`	OpenSplice/DDSI2Service/Internal/Watermarks/WhcLow
`Format`	Integer
`Default value`	100
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.7.34.2 Element WhcHigh

INTERNAL

High watermark for the `WHC`s in samples, transmitting is suspended when the `WHC` grows to this size.

`Full path`	OpenSplice/DDSI2Service/Internal/Watermarks/WhcHigh
`Format`	Integer
`Default value`	400
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.8 Element Watchdog

This element specifies the type of OS scheduling class that will be used by the thread that announces its liveliness periodically.

Full path	OpenSplice/DDSI2Service/Watchdog
Occurrences (min-max)	0 – 1
Child elements	*Element Class* *Element Priority*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.8.1 Element Class

This element specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/DDSI2Service/Watchdog/Class
Format	Enumeration
Default value	default
Valid values	realtime, timeshare, default
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.8.2 Element Priority

This element specifies the thread priority. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

Full path	OpenSplice/DDSI2Service/Watchdog/Priority
Format	Integer
Default value	0
Occurrences (min-max)	0 – 1

PRISMTECH

Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9 Element General

The `General` element specifies overall DDSI2 service settings.

Full path	OpenSplice/DDSI2Service/General
Occurrences (min-max)	0 – 1
Child elements	*Element ExternalNetworkMask* *Element AllowMulticast* *Element MulticastTimeToLive* *Element DontRoute* *Element UseIPv6* *Element EnableMulticastLoopback* *Element CoexistWithNativeNetworking* *Element StartupModeDuration* *Element StartupModeCoversTransient* *Element NetworkInterfaceAddress* *Element MulticastRecvNetworkInterfaceAddresses* *Element ExternalNetworkAddress*
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9.1 Element ExternalNetworkMask

This element specifies the network mask of the external network address. This element is only used when the configured external network address is specified. In this case locators discovered within the same external subnet (according to this mask) will be translated to an internal address. For this purpose the network part of the external address (as indicated by the mask) will be replaced by the network address part of the selected primary interface.

Full path	OpenSplice/DDSI2Service/General/ ExternalNetworkMask
Format	String
Default value	0.0.0.0

`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.2 Element AllowMulticast

This element specifies whether the DDSI2 service may use IP multicasting.

When set to `false`, no multicast packets will ever be sent by the service, but it will still listen to multicast packets from other nodes. Generally speaking, the peer nodes will have to be listed as Discovery/Peer elements, but the service will automatically discover multicasting peers that advertise a unicast address.

When set to `true`, multicast is used preferentially. If any Peer elements are given, they will explicitly be included in the discovery by means of packets sent to the specified addresses.

`Full path`	OpenSplice/DDSI2Service/General/AllowMulticast
`Format`	Boolean
`Default value`	true
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.3 Element MulticastTimeToLive

This element specifies the time-to-live for outgoing multicast packets.

By specifying a value of ‘`0`’, multicast traffic can be confined to the local node, and such ‘loopback’ performance is typically optimized by the operating system

`Full path`	OpenSplice/DDSI2Service/General/MulticastTimeToLive
`Format`	Integer
`Default value`	32

Valid values	0 – 255
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9.4 Element DontRoute

This element specifies whether the DDSI2 service disables routing of IP packets.

Full path	OpenSplice/DDSI2Service/General/DontRoute
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9.5 Element UseIPv6

This element specifies whether the DDSI2 service will be using IPv6 instead of IPv4.

Full path	OpenSplice/DDSI2Service/General/UseIPv6
Format	Boolean
Default value	false
Valid values	false, true
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9.6 Element EnableMulticastLoopback

This element specifies whether the DDSI2 service will allow IP multicast packets within the node to be visible to all DDSI participants in the node, including itself. It must be `true` for intra-node multicast communications, but if a node runs only a single OpenSplice DDSI2 service and does not host any other DDSI-capable programs, it may be set to `false` for improved performance.

`Full path`	OpenSplice/DDSI2Service/General/ EnableMulticastLoopback
`Format`	Boolean
`Default value`	true
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.7 Element CoexistWithNativeNetworking

This element specifies whether the DDSI2 service can operate in conjunction with the OpenSplice Network Service. When `false`, the DDSI2 service will take care of all communications, including those between OpenSplice nodes; when `true`, the DDSI2 service will only communicate with DDS implementations other than OpenSplice . In this case the NetworkService must be configured as well

`Full path`	OpenSplice/DDSI2Service/General/ CoexistWithNativeNetworking
`Format`	Boolean
`Default value`	false
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.8 Element StartupModeDuration

This element specifies how long the DDSI2 remains in its ‘startup’ mode. While in ‘startup’ mode all volatile reliable data published on the local node is retained as if it were transient local data, so that existing readers on remote nodes can get the data even though discovering them takes some time. ‘Best-effort’ data by definition need not arrive, and transient and persistent data are covered by the Durability service.

Once the system is stable, the DDSI2 service keeps track of the existence of remote readers whether or not matching writers exist locally, avoiding this discovery delay, and thereby ensuring this is merely a node startup issue.

Setting `StartupModeDuration` to `0` will disable it.

`Full path`	OpenSplice/DDSI2Service/General/StartupModeDuration
`Format`	time
`Default value`	2 s
`Valid values`	0 – 60000
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.9 Element StartupModeCoversTransient

This element configures whether ‘startup’ mode should also cover transient and persistent data, for configurations where the Durability service does not take care of it. It is recommended that configurations without defined merge policies leave this enabled.

`Full path`	OpenSplice/DDSI2Service/General/StartupModeCoversTransient
`Format`	Boolean
`Default value`	true
`Valid values`	false, true
`Occurrences (min-max)`	0 – 1

`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.10 Element NetworkInterfaceAddress

This element specifies which network interface card should be used. Each DDSI2 service is bound to only one network interface card (NIC). The card can be identified by its corresponding IP address, network interface name, or the network portion of the address. If the value `auto` is entered here, the OpenSplice middleware will attempt to look up an interface that has the required capabilities.

`Full path`	OpenSplice/DDSI2Service/General/ NetworkInterfaceAddress
`Format`	String
`Default value`	auto
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.1.1.9.11 Element MulticastRecvNetworkInterfaceAddresses

This element specifies which network interface cards should be used to receive. Each DDSI2 service can receive from multiple network interface cards (NIC). The cards can be uniquely identified by their corresponding IP addresses.

- If ‘`preferred`’ is entered here, the OpenSplice middleware will attempt to receive multicasts only from the interface selected as the preferred interface, either automatically or using the General/NetworkInterfaceAddress setting (see Section *4.7.1.1.9.10* above).
- If the value `all` is entered here, the OpenSplice middleware will attempt to receive from all interfaces that have the required capabilities;
- If `any` is entered here, the OpenSplice middleware will attempt to receive from whichever interface the operating system uses as the default.
- If `none` is entered, it will not listen to multicasts on any interface.

Otherwise if a comma-separated list of network address(es) is supplied it will attempt to receive from the NICs corresponding to all of the listed addresses.

Full path	OpenSplice/DDSI2Service/General/MulticastRecvNetworkInterfaceAddress
Format	String
Default value	preferred
Valid Values	preferred, all, any, none, comma-separated list of IP addresses
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.9.12 Element ExternalNetworkAddress

This element specifies the network address this DDSI2 service advertises in the discovery protocol. This can, for example, be used to be a specify an externally visible address when Network Address Translation is used. By default, it is the network address of the selected primary interface (please refer to General/NetworkInterfaceAddress).

Full path	OpenSplice/DDSI2Service/General/ExternalNetworkAddress
Format	String
Default value	auto
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.10 Element TCP

The TCP element allows specifying various parameters related to running DDSI over TCP.

Full path	OpenSplice/DDSI2Service/TCP
Occurrences (min-max)	0 – 1

Child elements	<none>
Required attributes	<none>
Optional attributes	*Attribute Enable* *Attribute Locators* *Attribute NoDelay* *Attribute Port*

4.7.1.1.10.1 Attribute Enable

This element enables the optional TCP transport.

Full path	OpenSplice/DDSI2Service/TCP[@Enable]
Format	boolean
Dimension	n/a
Default value	false
Valid values	true, false
Required	no

4.7.1.1.10.2 Attribute Locators

This element specifies what endpoints should be placed in unicast locators (`local` or `none`). If set as `none`, no unicast locators will be advertised *via* DDSI discovery so peers will use the discovery connection for communication. The default value is `local` which means that the listener endpoint is advertised so peers will use this to establish a new connection back to the process.

Full path	OpenSplice/DDSI2Service/TCP[@Locators]
Format	enumeration
Dimension	n/a
Default value	local
Valid values	local, none
Required	no

4.7.1.1.10.3 Attribute NoDelay

This element enables the `TCP_NODELAY` socket option, preventing multiple DDSI messages being sent in the same TCP request.

Full path	OpenSplice/DDSI2Service/TCP[@NoDelay]
Format	boolean
Dimension	n/a
Default value	false
Valid values	true, false
Required	no

4.7.1.1.10.4 Attribute Port

This element specifies the port number used for DDSI discovery (the TCP listening port). Dynamically allocated if zero. Disabled if `-1` or not configured.

Full path	OpenSplice/DDSI2Service/TCP[@Port]
Format	signed integer
Dimension	n.a.
Default value	-1
Valid values	-1 – 65535
Occurrences (min-max)	0 – 1
Child Elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.1.1.11 Element ThreadPool

The `ThreadPool` element allows specifying various parameters related to configuring a thread pool for sending DDSI messages

Full path	OpenSplice/DDSI2Service/TCP
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	*Attribute Enable* *Attribute ThreadMax* *Attribute Threads*

4.7.1.1.11.1 Attribute Enable

This element enables the optional thread pool.

`Full path`	OpenSplice/DDSI2Service/ThreadPool[@Enable]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	true, false
`Required`	no

4.7.1.1.11.2 Attribute ThreadMax

This element configures the maximum number of threads in the thread pool.

`Full path`	OpenSplice/DDSI2Service/ThreadPool [@ThreadMax]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	8
`Valid values`	0 - maxInt
`Required`	no

4.7.1.1.11.3 Attribute Threads

This element configures the initial number of threads in the thread pool.

`Full path`	OpenSplice/DDSI2Service/ThreadPool[@Threads]
`Format`	unsigned integer
`Dimension`	n/a
`Default value`	4
`Valid values`	0 - maxInt
`Required`	no

4.7.2 The DDSI2 Enhanced Networking Service

The DDSI2 Enhanced Networking Service is an extended version of the DDSI2 Networking Service adding multiple channels to allow end-to-end prioritisation of traffic based on the ‘transport priority’ QoS for writers, bandwidth limiting on the transmitter side to control maximum network bandwidth usage by a single node, mapping DCPS topic/partition combinations to different multicast addresses to avoid saturating networks with high-volume data that needs to be received by only a few nodes, and data encryption. Note that these are all features that are also available in the RT networking service.

The DDSI2 Enhanced Networking Service is selected by using the following xml in your configuration file:

```
<Service name="ddsi2e">
    <Command>ddsi2e</Command>
</Service>
```

The essential difference from the DDSI2 Networking Service in the configuration is the `<Command>` element: ‘`ddsi2e`’ selects the enhanced service.

Because it is a strict extension, the full set of settings of the DDSI2 Networking Service is also available in the DDSI2 Enhanced Networking Service. Rather than listing them all again here, please refer to the preceding Section 4.7.1, *The DDSI2 Networking Service*, on page 293.

However, the root element for the configuration is different. For the DDSI2 Enhanced Networking Service, it is ‘`OpenSplice/DDSI2EService`’, and when considering configuration settings listed in the preceding section for the DDSI2 Enhanced Networking Service, this difference should be kept in mind.

The Channels, Security and Partitioning elements are exclusive to the DDSI2 Enhanced Networking Service and are described below. Furthermore some options have been added to subcategories, which are given below as well. For all other options, please see Section 4.7.1, *The DDSI2 Networking Service*, on page 293, bearing in mind the difference in root element names.

4.7.2.1 Element DDSI2EService

The DDSI2 Enhanced Networking configuration expects a root element named *`OpenSplice/DDSI2EService`*. Within this root element, the networking daemon will look for several child-elements.

The DDSI2EService element is a strict superset of the DDSI2Service element. This section documents *only* the additional settings; please refer to Section 4.7.1.1, *Element DDSI2Service*, on page 293, for information on all other options.

Full path	OpenSplice/DDSI2EService
Occurrences (min-max)	0 – *
Child elements	*Element Channels* *Element Security* *Element Partitioning* *Element Internal*
Required attributes	*Attribute name*
Optional attributes	<none>

DDSI2E Formats and Units

Please refer to *DDSI2 Formats and Units* on page 294 for descriptions of the formats to be used for presenting *time*, *memory size* and *bandwidth* values.

4.7.2.1.1 Attribute name

This attribute identifies the configuration for the DDSI2E Service. Multiple DDSI2E service configurations can be specified in one single resource. The actual applicable configuration is determined by the value of the name attribute, which must match the specified under the element `OpenSplice/Service[@name]` in the Domain Service configuration.

Full path	OpenSplice/DDSI2EService/name
Format	String
Default value	
Required	false

4.7.2.1.2 Element Channels

This element is used to group a set of Channels.

The set of channels define the behaviour of the network concerning aspects as priority and latency budget. By configuring a set of channels, the Networking service is able to function as a scheduler for the network bandwidth. It achieves this by using the application-defined DDS QoS policies of the data to select the most appropriate channel to send the data.

Full path	OpenSplice/DDSI2EService/Channels
Occurrences (min-max)	0 – 1
Child elements	*Element Channel*
Required attributes	<none>
Optional attributes	<none>

4.7.2.1.2.1 Element Channel

This element specifies all properties of an individual Channel.

The Networking service will make sure messages with a higher priority precede messages with a lower priority and it uses the latency budget to assemble multiple messages into one UDP packet where possible, to optimize the bandwidth usage. Of course, its performance depends heavily on the compatbility of the configured channels with the used DDS QoS policies of the applications.

Full path	OpenSplice/DDSI2EService/Channels/Channel
Occurrences (min-max)	0 – 42
Child elements	*Element DiffServField* *Element QueueSize* *Element DataBandwidthLimit* *Element AuxiliaryBandwidthLimit*
Required attributes	<none>
Optional attributes	*Attribute name* *Attribute transportPriority*

4.7.2.1.2.1.1 Attribute name

The name uniquely identifies the channel.

Full path	OpenSplice/DDSI2EService/Channels/Channel/name
Format	String
Default value	
Required	false

4.7.2.1.2.1.2 Attribute transportPriority

This attribute sets the transport priority of the channel.

Messages sent to the network have a `transport_priority` quality of service value. Selection of a networking channel is based on the priority requested by the message and the priority offered by the channel. The priority settings of the different channels divide the priority range into intervals. Within a channel, messages are sorted in order of priority.

`Full path`	OpenSplice/DDSI2EService/Channels/Channel/ transportPriority
`Format`	Integer
`Default value`	0
`Required`	false

4.7.2.1.2.1.3 Element DiffServField

This element specifies the DiffServ setting for messages sent *via* this channel

`Full path`	OpenSplice/DDSI2EService/Channels/Channel/ DiffServField
`Format`	Integer
`Default value`	0
`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.2.1.4 Element QueueSize

This element specifies the number of messages the networking queue can contain. Messages sent to the network are written into the networking queue. The networking service will read from this queue. If this queue is full, the writer writing into the queue is suspended and will retry until success. Note that a full networking queue is a symptom of an improperly designed system.

Full path	OpenSplice/DDSI2EService/Channels/Channel/ QueueSize
Format	Integer
Default value	0
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.2.1.2.1.5 Element DataBandwidthLimit

This element specifies the maximum network bandwidth used for data on this channel.

Full path	OpenSplice/DDSI2EService/Channels/Channel/ DataBandwidthLimit
Format	bandwidth
Default value	inf
Occurrences (min-max)	0 – 1
Child elements	<none>
Required attributes	<none>
Optional attributes	<none>

4.7.2.1.2.1.6 Element AuxiliaryBandwidthLimit

This element specifies the maximum used network bandwidth for auxiliary traffic this channel (*e.g.* resends, heartbeats, *etc.*).

Full path	OpenSplice/DDSI2EService/Channels/Channel/ AuxiliaryBandwidthLimit
Format	bandwidth
Default value	inf
Occurrences (min-max)	0 – 1

`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.3 Element Security

The Security element specifies DDSI2 security profiles that can be used to encrypt network partitions.

`Full path`	OpenSplice/DDSI2EService/Security
`Occurrences (min-max)`	0 – 1
`Child elements`	*Element SecurityProfile*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.3.1 Element SecurityProfile

Every `SecurityProfile` has a `name`, a cipher and `cipherKey`.

`Full path`	OpenSplice/DDSI2EService/Security/ SecurityProfile
`Occurrences (min-max)`	0 – *
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute name* *Attribute cipher* *Attribute cipherKey*

4.7.2.1.3.1.1 Attribute name

A DDSI2 `SecuirtyProfile` is uniquely identified by its name.

`Full path`	OpenSplice/DDSI2EService/Security/ SecurityProfile/name

Format	String
Default value	
Required	true

4.7.2.1.3.1.2 Attribute cipher

This is a mandatory attribute. Depending on the declared cipher, the cipher key must have a specific length, 128 bits, 192 bits, 256 bits, or none at all. The following case-insensitive values are supported by the current implementation:

aes128 – implements AES cipher with 128 bit cipher key (16 Bytes, 32 hexadecimal characters). This cipher will occupy 34 bytes of each UDP packet being sent.

aes192 – implements the AES cipher with 192 bit cipher-key (24 Bytes, 48 hexadecimal characters). This cipher will occupy 34 bytes of each UDP packet being sent.

aes256 – implements the AES cipher with 256 bit cipher-key (32 Bytes, 64 hexadecimal characters. This cipher will occupy 34 bytes of each UDP packet being sent.

blowfish – implements the Blowfish cipher with 128 bit cipher-key (16 Bytes, 32 hexadecimal characters). This cipher will occupy 26 bytes of each UDP packet being sent.

null – implements the NULL cipher. The only cipher that does not require a cipherkey. This cipher will occupy 4 bytes of each UDP packet being sent. All ciphers except for the NULL cipher are combined with SHA1 to achieve data integrity.

Full path	OpenSplice/DDSI2EService/Security/SecurityProfile/cipher
Format	String
Default value	0
Required	true

4.7.2.1.3.1.3 Attribute cipherKey

The cipherKey attribute is used to define the secret key required by the declared cipher. The value can be a URI referencing an external file containing the secret key, or the secret key can be defined in-place directly as a string value.

The key must be defined as a hexadecimal string, each character representing 4 bits of the key; for example, *1ABC* represents the 16 bit key *0001 1010 1011 1100*. The key must not follow a well-known pattern and must match exactly the key length required by the chosen cipher. In the case of malformed cipher-keys, the security profile in question will be marked as invalid. Moreover, each network partition referring to the invalid Security Profile will not be operational and thus traffic will be blocked to prevent information leaks.

As all OpenSplice applications require read access to the XML configuration file, for security reasons it is recommended to store the secret key in an external file in the file system, referenced by the URI in the configuration file. The file must be protected against read and write access from other users on the host. Verify that access rights are not given to any other user or group on the host. Alternatively, storing the secret key in-place in the XML configuration file will give read/write access to all DDS applications joining the same OpenSplice node. Because of this, the ‘in-place’ method is strongly discouraged.

Full path	OpenSplice/DDSI2EService/Security/SecurityProfile/cipherKey
Format	String
Default value	
Required	false

4.7.2.1.4 Element Partitioning

The `Partitioning` element specifies DDSI2 network partitions and their mapping.

Full path	OpenSplice/DDSI2EService/Partitioning
Occurrences (min-max)	0 – 1
Child elements	*Element NetworkPartitions* *Element IgnoredPartitions* *Element PartitionMappings*
Required attributes	<none>
Optional attributes	<none>

4.7.2.1.4.1 Element NetworkPartitions

The `NetworkPartitions` element specifies the DDSI2 network partitions.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions
`Occurrences (min-max)`	0 – *
`Child elements`	*Element NetworkPartition*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.4.1.1 Element NetworkPartition

Every `NetworkPartition` has a `name`, an `address` and a `connected` flag.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions/NetworkPartition
`Occurrences (min-max)`	0 – *
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute SecurityProfile* *Attribute name* *Attribute address* *Attribute connected*

4.7.2.1.4.1.1.1 Attribute SecurityProfile

The `SecurityProfile` optionally specifies the security profile as defined in the Security section, to be used for this NetworkPartition.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions/NetworkPartition/ SecurityProfile
`Format`	String
`Default value`	null
`Required`	false

4.7.2.1.4.1.1.2 Attribute name

A DDSI2 NetworkPartition is uniquely identified by its `name`.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions/NetworkPartition/name
`Format`	String
`Default value`	
`Required`	false

4.7.2.1.4.1.1.3 Attribute address

The `address` is a list of one or more multicast addresses. If more than one is specified, then the different addresses are separated by commas. Readers that are mapped to this partition will advertise these multicast addresses as locators, from which writers will choose the most appropriate.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions/NetworkPartition/address
`Format`	String
`Default value`	
`Required`	false

4.7.2.1.4.1.1.4 Attribute connected

⚠ The `connected` flag is not yet implemented; all configured Network partitions will connect when needed.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ NetworkPartitions/NetworkPartition/connected
`Format`	Boolean
`Default value`	true
`Valid values`	true, false
`Required`	false

4.7.2.1.4.2 Element IgnoredPartitions

The `IgnoredPartitions` element specifies the topic/partition combinations that are not distributed over the network.

Full path	OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions
Occurrences (min-max)	0 – *
Child elements	*Element IgnoredPartition*
Required attributes	<none>
Optional attributes	<none>

4.7.2.1.4.2.1 Element IgnoredPartition

This element can be used to create a ‘local partition’ that is only available on the node on which it is specified, and therefor won’t generate network load. Any DCPS partition/topic combination specified in this element will not be distributed by the Networking service.

Full path	OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions/IgnoredPartition
Occurrences (min-max)	0 – *
Child elements	<none>
Required attributes	<none>
Optional attributes	*Attribute DCPSPartitionTopic*

4.7.2.1.4.2.1.1 Attribute DCPSPartitionTopic

The service will match any DCPS messages to the `DCPSPartitionTopic` expression and determine whether it matches. The `PartitionExpression` and `TopicExpression` are allowed to contain a ‘`*`’ wildcard, meaning that anything matches. An exact match is considered better than a wildcard match. If a DCPS message matches an expression it will not be sent to the network.

Full path	OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions/IgnoredPartition/DCPSP
Format	String
Default value	
Required	false

4.7.2.1.4.3 Element PartitionMappings

The `PartitionMappings` element specifies the mapping from topic/partitions to DDSI2 network partitions.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ PartitionMappings
`Occurrences (min-max)`	0 – *
`Child elements`	*Element PartitionMapping*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.4.3.1 Element PartitionMapping

This element specifies a mapping between a network partition and a partition-topic combination. In order to give networking partitions a meaning in the context of DCPS, mappings from DCPS partitions and topics onto networking partitions should be defined. DDSI2 allows for a set of partition mappings to be defined.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ PartitionMappings/PartitionMapping
`Occurrences (min-max)`	0 – *
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute NetworkPartition* *Attribute DCPSPartitionTopic*

4.7.2.1.4.3.1.1 Attribute NetworkPartition

The `NetworkPartition` attribute of a partition mapping defines that networking partition that data in a specific DCPS partition of a specific DCPS topic should be sent to.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ PartitionMappings/PartitionMapping/net
`Format`	String
`Default value`	
`Required`	false

4.7.2.1.4.3.1.2 Attribute DCPSPartitionTopic

The service will match any DCPS messages to the `DCPSPartitionTopic` expression and determine whether it matches. The `PartitionExpression` and `TopicExpression` are allowed to contain a '`*`' wildcard, meaning that anything matches. An exact match is considered better than a wildcard match. For every DCPS message, the best matching partition is determined and the data is sent over the corresponding networking partition as specified by the matching `NetworkPartition` element.

`Full path`	OpenSplice/DDSI2EService/Partitioning/ PartitionMappings/PartitionMapping/DCPSP
`Format`	String
`Default value`	
`Required`	false

4.7.2.1.5 Element Internal

The `Internal` elements deal with a variety of settings that are in no way supported and may change without notice.

`Full path`	OpenSplice/DDSI2EService/Internal
`Occurrences (min-max)`	0 – 1
`Child elements`	*Element AuxiliaryBandwidthLimit*
`Required attributes`	<none>
`Optional attributes`	<none>

4.7.2.1.5.1 Element AuxiliaryBandwidthLimit

INTERNAL

This element specifies the maximum network bandwidth used for auxiliary data: `ACKNACK`, Heartbeat, resends and discovery.

`Full path`	OpenSplice/DDSI2EService/Internal/ AuxiliaryBandwidthLimit
`Format`	time
`Default value`	inf

`Occurrences (min-max)`	0 – 1
`Child elements`	<none>
`Required attributes`	<none>
`Optional attributes`	<none>

4.8 Record and Replay (RnR) Service

The Record and Replay Service enables you to record data from a DDS domain to a storage, and replay data from a storage back into the DDS domain.

`Full path`	OpenSplice/RnRService/
`Occurrences (min-max)`	0 – 0
`Child-elements`	*Element Watchdog*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.8.1 Attribute name

This attribute identifies a configuration for the Record and Replay Service by name.

Multiple service configurations can be specified in one single resource file. The actual applicable configuration is determined by the value of the *name* attribute, which must match the one specified by *OpenSplice/Domain/Service[@name]* in the configuration of the Domain Service.

`Full path`	OpenSplice/RnRService[@name]
`Format`	string
`Dimension`	n/a
`Default value`	rnr
`Valid values`	any string
`Required`	true

4.8.2 Element *Watchdog*

This element controls the characteristics of the Watchdog thread.

`Full path`	OpenSplice/RnRService/Watchdog
`Occurrences (min-max)`	0 – 1
`Child-elements`	*Element Scheduling*
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.2.1 Element *Scheduling*

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

`Full path`	OpenSplice/RnRService/Watchdog/Scheduling
`Occurrences (min-max)`	1 – 1
`Child-elements`	*Element Priority*
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.2.1.1 Element *Priority*

This element specifies the thread priority that will be used by the Watchdog thread. Only priorities that are supported by the underlying operating system can be assigned to this element. The user may need special privileges from the underlying operating system to be able to assign some of the privileged priorities.

`Full path`	OpenSplice/RnRService/Watchdog/Scheduling/ Priority
`Occurrences (min-max)`	0 – 1
`Child-elements`	<none>
`Required attributes`	<none>
`Optional attributes`	*Attribute priority_kind* *Attribute Class*

4.8.2.1.1.1 Attribute priority_kind

This attribute specifies whether the specified Priority is a relative or absolute priority.

Full path	OpenSplice/RnRService/Watchdog/Scheduling/Priority[@priority_kind]
Format	string
Dimension	n/a
Default value	Relative
Valid values	Relative, Absolute
Required	false

4.8.2.1.1.2 Attribute Class

This attribute specifies the thread scheduling class that will be used by the watchdog thread. The user may need the appropriate privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

Full path	OpenSplice/RnRService/Watchdog/Scheduling/Priority[@class]
Format	string
Dimension	n/a
Default value	Default
Valid values	Default, Timeshare, Realtime
Required	true

4.8.3 Element *Storage*

This element specifies a storage to use for recording and/or replaying data.

Currently the storage supports the types XML and CDR.

A storage of type XML will store the recorded data in XML format. A storage of type CDR will store the recorded data in binary CDR format.

Note that storages can also be created, or their properties modified, by Record and Replay configuration commands. These commands use the same syntax to specify configuration data as the OpenSplice configuration file, so the description given here also applies to configuration commands.

Full path	OpenSplice/RnRService/Storage
Occurrences (min-max)	0 – 0

`Child-elements`	*Element rr_storageAttrXML* *Element rr_storageAttrCDR*
`Required attributes`	*Attribute name*
`Optional attributes`	<none>

4.8.3.1 Attribute name

The name used to identify the storage in Record and Replay commands.

`Full path`	OpenSplice/RnRService/Storage[@name]
`Format`	string
`Dimension`	n/a
`Default value`	default
`Valid values`	any string
`Required`	true

4.8.3.2 Element rr_storageAttrXML

This element contains attributes describing an XML storage.

`Full path`	OpenSplice/RnRService/Storage/ rr_storageAttrXML
`Occurrences (min-max)`	0 – 1
`Child-elements`	*Element filename*
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.3.2.1 Element filename

The file used to store XML data. The filename may include a relative or absolute path. If a path is omitted, the storage file is created in the current working directory.

`Full path`	OpenSplice/RnRService/Storage/ rr_storageAttrXML/filename
`Format`	string
`Default value`	rnr-storage.dat
`Occurrences (min-max)`	0 – 1
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.3.3 Element rr_storageAttrCDR

This element contains the attributes describing a CDR storage.

`Full path`	OpenSplice/RnRService/Storage/ rr_storageAttrCDR
`Occurrences (min-max)`	0 – 1
`Child-elements`	*Element filename*
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.3.3.1 Element filename

The file used to store CDR data. The filename may include a relative or absolute path. If a path is omitted, the storage file is created in the current working directory.

`Full path`	OpenSplice/RnRService/Storage/ rr_storageAttrCDR/filename
`Format`	string
`Default value`	rnr-storage.dat
`Occurrences (min-max)`	0 – 1
`Required attributes`	<none>
`Optional attributes`	<none>

4.8.3.4 Element Statistics

Maintain and optionally publish statistics for this storage.

`Full path`	OpenSplice/RnRService/Storage/Statistics
`Occurrences (min-max)`	0 – 1
`Child-elements`	<none>
`Required attributes`	*Attribute enabled* *Attribute publish_interval*
`Optional attributes`	*Attribute reset*

4.8.3.4.1 Attribute enabled

This attribute specifies whether statistics should be maintained for this storage.

`Full path`	OpenSplice/RnRService/Storage/Statistics [@enabled]
`Format`	boolean
`Dimension`	n/a
`Default value`	true
`Valid values`	true, false
`Required`	true

4.8.3.4.2 Attribute publish_interval

This attribute specifies the publication interval of the statistics belonging to this storage, in a Record and Replay storage statistics topic. The publish interval is a value in seconds but may also be set to `-1`, which means that the statistics are published when the storage is closed. Note that a value of `0` means that statistics are never published for this storage.

`Full path`	OpenSplice/RnRService/Storage/Statistics [@publish_interval]
`Format`	Integer
`Dimension`	n/a
`Default value`	30
`Valid values`	-1, 0, or any positive integer
`Required`	true

4.8.3.4.3 Attribute reset

This attribute enables you to reset the current values of statistics belonging to the storage. Note that this only makes sense in a configuration command for an existing storage, since new storages created from the OpenSplice configuration file always start out with empty statistics.

`Full path`	OpenSplice/RnRService/Storage/Statistics [@reset]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	false, true
`Required`	false

4.8.4 Element *Tracing*

This element enables the output of debugging information from the RnR service to a log file.

`Full path`	OpenSplice/RnRService/Tracing
`Occurrences (min-max)`	0 – 1
`Child-elements`	<none>
`Required attributes`	*Attribute OutputFile*
`Optional attributes`	*Attribute AppendToFile* *Attribute Verbosity* *Attribute EnableCategory*

4.8.4.1 Attribute OutputFile

This option specifies where the log is printed to. Note that ‘`stdout`’ is considered a legal value that represents ‘standard out’ and ‘`stderr`’ is a legal value representing ‘standard error’.

`Full path`	OpenSplice/RnRService/Tracing[@OutputFile]
`Format`	string
`Dimension`	file name
`Default value`	rnr.log
`Valid values`	depends on operating system
`Required`	true

4.8.4.2 Attribute AppendToFile

This option specifies whether the output is to be appended to an existing log file. The default is to overwrite the log file if it exists.

`Full path`	OpenSplice/RnRService/Tracing[@AppendToFile]
`Format`	boolean
`Dimension`	n/a
`Default value`	false
`Valid values`	false, true
`Required`	false

4.8.4.3 Attribute Verbosity

This option specifies the verbosity level of the logging information. The higher the level, the more (detailed) information will be logged.

`Full path`	OpenSplice/RnRService/Tracing[@Verbosity]
`Format`	enumeration
`Dimension`	n/a
`Default value`	INFO
`Valid values`	SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, NONE
`Required`	false

4.8.4.4 Attribute EnableCategory

This option enables logging levels for categories independently of the categories selected by specifiying a verbosity level.

Multiple categories, seperated by a comma, can be supplied.

The following categories are available:

FATAL: Errors that are potentially fatal for the correct operation of the service.

ERROR: Non-fatal errors.

WARNING: Warnings that indicate for example incorrect or unsupported usage of the service.

INFO: Descriptive messages, logged when certain important events occur.

CONFIG: Events related to the service configuration.

TRACE: Detailed messages describing the behaviour of the service.

RECORD: Messages for each recorded sample.

REPLAY: Messages for each replayed sample.

`Full path`	OpenSplice/RnRService/Tracing [@EnableCategory]
`Format`	enumeration
`Dimension`	n/a
`Default value`	<none>
`Valid values`	FATAL, ERROR, WARNING, INFO, CONFIG, TRACE, RECORD, REPLAY
`Required`	false

4.9 Example Reference Systems

The OpenSplice middleware can be deployed for different kinds of systems. This section identifies several different systems that will be used as reference systems throughout the rest of this manual. Each needs to be configured differently in order to fit its requirements. The intention of this section is to give the reader an impression of the possible differences in system requirements and the configuration aspects induced.

4.9.1 Zero Configuration System

The OpenSplice middleware comes with a default configuration file that is intended to give a satisfactory out-of-the-box experience. It suits the standard situation of a system containing a handful of nodes with only a few applications per node (enabling standalone 'single-process' deployment) and where requirements on data distribution latencies, volumes and determinism are not too demanding (enabling use of the standard DDSI networking service).

Starting and running any systems that satisfy these conditions should not be a problem. Nodes can be started and shutdown without any extra configuration because the default discovery mechanism will keep track of the networking topology.

4.9.2 Single Node System

Systems that have to run as a federation on a single node can be down-scaled considerably by not starting the networking and durability daemons. The networking daemon is obviously not needed because its responsibility is forwarding data to and from the network, which is not present. The durability daemon is not needed because the OpenSplice libraries themselves are capable of handling durable data on a single node.

Note that this is not the case for single process deployments. Multiple single process applications that are running on the same machine node can only communicate when there is a networking service running within each process. This is because there is no shared administration between the applications, unlike for the shared memory deployments when a networking service is not required for a single node system.

With a single node system, the OpenSplice services do not have much influence on application behaviour. The application has full control over its own thread priorities and all OpenSplice activities will be executed in the scope of the application threads.

One exception to this is the listener thread. This thread is responsible for calling listener functions as described in the DDS specification.

4.9.3 Medium Size Static (Near) Real-time System

Many medium size systems have highly demanding requirements with respect to data distribution latencies, volumes and predictability. Such systems require configuration and tuning at many levels. The OpenSplice middleware will be an important player in the system and therefore is highly configurable in order to meet these requirements. Every section reflects on an aspect of the configuration.

4.9.3.1 High Volumes

The OpenSplice middleware architecture is designed for efficiently transporting many small messages. The networking service is capable of packing messages from different writing applications into one network packet. For this, the latency budget quality of service should be applied. A latency budget allows the middleware to optimise on throughput. Messages will be collected and combined during an interval allowed by the latency budget. This concerns networking traffic only.

A network channel that has to support high volumes should be configured to do so. By default, the resolution parameter is set to 50 ms. This means that latency budget values will be truncated to multiples of 50 ms, which is a suitable value. For efficient packing, the FragmentSize should be set to a large value, for example 8000. This means that data will be sent to the network in chunks of 8 kilobytes. A good value for MaxBurstSize depends on the speed of the attached network and on the networking load. If several writers start writing simultaneously at full speed during a longer period, receiving nodes could start losing packets. Therefore, the writers might need to be slowed down to a suitable speed.

Note that message with a large latency budget might be overtaken by messages with a smaller latency budget, especially if they are transported *via* different networking channels.

4.9.3.2 Low Latencies

If messages are to be transported with requirements on their end to end times, a zero latency budget quality of service should be attached. This results in an immediate wake-up of the networking service at the moment that the message arrives in a networking queue. For optimal results with respect to end-to-end latencies, the thread priority of the corresponding networkChannel should be higher than the thread priority of the writing application. with the current implementation, a context switch between the writing application and the networking channel is always required. With the correct priorities, the induced latency is minimized.

The value of the Resolution parameter has its consequences for using latency budgets. The networking service will ignore any latency budgets that have a value smaller than Resolution.

The effect of sending many messages with a zero latency budget is an increase of CPU load. The increasing number of context switches require extra processing. This quality of service should therefore be consciously used.

4.9.3.3 Responsiveness

Especially with respect to reliable transport over the network, responsiveness is an important aspect. Whenever a reliably sent message is lost on the network, the sending node has to initiate a resend. Since OpenSplice networking uses an acknowledgement protocol, it is the up to the sending side to decide when to resend a message. This behaviour can be tuned.

First of all, the Resolution parameter is important. This parameter gives the interval at which is checked if any messages have to be resent. The RecoveryFactor parameter indicates how many of these checks have to be executed before actually resending a message. If Resolution is scaled down, messages will be resent earlier. If Recovery factor is scaled down, messages will be resent earlier as well.

4.9.3.4 Topology Discovery

OpenSplice RT-Networking Service implements a discovery protocol for discovering other nodes in the system. As long as only one node is present, nothing has to be sent to the network. As soon as at least two nodes are present, networking starts sending data to the network. The node-topology detection also allows for quick reaction to topology changes, such as when a publishing node disappears (due to a disconnection or a node crash); a ‘tightly-configured’ discovery allows for swift detection of such topology changes, and related updating of DDS-level liveliness administration.