ADLINK

Deployment Guide

Release 6.x

Contents

1

Preface

1.1 About the Deployment Guide

The *Vortex OpenSplice 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 Vortex OpenSplice Documentation Set.

The *Deployment Guide* is intended to be used after reading and following the instructions in the *Vortex OpenSplice Getting Started Guide*.

1.2 Intended Audience

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

1.3 Organisation

The *Overview* gives a general description of the Vortex OpenSplice architecture.

This is followed by *Service Descriptions*, which explain how Vortex OpenSplice provides integration of real-time DDS and the non-/near-real-time enterprise DBMS domains.

The *Tools* section introduces the OpenSplice system management tools.

Full details of the configuration elements and attributes of all Vortex OpenSplice services are given in the *Configuration* section.

1.4 Conventions

The icons shown below are used in ADLINK product documentation to help readers to quickly identify information relevant to their specific use of Vortex OpenSplice.

Icon	*Meaning*
	Item of special significance or where caution needs to be taken.
	Item contains helpful hint or special information.
	Information applies to Windows (*e.g.* XP, 2003, Windows 7) only.
	Information applies to Unix-based systems (*e.g.* Solaris) only.
	Information applies to Linux-based systems (*e.g.* Ubuntu) only.
	C language specific.
	C++ language specific.
	C# language specific.
	Java language specific.

2

Overview

This chapter explains the Vortex OpenSplice 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.

2.1 Vortex OpenSplice Architecture

Vortex OpenSplice is highly configurable, even allowing the architectural structure of the DDS middleware to be chosen by the user at deployment time.

Vortex OpenSplice 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, Vortex OpenSplice 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 Vortex 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 Vortex OpenSplice 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.

2.1.1 Single Process architecture

This deployment allows the DDS applications and Vortex 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.

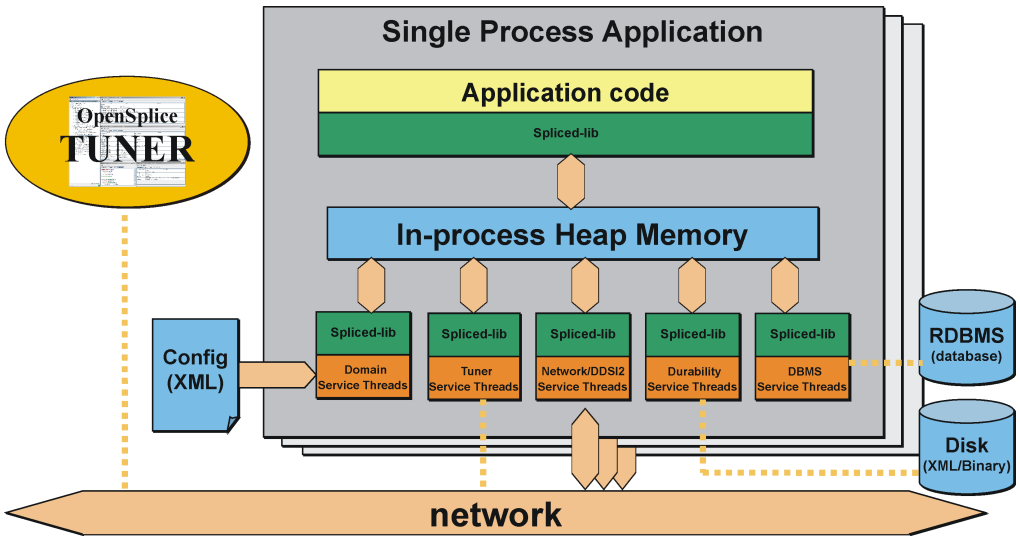
Each DDS application on a processing node is implemented as an individual, self-contained standalone operating system process (*i.e.* all of the DDS administration and necessary services have been linked into the application process). This is known as a **single process** application. Communication between 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 Vortex OpenSplice and allows for easy deployment with minimal configuration required for a running DDS system.

The diagram The Vortex OpenSplice Single Process Architecture shows an overview of the single process architecture of Vortex OpenSplice.

The Vortex OpenSplice Single Process Architecture



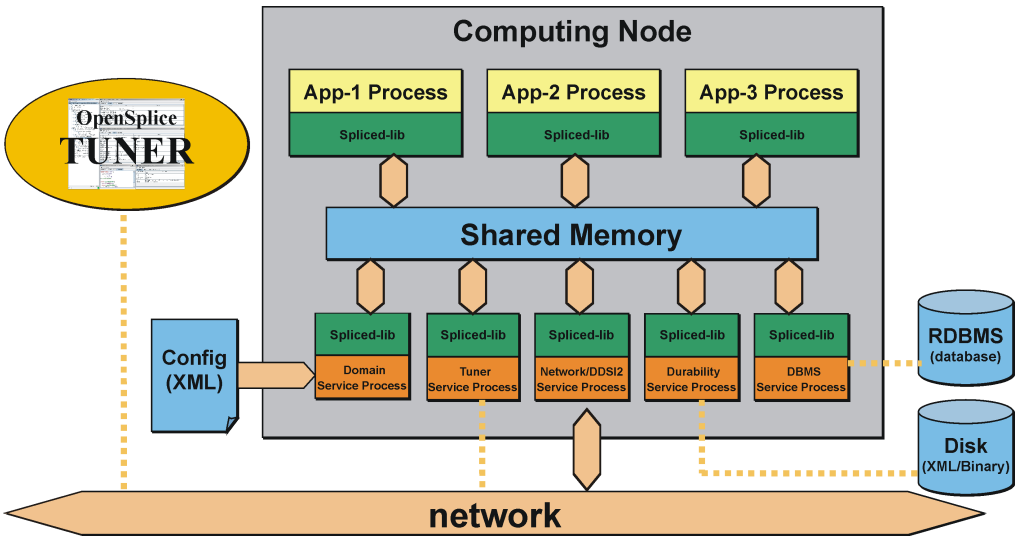
2.1.2 Shared Memory architecture

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 Vortex OpenSplice administration interface directly with the shared memory which is created by the Vortex 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 Vortex 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.

The diagram The Vortex OpenSplice Shared Memory Architecture shows an overview of the shared memory architecture of Vortex OpenSplice on *one* computing node. Typically, there are *many* nodes within a system.

The Vortex OpenSplice Shared Memory Architecture



2.1.3 Comparison of Deployment Architectures

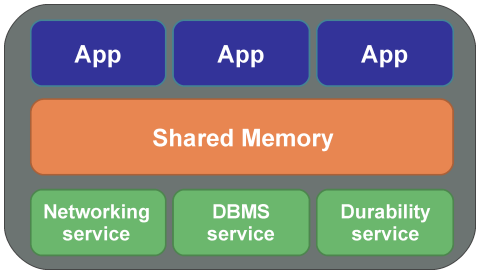
Simple when sufficient, Performant when required

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)

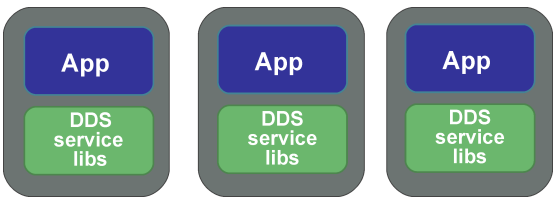
Federated Application Cluster



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)

Non-federated, single-process Applications



2.1.4 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.

NOTE for VxWorks kernel mode builds of OpenSplice the single process feature of the OpenSplice domain must not be enabled. i.e. "<SingleProcess>true</SingleProcess>" must not be included in the OpenSplice Configuration xml. The model used on VxWorks kernel builds is always that an area of kernel memory is allocated to store the domain database (the size of which is controlled by the size option in the Database configuration for OpenSplice as is used on other platforms for the shared memory model.) This can then be accessed by any task on the same VxWorks node.

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' Vortex 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 Vortex OpenSplice 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_`.

2.2 Vortex OpenSplice Usage

The Vortex 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.

Unix **Linux**

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 Vortex OpenSplice installation:

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

Windows

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

2.2.1 Starting Vortex OpenSplice for a Single Process Deployment

For 'standalone' single process deployment, there is no need to start the Vortex OpenSplice middleware before starting the DDS application, since the application itself will implicitly start the library threads of Vortex Open-

Splice Domain Service and associated services at the point when the DDS `create_participant` operation is invoked by the standalone application process.

2.2.2 Starting Vortex OpenSplice for a Shared Memory Deployment

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

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

This will start the Domain Service using the default configuration.

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

2.2.3 Monitoring

The Vortex 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.

2.2.3.1 Diagnostic Messages

Vortex 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.

Vortex 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 the *Configuration* section for details).

Values specified in the domain configuration override the environment values.

2.2.3.2 Vortex OpenSplice Tuner

The intention of Vortex OpenSplice Tuner, `ospltun`, is to provide facilities for monitoring and controlling Vortex OpenSplice, as well as the applications that use OpenSplice for the distribution of data. The *Vortex OpenSplice Tuner User Guide* specifies the capabilities of Vortex 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 run on a node that is already running Vortex OpenSplice using the shared memory deployment mode.

The Tuner will also be able to monitor and control a Domain running as a single process if the Tuner itself is started as the single process application with other DDS applications clustered in the process by being deployed as a single process application cluster. Please refer to the *Vortex OpenSplice Tuner User Guide* for a description of how to cluster applications together in a single process.

2.2.3.3 Vortex OpenSplice Memory Management Statistics Monitor

The Vortex 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 *the Tools chapter* 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.

2.2.4 Stopping Vortex OpenSplice

2.2.4.1 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 Vortex OpenSplice middleware running within the process will be stopped and the process will terminate.

2.2.4.2 Stopping a Shared Memory deployment

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

```
% ospl stop
```

The Vortex 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.

Stopping OSPL by using signals

Alternatively the Vortex OpenSplice domain service can also be stopped by sending a signal to the `ospl` process, assuming the process was started using the `-f` option.

Unix

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

Stopping Applications in Shared Memory Mode

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

When Vortex OpenSplice applications terminate naturally, a cleanup mechanism is executed that releases any references held to the shared memory within Vortex 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 Vortex OpenSplice will become inconsistent and no recovery will then be possible other than re-starting Vortex OpenSplice and all applications on the node.

2.2.5 Deploying Vortex OpenSplice on VxWorks 6.x

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

2.2.6 Deploying Vortex OpenSplice on Integrity

The **Integrity** version of Vortex OpenSplice 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 detailed information about Vortex OpenSplice deployment on Integrity.

2.2.7 Installing/Uninstalling the Vortex OpenSplice C# Assembly to the Global Assembly Cache

The installer for the commercial distribution of Vortex OpenSplice 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.

2.2.7.1 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 <Vortex OpenSplice installation path>\bin\dcpssacsAssembly.dll
```

where `<Vortex OpenSplice installation path>` is the installation path of the Vortex OpenSplice 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 \ADLINK\VortexOpenSplice\V6.6.0\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>
```

2.2.7.2 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 `<Vortex OpenSplice 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 Vortex OpenSplice 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 Vortex OpenSplice C# Assembly you specify the version you want to uninstall.

2.3 Vortex OpenSplice Configuration

Each application domain has its own characteristics; Vortex OpenSplice therefore 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 Vortex 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.

2.3.1 Configuration Files

Vortex 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 Vortex OpenSplice, a reference to a configuration is expressed in the form of a Uniform Resource Identifier (URI). Currently, only file URIs are supported (for example, `file:///opt/ospl/config/ospl.xml`).

When Vortex 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.

2.3.2 Environment Variables

The Vortex 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 Vortex 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.

2.3.2.1 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 Vortex OpenSplice but of course can be overridden to refer to a customer configuration.

For single process mode operation this variable is required; see also Single Process architecture in this *Guide*, and the detailed description of the Element `//OpenSplice/Domain/SingleProcess` in the *Configuration* section.

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

2.3.3 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 (`//OpenSplice/Domain/SingleProcess`). See the *Configuration* section for full details. In such a deployment, each Vortex OpenSplice 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 Vortex OpenSplice 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 Vortex OpenSplice system, there is no requirement to pass any Vortex OpenSplice configuration parameters to the application.

2.3.4 Configuration of Shared Memory deployment

In order to have Vortex 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 Vortex OpenSplice instance, the same mechanism holds. Use:

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

Several instances of Vortex 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 Vortex OpenSplice instances by using the command:

```
% ospl list
```

To stop all active Vortex 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.

2.3.5 Temporary Files

Please note that for a *shared memory* deployment, Vortex 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 locations 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.

2.4 Applications which operate in multiple domains

Vortex 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 Vortex 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.

Vortex 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 Vortex OpenSplice daemon (and any services) for that domain and the application itself. This requires some thought when configuring multiple Vortex 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 (please also see the description of the XML element `//OpenSplice/Domain/Database/Address` in the *Configuration* section).

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.

2.4.1 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.

2.5 Time-jumps

Observed time discontinuities can affect data ordering and processing of middleware actions. Time-jumps can be caused by adjusting the clock forward or backward. When resuming from being suspended, time will seem to have jumped forward as if the clock was advanced.

2.5.1 Effect on data

When a sample is published, a time stamp is determined at the source which is attached to the sample before it is sent. The subscriber stores the time stamp at which the sample is received in the sample as well. In DDS samples are ordered within the history of an instance based on either the source time stamp or the reception

time stamp. This is controlled by means of the `DestinationOrderQosPolicy`.

The `HistoryQosPolicy` controls how many historic samples are stored in a reader. By default, a DataReader has a `KEEP_LAST` history with a depth of `1`. This means that only the 'last' (based on the ordering defined by the `DestinationOrderQosPolicy`) sample for each instance is maintained by the middleware. When a sample is received by the subscriber, it determines whether and where to insert the sample in the history of an instance based on either the source

time stamp or the reception time stamp, potentially replacing an existing sample of

the instance.

BY_SOURCE_ time stamp If samples are ordered by source time stamp and time is set back 1 hour on the subscriber node, nothing changes. If it is set back one hour on the publisher node, samples written after the time has changed have 'older' source time stamps and will therefore not overwrite the samples in the history from before the time changed.

BY_RECEPTION_ time stamp If samples are ordered by destination time stamp and time is set back back 1 hour on the publisher node, nothing changes. If it is set back one hour on the subscriber node, samples delivered after the time has changed have 'older' reception time stamps and will therefore not overwrite the samples in the history from before the time changed.

2.5.2 Effect on processing

Processing of relative time actions, actions for which a time contract exists with local entities (*e.g.*, inter-process leases, wait for attachment of a service) or time contracts involving remote parties (*e.g.*, heartbeats, deadline) may not behave as expected when time is advanced discontinuously by an operator or when a system is suspended (*e.g.*, hibernate or standby) and resumed. If the OS doesn't support alternative clocks that aren't influenced by this, the middleware may for example stop working because `spliced` doesn't seem to have updated its lease on time, causing services/applications to conclude that `spliced` isn't running anymore.

Also, timed waits may not have the expected duration; too short when time is advanced and too long when time is turned back. Modern Windows and Linux OS's provide these alternative clocks. If the clocks are available, the middleware will use these to prevent the adverse effects of observed time-jumps on its processing.

2.5.3 Background information

The basic clock used for time stamps of data published in DDS is the real-time clock. This time is expressed as the time since the Unix epoch (00:00:00 on Thursday the 1st of January 1970, UTC). All systems support some form of a real-time clock. For most distributed systems that use time, the clocks on different nodes should have similar notions of time progression and because all systems try to keep track of the actual time as accurately as possible, the real-time clock is typically a very good distributed notion of time. If a machine is not synchronised with the actual time, correcting the offset will cause the real-time clock to become discontinuous. These discontinuities make it impossible even to track relative times, so this is where monotonic clocks are needed. However, not all systems have support for monotonic clocks with near real-time time progression.

The following clock-types are used by the middleware to cope with time discontinuities in processing of data, local leases and remote time based contracts if supported by the OS.

Real-time clock This is the main clock used for time stamps on data and data-related actions. This time is typically kept close to the actual time by the OS by means of NTP or the like. This clock can also be provided by the customer through the *'UserClock'* functionality (`//OpenSplice/Domain/UserClockService` is fully described in the *Configuration* section).

Monotonic clock This is a clock that never jumps back and which provides a measure for the time a machine has been running since boot. When the time is adjusted, this clock will not jump forward or backward. This clock doesn't include the time spent when a machine was suspended.

Elapsed time clock This is also a monotonic clock, since it measures the elapsed time since some undefined, but fixed time. This clock is also not affected by adjusting the real-time clock, but it does include time the machine was suspended.

2.6 Time stamps and year 2038 limit

The DDS_Time_t time stamp definition contains a 32-bit second field with an epoch of 01-01-1970. As a result of this the second field is unable to represent a time after year 2038. From version 6.7 this problem is addressed by changing the second field to a 64-bit representation. For now this change is only done for CORBA C++ and CORBA Java and all internal DDS data structures. All other language bindings still use the 32-bit representation. Version 6.7 is fully compatible with older versions and will communicate by default in the 32-bit time representation with other nodes. If the domain/y2038Ready option is set, the node will

use the new 64-bit second representation which makes it incompatible with older nodes prior to version 6.7. (`//OpenSplice/Domain/y2038Ready` is fully described in the *Configuration* section)

2.6.1 CORBA C++

By default the CORBA C++ library (dcpsccpp) that comes with OpenSplice is built with support for the 32-bit DDS_Time_t representation. To rebuild this library to get support for the new 64-bit DDS_Time_t representation please look at the `OSPL_HOME/custom_lib/ccpp/README` document which explains how to do this.

2.6.2 CORBA Java

By default the CORBA Java library (dcpscj.jar) that comes with OpenSplice is built with support for the 32-bit DDS_Time_t representation. A new library dcpscj_y2038_ready.jar is added which supports the new 64-bit DDS_Time_t representation. This library can be used when time stamps beyond year 2038 are needed.

2.6.3 Migration

client-durability Users that use client-durability cannot use times beyond 2038. This is because the client durability protocol uses DDS_Time_t in 32-bit. Also, Lite and Cafe do not support 64-bit yet.

DDS_Time_t in user data model Users that currently use DDS_Time_t in their user-defined data structures cannot migrate a running system. If they want to migrate, the complete system must be shut down and delete all storages containing the old 32-bit dds_time topics stored by the durability service. Rebuild the data models with the new 64-bit dds_time topics and restart the system. Running a mixed environment with old and new dds_time structures will result in topic mismatches.

Record and Replay (RnR) service Users that use the Record and Replay service cannot use time beyond 2038. This is because the RnR service uses 32-bit times in the provided api.

No client durability and no DDS_Time_t usage Customers that do not use DDS_Time_t in their user-defined data structures AND do not use client durability can migrate in two steps:

- First update all nodes to minimal version 6.7 to be compatible with the 64-bit time stamps, but don't set the domain/y2038Ready option
- If all nodes are running compatible versions, node by node can be changed to use 64-bit time stamps by setting the domain/y2038Ready option to true.

2.6.4 Platform support

- Linux 64-bit: On 64-bit platforms linux already supports 64-bit time. No action required.
- Linux 32-bit: On 32-bit platforms linux support for 64-bit time stamps is still in development. To provide y2038 safe time in GLIBC it is proposed that the user code defines _TIME_BITS=64 to get 64bit time support. When GLIBC sees _TIME_BITS=64 or when the system is 64bit it will set __USE_TIME_BITS64 to indicate that it will use 64bit time. Note that this is not yet supported. See: https://sourceware.org/glibc/wiki/Y2038ProofnessDesign?rev=83
- Windows: 64-bit time stamps are supported

NOTE: Network Time Protocol: (This is outside the scope of OpenSplice) When NTP is used then there may be a problem that the time stamp will rollover in 2036. This may not be an issue when version 4 of the NTP protocol is used which provides specification of an era number and era offset.

2.6.5 DDS_Time structure change

The new DDS_Time representation which contains a 64-bit second field:

```
module DDS {
    struct Time_t {
        long long sec;
        unsigned long nanosec;
    };
```

The original DDS_Time representation with a 32-bit second field:

```
module DDS {
    struct Time_t {
        long sec;
        unsigned long nanosec;
    };
```

3

Service Descriptions

Vortex OpenSplice middleware includes several services; each service has a particular responsibility. All of the services are described in the following sections.

The *Shared Memory architecture* shows all of the services included with Vortex OpenSplice.

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 explain each of the services and their responsibilities.

The Domain Service

The Durability Service

The Networking Service

The DDSI2 and DDSI2E Networking Services

The NetworkingBridge Service

The Tuner Service

The DbmsConnect Service

For the Recording and Replay Service, see the its own specific guide.

Vortex OpenSplice middleware and its services can be configured using easy-to-maintain XML files.

Full details of how to use XML files to configure the elements and attributes of all Vortex OpenSplice services are given in the *Configuration* section.

4

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. Without an database size configured 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. To be able to manage the maximum database size a database size can also be given in the *single process* mode. Then the Domain Service creates the DDS database within the heap memory of the process with the given size and will use it's own memory manager in this specific allocated memory.

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 are detailed descriptions of all of the available configuration parameters and their purpose in the *Configuration* section

5

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 are detailed descriptions of all of the available configuration parameters and their purpose in the *Configuration* section.

5.1 Durability Service Purpose

Vortex OpenSplice 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 implies 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 Vortex OpenSplice, the realisation of the non-volatile properties is the responsibility of the durability service. Maintenance and provision of historical data could in theory be done by a single durability service in the domain, but for fault-tolerance and efficiency one durability service is usually running on every computing node. These durability services are on the 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.

5.2 Durability Service Concepts

The following subsections describe the concepts that drive the implementation of the OpenSplice Durability Service.

5.2.1 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 Merge policy) to and from nodes that have equal or different roles.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/Domain/Role`
- `//OpenSplice/DurabilityService/NameSpaces/Policy/Merge[@scope]`

5.2.2 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 a name-space is 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); 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 the 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/NameSpace`

5.2.3 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy`

5.2.3.1 Alignment policy

The durability services in a domain are on the 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 the 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 the 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy[@aligner]`
- `//OpenSplice/DurabilityService/NameSpaces/Policy[@alignee]`

5.2.3.2 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'.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy[@durability]`

5.2.3.3 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 Persistent data injection). This mechanism will, during the startup of the durability service, negotiate with other services which one has the best set (see Master selection). 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 any 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 the system level.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy[@delayedAlignment]`

5.2.3.4 Merge policy

A *'split-brain syndrome'* can be described as the situation in which two different nodes (possibly) have a different perception of (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. This syndrome also 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 of the set of data will be different.

In many cases, after this has occurred the exchange of information 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, but this will 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 Vortex OpenSplice 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.

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

REPLACE Dispose and replace all historical data by the data set that is available on the connecting node. Because all data is disposed first, a side effect is that instances present both before and after the merge operation transition through `NOT_ALIVE_DISPOSED` and end up as *NEW* instances, with corresponding changes to the instance generation counters.

CATCHUP Updates the historical data to match the historical data on the remote node by disposing those instances available in the local set but not in the remote set, and adding and updating all other instances. The resulting data set is the same as that for the *REPLACE* policy, but without the side effects. In particular, the instance state of instances that are both present on the local node and remote node and for which no updates have been done will remain unchanged.

Note that *REPLACE* and *CATCHUP* result in the same data set, but the instance states of the data may differ.

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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy/Merge`

5.2.3.5 Prevent aligning equal data sets

As explained in previous sections, temporary disconnections can cause durability services to get out-of-sync, meaning that their data sets may diverge. To recover from such situations merge policies have been defined (see Merge policy) where a user can specify how to combine divergent data sets when they become reconnected. Many of these situations involve the transfer of data sets from one durability service to the other. This may generate a considerable amount of traffic for large data sets.

If the data sets do not get out-of-sync during disconnection it is not necessary to transfer data sets from one durability service to the other. Users can specify whether to compare data sets before alignment using the `equalityCheck` attribute. When this check is enabled, hashes of the data sets are calculated and compared; when they are equal, no data will be aligned. This may save valuable bandwidth during alignment. If the hashes are different then the complete data sets will be aligned.

Comparing data sets does not come for free as it requires hash calculations over data sets. For large sets this overhead may become significant; for that reason is not recommended to enable this feature for frequently-changing data sets. Doing so will impose the penalty of having to calculate hashes when the hashes are likely to differ and the data sets need to be aligned anyway.

Comparison of data sets using hashes is currently only supported for operational nodes that diverge; no support is provided during initial startup.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy[@equalityCheck]`

5.2.3.6 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, independently 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'*.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/NameSpaces/Policy[@nameSpace]`

5.2.3.7 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 grow exponentially 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.

5.3 Mechanisms

5.3.1 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.

5.3.2 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,0 the durability service will terminate automatically as well.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/Network`

5.3.3 Interaction with applications

The durability service is responsible for providing 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 such a case historical data still is delivered by the durability service when it becomes available.

5.3.4 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/Network/Alignment/RequestCombinePeriod`

5.3.5 Tracing

Configuring durability services throughout a domain and finding out what exactly happens during the lifecycle of the service can prove difficult.

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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/Tracing`

5.4 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.

5.4.1 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/Network/Heartbeat`

5.4.2 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 the section about the Name-spaces 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.

5.4.3 Master selection

For each Namespace and Role combination there shall be at most one Master Durability Service. The Master Durability Service coordinates single source re-publishing of persistent data and to allow parallel alignment after system start-up, and to coordinate recovery of a split brain syndrome after connecting nodes having selected a different Master indicating that more than one state of the data may exist.

Therefore after system start-up as well as after any topology change (i.e. late joining nodes or leaving master node) a master selection process will take place for each affected Namespace/Role combination.

To control the master selection process a masterPriority attribute can be used.

Each Durability Service will have a configured masterPriority attribute per namespace which is an integer value between 0 and 255 and which specifies the eagerness of the Durability Service to become Master for that name-space. The values 0 and 255 have a special meaning. Value 0 is used to indicate that the Durability Service will never become Master. The value 255 is used to indicate that the Durability Service will not use priorities but instead uses the legacy selection algorithm. If not configured the default is 255.

During the master selection process each Durability service will exchange for each namespace the masterPriority and quality. The namespace quality is the timestamp of the latest update of the persistent data set stored on disk and only plays a role in master selection initially when no master has been chosen before and persistent data has not been injected yet.

Each Durability Service will determine the Master based upon the highest non zero masterPriority and in case of multiple masters further select based on namespace quality (but only if persistent data has not been injected before) and again in case of multiple masters select the highest system id. The local system id is an arbitrary value which unique identifies a durability service. After selection each Durability Service will communicate their determined master and on agreement effectuate the selection, on disagreement which may occur if some Durability Services had a temporary different view of the system this process of master selection will restart until all Durability

Services have the same view of the system and have made the same selection. If no durability services exists having a masterPriority greater than zero then no master will be selected.

Summarizing, the precedence rules for master selection are (from high to low):

1. The namespace masterPriority
2. The namespace quality, if no data has been injected before.
3. The Durability Service system id, which is unique for each durability service.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DurabilityService/Network/InitialDiscoveryPeriod`

5.4.4 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.

If 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 at a different moment in time. Therefore only one of them should be allowed to 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. If 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* — If 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. If 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* – If 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 postpones alignment until a master becomes available in the domain.

It is strongly discouraged to re-inject persistent data from a persistent store in a running system after persistent data has been published. Behaviour of re-injecting persistent stores in a running system is not specified and may be changed over time.

5.4.5 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.

5.4.6 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 either 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.

5.4.7 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 services.

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.

5.4.8 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.

5.5 Threads description

This section contains a short description of each durability thread. When applicable, relevant configuration parameters are mentioned.

5.5.1 ospl_durability

This is the main durability service thread. It starts most of the other threads, e.g. the listener threads that are used to receive the durability protocol messages, and it initiates initial alignment when necessary. This thread is responsible for periodically updating the service-lease so that the splice-daemon is aware the service is still alive. It also periodically (every 10 seconds) checks the state of all other service threads to detect if deadlock has occurred. If deadlock has occurred the service will indicate which thread didn't make progress and action can be taken (e.g. the service refrains from updating its service-lease, causing the splice daemon to execute a failure action). Most of the time this thread is asleep.

5.5.2 conflictResolver

This thread is responsible for resolving conflicts. If a conflict has been detected and stored in the conflictQueue, the conflictResolver thread takes the conflict, checks whether the conflict still exists, and if so, starts the procedure to resolve the conflict (i.e., start to determine a new master, send out sample requests, etc).

5.5.3 statusThread

This thread is responsible for the sending status messages to other durability services. These messages are periodically sent between durability services to inform each other about their state (.e.,g INITIALIZING or TERMINATING).

Configuration parameters:

- `//OpenSplice/DurabilityService/Watchdog/Scheduling`

5.5.4 d_adminActionQueue

The durability service maintains a queue to schedule timed action. The d_adminActionQueue periodically checks (every 100 ms) if an action is scheduled. Example actions are: electing a new master, detection of new local groups and deleting historical data.

Configuration parameters:

- `//OpenSplice/DurabilityService/Network/Heartbeat/Scheduling`

5.5.5 AdminEventDispatcher

Communication between the splice-daemon and durability service is managed by events. The AdminEventDispatcher thread listens and acts upon these events. For example, the creation of a new topic is noticed by the splice-daemon, which generates an event for the durability service, which schedules an action on to request historical data for the topic.

5.5.6 groupCreationThread

The groupCreationThread is responsible for the creation of groups that exist in other federations. When a durability service receives a newGroup message from another federation, it must create the group locally in order to acquire data for it. Creation of a group may fail in case a topic is not yet known. The thread will retry with a 10ms interval.

5.5.7 sampleRequestHandler

This thread is responsible for the handling of sampleRequests. When a durability service receives a d_sampleRequest message (see the sampleRequestListener thread) it will not immediately answer the request, but wait some time until the time to combine requests has been expired (see //OpenSplice/DurabilityService/Network/Alignment/RequestCombinePeriod). When this time has expired the sampleRequestHandler will answer the request by collecting the requested data and sending the data as d_sampleChain messages to the requestor.

Configuration parameters:

- `//OpenSplice/DurabilityService/Network/Alignment/AlignerScheduling`

5.5.8 resendQueue

This thread is responsible for injection of message in the group after it has been rejected before. When a durability service has received historical data from another fellow, historical data is injected in the group (see d_sampleChain). Injection of historical data can be rejected, e.g., when a resource limits are being used. When this happens, a new attempt to inject the data is scheduled overusing the resendQueue thread. This thread will try to deliver the data 1s later.

Configuration parameters:

- `//OpenSplice/DurabilityService/Network/Alignment/AligneeScheduling`

5.5.9 masterMonitor

The masterMonitor is the thread that handles the selection of a new master. This thread is invoked when the conflict resolver detects that a master conflict has occurred. The masterMonitor is responsible for collecting master proposals from other fellows and sending out proposals to other fellows.

5.5.10 groupLocalListenerActionQueue

This thread is used to handle historical data requests from specific readers, and to handle delayed alignment (see //OpenSplice/DurabilityService/NameSpaces/Policy[@delayedAlignment])

5.5.11 d_groupsRequest

The d_groupsRequest thread is responsible for processing incoming d_groupsRequest messages from other fellows. When a durability service receives a message from a fellow, the durability service will send information about its groups to the requestor by means of d_newGroup messages. This thread collects group information, packs it in d_newGroup messages and send them to the requestor. This thread will only do something when a d_groupsRequest has been received from a fellow. Most of the time it will sleep.

5.5.12 d_nameSpaces

This thread is responsible for processing incoming d_nameSpaces messages from other fellows. Durability services send each other their namespaces state so that they can detect potential conflicts. The d_nameSpaces thread processes and administrates every incoming d_nameSpace. When a conflict is detected, the conflict is scheduled which may cause the conflictResolver thread to kick in.

5.5.13 d_nameSpacesRequest

The d_nameSpacesRequest thread is responsible for processing incoming d_nameSpacesRequest messages from other fellows. A durability service can request the namespaces form a fellow by sending a d_nameSpacesRequest message to the fellow. Whenever a durability service receives a d_nameSpacesRequest messages it will respond by sending its set of namespaces to the fellow. The thread handles incoming d_nameSpacesRequest messages. As a side effect new fellows can be discovered if a nameSpacesRequest is received from an unknown fellow.

5.5.14 d_status

The d_status thread is responsible for processing incoming d_status messages from other fellows. Durability services periodically send each other status information (see the statusThread). NOTE: in earlier versions missing d_status messages could lead to the conclusion that a fellows has been removed. In recent versions this mechanism has been replaced so that the durability service slaves itself to the liveness of remote federations based on heartbeats (see thread dcpsHeartbeatListener). Effectivily, the d_status message is not used anymore to verify liveliness of remote federations, it is only used to transfer the durability state of a remote federation.

5.5.15 d_newGroup

The d_newGroup thread is responsible for handling incoming d_newGroup messages from other fellows. Durability services inform each other about groups in the namespaces. They do that by sending d_newGroup messages to each other (see also thread d_groupsRequest). The d_newGroup thread is responsible for handling incoming groups.

5.5.16 d_sampleChain

The d_sampleChain thread handles incoming d_sampleChain messages from other fellows. When a durability service answers an d_sampleRequest, it must collect the requested data and send it to the requestor. The collected data is packed in d_sampleChain messages. The d_sampleChain thread handles incoming d_sampleChain messages and applies the configured merge policy for the data. For example, in case of a MERGE it injects all the received data in the local group and delivers the data to the available readers.

Configuration parameters:

- `//OpenSplice/DurabilityService/Network/Alignment/AligneeScheduling`

5.5.17 d_sampleRequest

The d_sampleRequest thread is responsible for handling incoming d_sampleRequest messages from other fellows. A durability service can request historical data from a fellow by sending a d_sampleRequest message. The d_sampleRequest thread is used to process d_sampleRequest messages. Because d_sampleRequest messages are not handled immediately, they are stored in a list and handled later on (see thread sampleRequestHandler).

5.5.18 d_deleteData

The d_deleteData thread is responsible for handling incoming d_deleteData messages from other fellows. An application can call delete_historical_data(). This causes all historical data up till now to be deleted. To propagate deletion of historical data to all available durability services in the system, durability services send each other a d_deleteData message. The d_deleteData thread handles incoming d_deleteData messages and takes care that the relevant data is deleted. This thread will only be active after delete_historical_data() is called.

5.5.19 dcpsHeartbeatListener

The dcpsHeartbeatListener is responsible for the liveliness detection of remote federations. This thread listens to DCPSHeartbeat messages that are sent by federation. It is used to detect new federations or federations that disconnect. This thread will only do something when there is a change in federation topology. Most of the time it will be asleep.

5.5.20 d_capability

The thread is responsible for processing d_cability messages from other fellows. As soon as a durability service detects a fellow it will send its list of capabilities to the fellow. The fellow can use this list to find what functionality is supported by the durability service. Similarly, the durability service can receive capabilities from the fellow. This thread is used to process the capabilities sent by a fellow. This thread will only do something when a fellow is detected. | |

5.5.21 remoteReader

The remoteReader thread is responsible for the detection of remote readers on other federations. The DDSI service performs discovery and reader-writing matching. This is an asynchronous mechanism. When a durability service (say A) receives a request from a fellow durability service (say B) and DDSI is used as networking service, then A cannot be sure that DDSI has already detected the reader on B that should receive the answer to the request. To ensure that durability services will only answer if all relevant remote readers have been detected, the remoteReader thread keeps track of the readers that have been discovered by ddsi. Only when all relevant readers have been discovered durability services are allowed to answer requests. This prevents DDSI from dropping messages destined for readers that have not been discovered yet.

5.5.22 persistentDataListener

The persistentDataListenerThread is responsible for persisting durable data. When a durability service retrieves persistent data, the data is stored in a queue. The persistentDataListener thread retrieves the data from the queue and stores it in the persistent store. For large data sets persisting the data can take quite some time, depending mostly on the performance of the disk.

Note this thread is only created when persistency is enabled (//OpenSplice/DurabilityService/Persistent/StoreDirectory has a value set):

Configuration parameters:

- `//OpenSplice/DurabilityService/Persistent/Scheduling`

5.5.23 historicalDataRequestHandler

This thread is responsible for handling incoming historicalDataRequest messages from durability clients. In case an application does not have the resources to run a durability service but still wants to acquire historical data it can configure a client. The client sends HistoricalDataRequest messages to the durability service. These messages are handled by the historicalDataRequestHandler thread.

Note this thread is only created when client durability is enabled (//OpenSplice/DurabilityService/ClientDurability element exists)

5.5.24 durabilityStateListener

This thread is responsible for handling incoming durabilityStateRequest messages from durability clients.

Note this thread is only created when client durability is enabled (//OpenSplice/DurabilityService/ClientDurability element exists)

6

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 Secure Native Networking Service.

There are detailed descriptions of all of the available configuration parameters and their purpose in the *Configuration* section.

6.1 The Native Networking Service

For large-scale LAN-based systems that demand maximum throughput, the native RTNetworking service is the optimal implementation of DDS networking for Vortex OpenSplice 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.

Please refer to the *Configuration* section for fully-detailed descriptions of how to configure:

- `//OpenSplice/NetworkService`
- `//OpenSplice/SNetworkService`

6.2 The Secure Native Networking Service

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

Please refer to the *Configuration* section for details.

6.2.1 Compression

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

In early OpenSplice 6.x releases, the *zlib* library was used at its default setting whenever the compression option on a network partition was enabled. Now 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. These settings apply to all partitions in which compression is enabled.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/NetworkService/Compression`

6.2.1.1 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.

6.2.1.2 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.

6.2.1.3 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.

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.

6.2.1.4 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;
}
```

6.2.2 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.

Please refer to the *Configuration* section for fully-detailed descriptions of how to configure:

- `//OpenSplice/NetworkService/Compression[@PluginLibrary]`
- `//OpenSplice/NetworkService/Compression[@PluginInitFunction]`
- `//OpenSplice/NetworkService/Compression[@PluginParameter]`

6.2.3 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.*

7

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.

Please refer to the *Configuration* section fully-detailed descriptions of:

- `//OpenSplice/DDSI2Service`
- `//OpenSplice/DDSI2EService`

7.1 DDSI Concepts

Both DDSI 2.1 and 2.2 standards are very intimately related to the DDS 1.2 and 1.4 standards, 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 Vortex OpenSplice implementation, DDSI2, which are addressed in subsequent sections. This division was chosen to aid readers interested in interoperability to understand where the specification ends and the Vortex OpenSplice implementation begins.

7.1.1 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 that accidental cross-domain communication is impossible with the default mapping.

DDSI does not communicate the DCPS port number in the discovery protocol; it assumes that each domain id maps to a unique port number. 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.

7.1.2 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; 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.

7.1.3 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, *etc.*. The use of these techniques is required to allow for a performant DDSI implementation, whilst avoiding the need for sending redundant messages.

7.1.4 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 Vortex OpenSplice the durability service implements all three durability kinds without requiring any special support from the networking services, ensuring that the full set of durability features is always available between Vortex OpenSplice nodes.

7.1.5 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.

7.2 Vortex OpenSplice DDSI2 specifics

7.2.1 Translating between Vortex OpenSplice and DDSI

Given that DDSI is the DDS interoperability specification, that the mapping between DCPS entities and DDSI entities is straightforward, and that Vortex OpenSplice is a full implementation of the DDS specification, one

might expect that relationship between Vortex 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 Vortex 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 Vortex 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 are 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 Vortex 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.

7.2.2 Federated versus Standalone deployment

As has been described elsewhere (see the *Overview* in this *Guide* and also the *Getting Started Guide*), Vortex 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 Vortex OpenSplice service processes on that node, or a standalone one in which a single process incorporates the Vortex OpenSplice services as libraries.

This *Guide* 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.

7.2.3 Discovery behaviour

7.2.3.1 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 the *Configuration* section).

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 Vortex OpenSplice the built-in topics are first-class topics, *i.e.* the only difference between application topics and the built-in topics in Vortex OpenSplice is that the built-in topics are pre-defined and that they are published and used by the Vortex 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 Vortex 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.

7.2.3.2 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 of 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.

7.2.3.3 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 Combining multiple participants and refer to the *Configuration* section for a detailed description of `//OpenSplice/DDSI2Service/Internal/SquashParticipants`.) 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/BuiltinEndpointSet`
- `//OpenSplice/DDSI2Service/Internal/ConservativeBuiltinReaderStartup`

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.

7.2.3.4 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. 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DDSI2Service/Internal/WriterLingerDuration`

7.2.3.5 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.

7.2.4 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 Vortex 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.) 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` bytes in unacknowledged samples, it stalls the writer until the number of bytes in unacknowledged samples drops below `Internal/Watermarks/WhcLow`.

While ideally only the one writer would be stalled, the interface between the Vortex OpenSplice kernel and DDSI2 is such that other outgoing traffic may be stalled as well. See Unresponsive readers & head-of-stream blocking.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/AggressiveKeepLast1Whc`
- `//OpenSplice/DDSI2Service/Internal/Watermarks/WhcHigh`
- `//OpenSplice/DDSI2Service/Internal/Watermarks/WhcLow`

7.2.5 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`), 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 Vortex 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`. If this timeout is set larger than roughly the domain expiry time (`//OpenSplice/Domain/Lease/ExpiryTime`), 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 Channel configuration.

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. Readers that are marked unresponsive by a writer may therefore observe sample loss. The 'sample lost' status of the data readers can be used to detect this.

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, and so on.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/AggressiveKeepLast1Whc`

- `//OpenSplice/DDSI2Service/Internal/ResponsivenessTimeout`
- `//OpenSplice/Domain/Lease/ExpiryTime`

7.2.6 Handling of multiple partitions and wildcards

7.2.6.1 Publishing in multiple partitions

A variety of design choices allow Vortex 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 Vortex 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.

7.2.6.2 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 previous subsection (Publishing in multiple partitions).

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

7.3 Network and discovery configuration

7.3.1 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/NetworkService/General/NetworkInterfaceAddress`

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.

7.3.1.1 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` 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`.

To disable incoming multicasts, or to control from which interfaces multicasts are to be accepted, one can use the `General/MulticastRecvInterfaceAddresses` setting. This allows listening on no interface, the preferred, all or a specific set of interfaces.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/General/AllowMulticast`
- `//OpenSplice/DDSI2Service/Internal/SuppressSpdpMulticast`
- `//OpenSplice/DDSI2Service/General/MulticastRecvNetworkInterfaceAddress`

7.3.1.2 Discovery configuration

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` 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, 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. 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 Controlling port numbers) 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/Domain/Id`
- `//OpenSplice/DDSI2Service/Discovery/DomainId`
- `//OpenSplice/DDSI2Service/Discovery/SPDPMulticastAddress`
- `//OpenSplice/DDSI2Service/Discovery/Peers`
- `//OpenSplice/DDSI2Service/Discovery/ParticipantIndex`

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.

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.

Endpoint discovery

Although the SEDP protocol never requires any configuration, the network partitioning of Vortex 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.

7.3.2 Combining multiple participants

In a Vortex 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 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 Vortex 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 Vortex OpenSplice the built-in topics are not derived from the DDSI discovery, and hence in a Vortex OpenSplice-only network the use of the `Internal/SquashParticipants` setting will not result in any loss of information in the DCPS API or in the Vortex OpenSplice tools such as the Tester.

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

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DDSI2Service/Internal/SquashParticipants`

7.3.3 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; however, they *are* configurable and the interested reader is referred to the DDSI 2.1 or 2.2 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.

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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Discovery/Ports/Base`
- `//OpenSplice/DDSI2Service/Discovery/Ports/DomainGain`
- `//OpenSplice/DDSI2Service/Discovery/Ports/ParticipantGain`
- `//OpenSplice/DDSI2Service/Discovery/ParticipantIndex`
- `//OpenSplice/DDSI2Service/Compatibility/ManySocketsMode`

7.3.4 Coexistence with Vortex OpenSplice RTNetworking

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

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DDSI2Service/General/CoexistWithNativeNetworking`

7.4 Data path configuration

7.4.1 Data path architecture

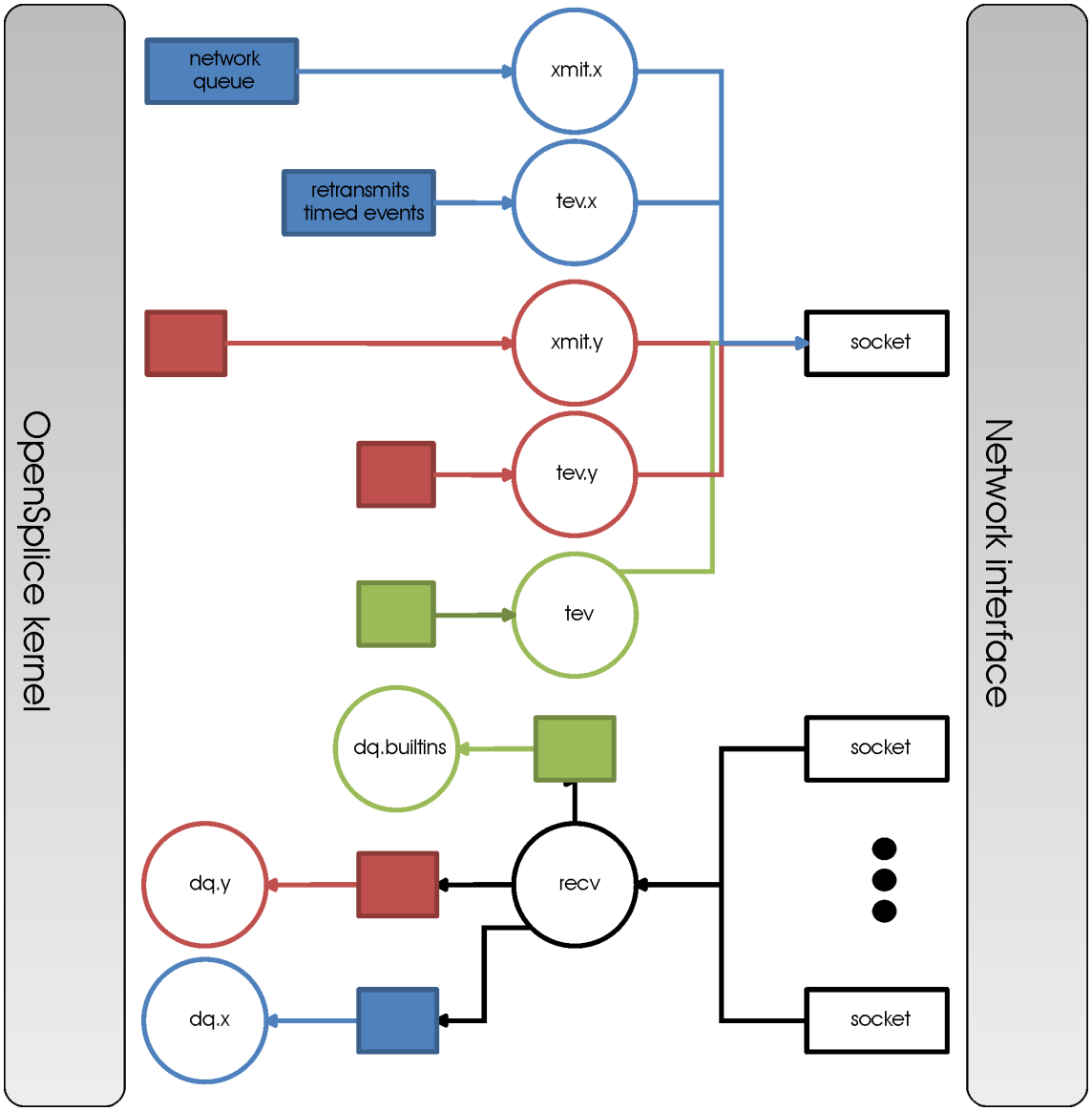
The data path in DDSI2 consists of a transmitting and a receiving side. The main path in the transmit side accepts data to be transmitted from the Vortex OpenSplice kernel *via* a network queue and administrates and formats the

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.

The diagram Data flow using two channels 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.

Data flow using two channels



7.4.2 Transmit-side configuration

7.4.2.1 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 Channel configuration.

Each channel has its own transmit thread, draining a queue with samples to be transmitted from the Vortex 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 Unresponsive readers & head-of-stream blocking.

The DDSI control messages (Heartbeat, AckNack, *etc.*) 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.

7.4.2.2 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`, 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. 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` controls the length of this time window.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/NackDelay`
- `//OpenSplice/DDSI2Service/Internal/RetransmitMerging`
- `//OpenSplice/DDSI2Service/Internal/RetransmitMergingPeriod`

7.4.2.3 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`, which limits the number of retransmit messages, the second `Internal/MaxQueuedRexmitBytes` which limits the number of bytes. The latter is automatically set based on the combination of the allowed transmit bandwidth and the `Internal/NackDelay` setting, as an approximation of the likely time until the next potential retransmit request from the reader.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/MaxQueuedRexmitMessages`
- `//OpenSplice/DDSI2Service/Internal/MaxQueuedRexmitBytes`
- `//OpenSplice/DDSI2Service/Internal/NackDelay`

7.4.2.4 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`, 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`, 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/FragmentSize`
- `//OpenSplice/DDSI2Service/Internal/MaxMessageSize`

7.4.3 Receive-side configuration

7.4.3.1 Receive processing

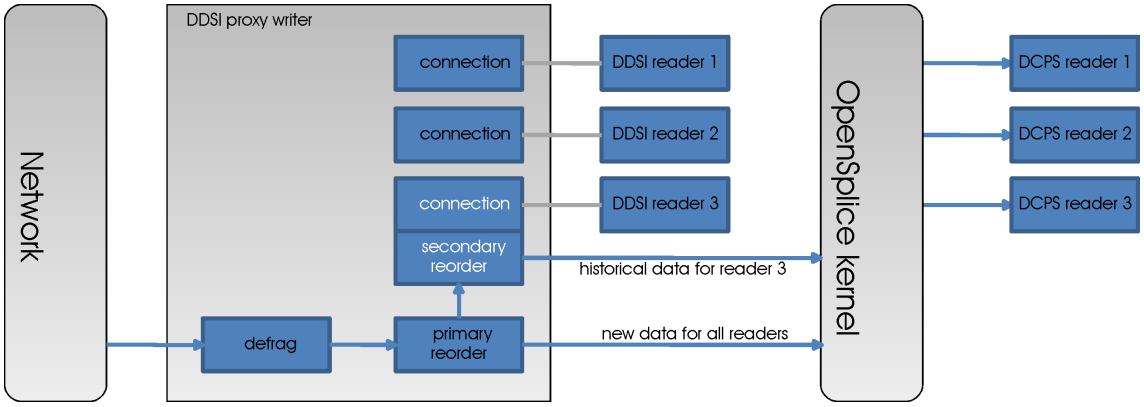
Receiving of data is split into multiple threads, as also depicted in the overall DDSI2 data path diagram Data flow using two channels:

- 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 the diagram Proxy writer with multiple data readers.

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` and for unreliable data to `Internal/DefragUnreliableMaxSamples`.

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`, the size of the second to `Internal/SecondaryReorderMaxSamples`.

In between the receive thread and the delivery threads sit queues, of which the maximum size is controlled by the `Internal/DeliveryQueueMaxSamples` 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/DefragReliableMaxSamples`
- `//OpenSplice/DDSI2Service/Internal/DefragUnreliableMaxSamples`
- `//OpenSplice/DDSI2Service/Internal/PrimaryReorderMaxSamples`
- `//OpenSplice/DDSI2Service/Internal/SecondaryReorderMaxSamples`
- `//OpenSplice/DDSI2Service/Internal/DeliveryQueueMaxSamples`

7.4.3.2 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Internal/SynchronousDeliveryLatencyBound`
- `//OpenSplice/DDSI2Service/Internal/SynchronousDeliveryPriorityThreshold`

7.4.4 Direction-independent settings

7.4.4.1 Maximum sample size

DDSI2 provides a setting, `Internal/MaxSampleSize`, 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 Vortex OpenSplice info log. DDSI2 behaves as if the sample never existed. The current structure of the interface between the Vortex OpenSplice kernel and the Vortex 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.

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DDSI2Service/Internal/MaxSampleSize`

7.5 DDSI2E Enhanced features

7.5.1 Introduction to DDSI2E

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.

7.5.2 Channel configuration

7.5.2.1 Channel configuration overview

DDSI2E allows defining *channels*, which are independent data paths within the DDSI service. Vortex 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 Unresponsive readers & head-of-stream blocking. 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, *etc.*), 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 Vortex 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.

7.5.2.2 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 Vortex 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, *etc.*) 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.

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`. 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` to limit the network bandwidth the discovery & liveliness protocols can consume.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2EService/Channels/Channel/QueueSize`
- `//OpenSplice/DDSI2Service/Sizing/NetworkQueueSize`
- `//OpenSplice/DDSI2EService/Channels/Channel/DataBandwidthLimit`
- `//OpenSplice/DDSI2EService/Channels/Channel/AuxiliaryBandwidthLimit`
- `//OpenSplice/DDSI2EService/Internal/AuxiliaryBandwidthLimit`

7.5.2.3 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.

7.5.2.4 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 Thread configuration), another is by setting a bandwidth limit on the auxiliary data of the channel (see Transmit side).

Please refer to the *Configuration* section for a detailed description of:

- `//OpenSplice/DDSI2EService/Channels/Channel/AuxiliaryBandwidthLimit`.)

7.5.2.5 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.

7.5.3 Network partition configuration

7.5.3.1 Network partition configuration 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.

7.5.3.2 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 Local discovery and built-in topics), 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.

7.5.3.3 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.

7.5.3.4 On interoperability

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

7.5.4 Encryption configuration

7.5.4.1 Encryption configuration overview

DDSI2E encryption support allows the definition of *'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.

7.5.4.2 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.

7.6 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 *the diagram* in *Data path architecture*. The properties that can be controlled are:

- stack size,
- scheduling class, and

- scheduling priority.

The threads are named and the attribute `Threads/Thread[@name]` 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.

(See the detailed description of `OpenSplice/DDSI2Service/Threads/Thread[@name]` in the *Configuration* section)

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 Vortex 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 Vortex 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'*.

7.7 Reporting and tracing

DDSI2 can produce highly detailed traces of all traffic and internal activities. It enables 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 Vortex 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 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.

7.8 Compatibility and conformance

7.8.1 Conformance modes

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

(Please refer to the *Configuration* section for a detailed description of `//OpenSplice/DDSI2Service/Compatibility/StandardsConformance`.)

In *pedantic* mode, it strives very hard to strictly conform to the DDSI 2.1 and 2.2 standards. 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-Vortex 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 other vendors' implementations, but this is not the case. The deviations in other vendors' implementations are not required to implement DDSI 2.1 (or 2.2), as is proven by the Vortex OpenSplice DDSI2 service, and they cannot rightly be considered 'true' implementations of the DDSI 2.1 (or 2.2) standard.

The default mode, *lax*, attempts to work around (most of) the deviations of 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 *the next section* for more information.) In lax mode, the Vortex 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 Vortex 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.

7.8.1.1 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 Vortex 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 Vortex OpenSplice DCPS data reader. The problem is that once the Vortex OpenSplice reader informs the RTI writer that it has received all data (using a 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/StandardsConformance`: *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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Compatibility/StandardsConformance`
- `//OpenSplice/DDSI2Service/Compatibility/AckNackNumbitsEmptySet`

7.8.1.2 Compatibility issues with TwinOaks

Interoperability with TwinOaks CoreDX requires:

- `Compatibility/ManySocketsMode`: *true*
- `Compatibility/StandardsConformance`: *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; this 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 that 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.

Please refer to the *Configuration* section for detailed descriptions of:

- `//OpenSplice/DDSI2Service/Compatibility/ManySocketsMode`
- `//OpenSplice/DDSI2Service/Compatibility/StandardsConformance`
- `//OpenSplice/DDSI2Service/Compatibility/AckNackNumbitsEmptySet`
- `//OpenSplice/DDSI2Service/Compatibility/ExplicitlyPublishQosSetToDefault`

8

The NetworkingBridge Service

The OpenSplice NetworkingBridge is a pluggable service that allows bridging of data between networking services. This section gives an overview of the features of the NetworkingBridge.

The configuration parameters that control the behaviour of the NetworkingBridge are described in the *Configuration* section.

8.1 Background

When a networking service is selected that best suits a specific deployment, sometimes a part of the data needs to be obtained from or disclosed to a system that is using a different kind of networking service. The NetworkingBridge allows `DCPSPublications` and `DCPSSubscriptions` to be matched and the related data forwarded between a RTNetworking system and a DDSI2 system and *vice versa*.

The NetworkingBridge employs a fast path in the OpenSplice kernel by directly connecting the network queues of the bridged services. This also allows full end-to-end flow control mechanisms to be realised across the bridge. Which publications/subscriptions are bridged can be controlled by means of white- and black-lists.

The NetworkingBridge relies on the discovery of publications and subscriptions by the common means for the networking services. This means that it relies on the real transient topics, aligned by the Durability service, for the RTNetworking part of the bridge. For the part that connects to DDSI2 the native DDSI2 discovery of end-points is used. In order for DDSI2 to only advertise bridged publications and subscriptions, the `LocalDiscoveryPartition` used for regular discovery should be set to a non-existent partition, as can be seen in the following example. This discovery takes some time and can introduce a short delay before data is bridged.

8.2 Example Configuration

In order to properly configure the NetworkingBridge for bridging data between RTNetworking and DDSI2, both networking services (and the Durability service for the alignment of the builtin topics of the RTNetworking side) have to be configured. Filtering is also configured with the NetworkingBridge.

An example configuration file for bridging of all data (excluding Topic *MyLocalTopic*) in partition *BridgedPartition* is shown below.

```
<OpenSplice>
   <Domain>
      <Name>NetworkingBridgeExample</Name>
      <Id>0</Id>
      <Service name="networking">
         <Command>networking</Command>
      </Service>
      <Service name="ddsi2e">
         <Command>ddsi2e</Command>
      </Service>
      <Service name="nwbridge">
        <Command>nwbridge</Command>
```

```
    </Service>
    <Service name="durability">
      <Command>durability</Command>
    </Service>
  </Domain>
  <NetworkService name="networking">
    <Partitioning>
      <GlobalPartition Address="broadcast"/>
    </Partitioning>
    <Channels>
      <Channel default="true" enabled="true" name="BestEffort" reliable="false">
        <PortNr>54400</PortNr>
      </Channel>
      <Channel enabled="true" name="Reliable" reliable="true">
        <PortNr>54410</PortNr>
      </Channel>
    </Channels>
    <Discovery enabled="true">
      <PortNr>54420</PortNr>
    </Discovery>
  </NetworkService>
  <DDSI2EService name="ddsi2e">
    <Discovery>
      <LocalDiscoveryPartition>ThisIsNotAPartition</LocalDiscoveryPartition>
    </Discovery>
  </DDSI2EService>
  <NetworkingBridgeService name="nwbridge">
    <Include>
      <!-- Multiple entries can be added here with DCPSPartitionTopic expressions
           on what to include -->
      <Entry DCPSPartitionTopic="BridgedPartition.*"/>
    </Include>
    <Exclude>
      <!-- Multiple entries can be added here with DCPSPartitionTopic expressions
           on what to exclude.
           If a DCPSPublication or DCPSSubscription matches both the include- and
           exclude expressions it will be excluded. -->
      <Entry DCPSPartitionTopic="*.MyLocalTopic"/>
    </Exclude>
  </NetworkingBridgeService>
  <DurabilityService name="durability">
    <Network>
      <Alignment>
        <TimeAlignment>FALSE</TimeAlignment>
        <RequestCombinePeriod>
          <Initial>2.5</Initial>
          <Operational>0.1</Operational>
        </RequestCombinePeriod>
      </Alignment>
      <WaitForAttachment maxWaitCount="10">
        <ServiceName>networking</ServiceName>
        <ServiceName>ddsi2e</ServiceName>
      </WaitForAttachment>
    </Network>
    <NameSpaces>
      <NameSpace name="defaultNamespace">
        <Partition>*</Partition>
      </NameSpace>
      <Policy nameSpace="defaultNamespace" durability="Durable"
                        alignee="Initial" aligner="True"/>
    </NameSpaces>
  </DurabilityService>
  <Description>Federated deployment for extending an RTNetworking-based
```

```
                domain into a DDSI network.</Description>
</OpenSplice>
```

9

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 a detailed description of the available configuration parameters and their purpose in the *Configuration* section, starting at the section on `//OpenSplice/NetworkService/Tracing`.

10

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 delivering *'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 third-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.

OpenSplice DbmsConnect is based on SQL:1999 and utilizes ODBC 2.x to interface with third-party DBMS systems. Interoperability has been verified with MySQL 5.0 and Microsoft SQL Server 2008. With limited strict conformance of most RDBMS's to both the SQL as well as the ODBC standard, support for other customer-chosen DBMS systems may require a porting activity of the DbmsConnect service.

As SQL tables have no support for unbounded sequences and sequences of complex types, mapping such DDS_Types to tables is not supported.

10.1 Usage

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

10.2 DDS and DBMS Concepts and Types Mapping

The concepts within DDS and DBMS are related to each other as listed in the table DDS to DBMS mapping: concepts.

DDS to 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 the table DDS to DBMS mapping: primitive types.

DDS to 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>)

DDS to DBMS mapping: complex (composite) types

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

10.3 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.

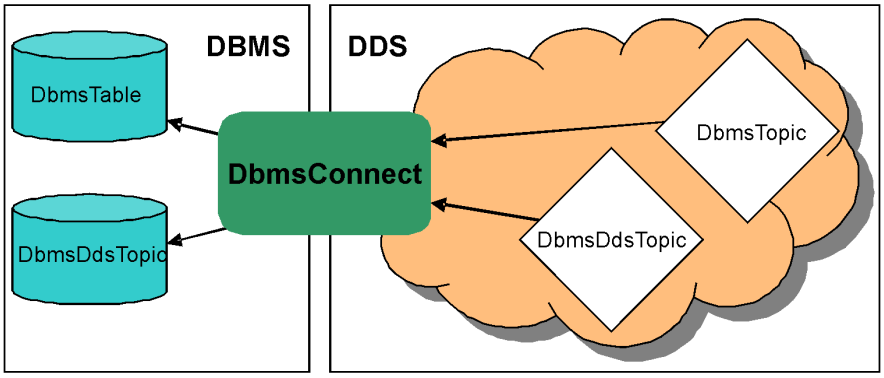
10.4 DDS to DBMS Use Case

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.

10.4.1 DDS to 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 does not yet exist) with the same name. This is shown in The DDS to DBMS scenario.

The DDS to DBMS scenario



10.4.2 DDS to DBMS 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    ...
```

10.4.2.1 DDS to DBMS Configuration 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.

10.5 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.

10.5.1 DBMS to DDS 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 identically-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 The DDS to DBMS scenario.

10.5.2 DBMS to DDS Configuration

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

```
11    ...
13    <DbmsConnectService name="dbmsconnect">
14       <DbmsToDds publish_initial_data="false">
15          <NameSpace partition="dbmsPartition" table="Dbms*"
16             dsn="DSN" usr="USR" pwd="PWD" odbc="ODBC">
17             <Mapping topic="DbmsTopic" table="DbmsTable"
18                publish_initial_data="true"/>
19          </NameSpace>
20       </DbmsToDds>
21    </DbmsConnectService
22    ...
```

10.5.2.1 DBMS to DDS Configuration 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 that set at line *13*.

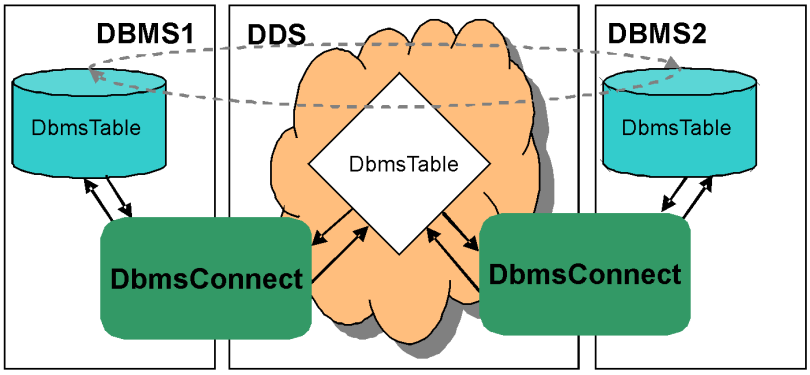
10.6 Replication Use Case

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.

10.6.1 Replication 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 The Replication scenario, where the dashed arrows show the transparent role of DDS in this scenario.

The Replication scenario



10.6.2 Replication 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   ...
```

10.6.2.1 Replication Configuration 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 (analogous to both the DDS to DBMS Use Case and the DBMS to DDS Use Case).

While this (intuitive) cyclic definition would work, more configuration is needed to support this use case properly. In order to prevent modifications from causing (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*.

11

Tools

The Vortex OpenSplice 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 either refers to the appropriate manual, or provides a separate section with a detailed description of the tools' possibilities and command-line interface.

11.1 Introduction

The Vortex OpenSplice 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 Vortex OpenSplice configuration files, providing context sensitive validation and help. More details about this tool can be found in osplconf: the OpenSplice Configuration editor.
- Control & Monitoring Tools:
  - `ospl` (OpenSplice service manager): A tool that can be used to start, stop and monitor the Vortex OpenSplice Domain Service (only applicable to the Federated Deployment Mode). More details about this tool can be found in ospl: the OpenSplice service manager.
  - `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 mmstat: Memory Management Statistics.
  - `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 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_TesterUserGuide.pdf`.
- `nodemon` (OpenSplice Node Monitor): A tool that publishes the following system-monitoring data into the OpenSplice backbone. More details about this tool can be found in the *OpenSplice Node Monitor User Guide* contained in the file `OpenSplice_TNodeMonitorGuide.pdf`.

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

11.2 osplconf: the OpenSplice Configuration editor

The OpenSplice Configuration Editor provides the following command line instruction:

-uri=[URI] — specifies the domain config file that needs to be opened

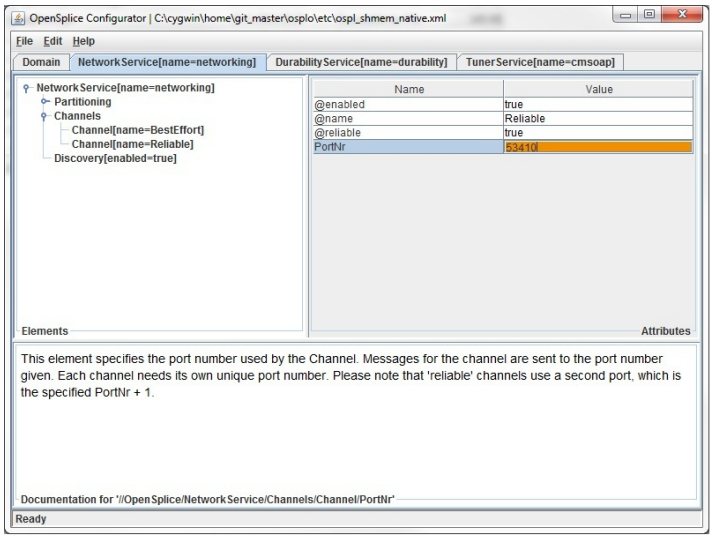
e.g. `osplconf -uri=$OSPL_URI`

When started, the OpenSplice Configuration Editor can help you in several ways to tailor the deployment settings for your Vortex OpenSplice 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 descriptions contained in the *Configuration* section. These descriptions also give some additional information, such as 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:

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.

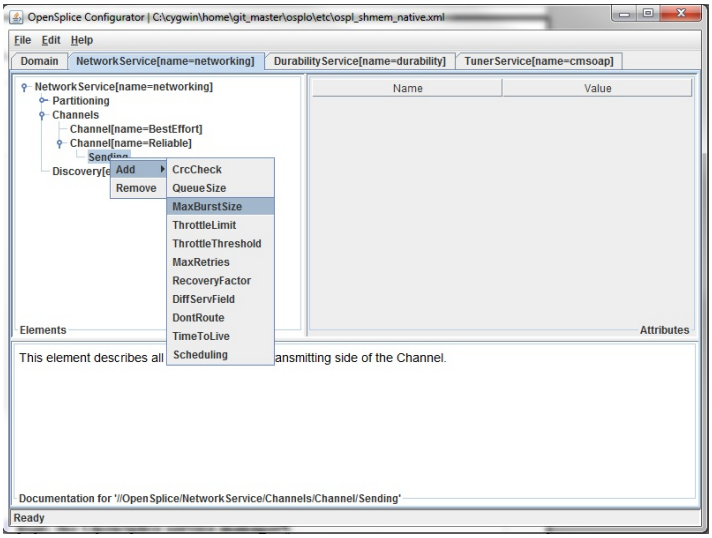
If the appropriate service is not configured, and so its tab is not visible on the top, it can be added by using the top menu-bar (*Edit* > *Add Service*).

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.

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`.

11.3 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.

11.4 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 provide detailed descriptions of the different `mmstat` modes mentioned above.

11.4.1 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.

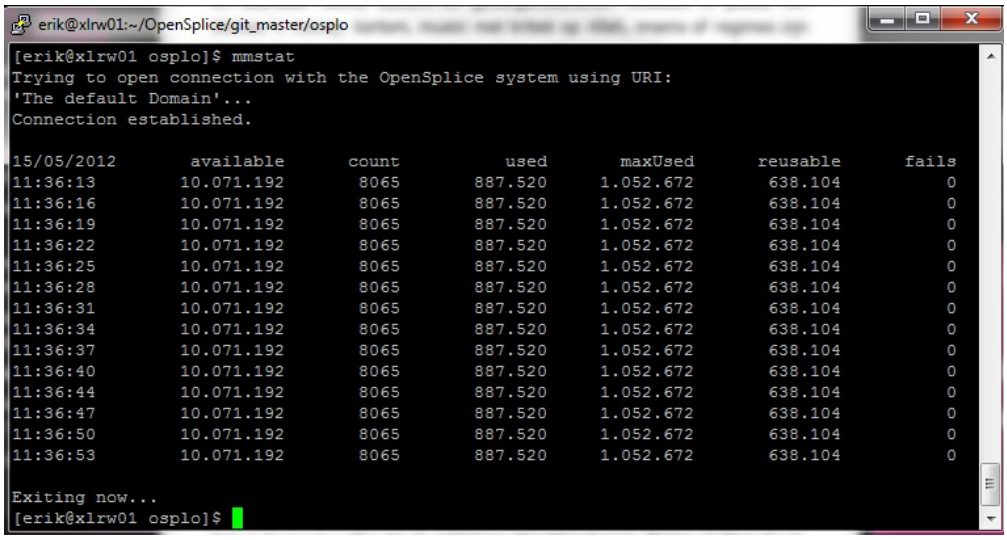
The numbers that will be displayed in this mode are:

- 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.

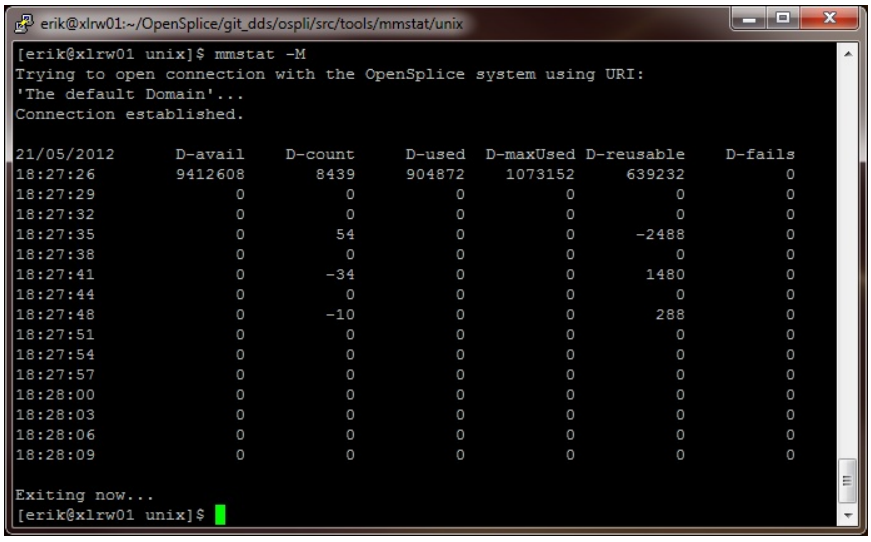
Typical mmstat view



11.4.2 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.

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.

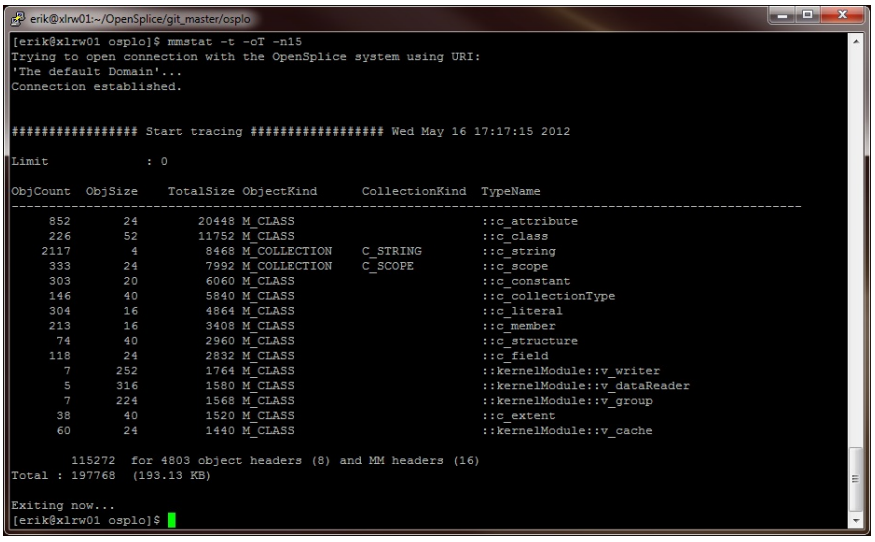
- the difference in the number of objects that is allocated in the shared memory relative to the previous measurement.
- the difference in the amount of shared memory that is in use by the allocated objects relative to the previous measurement.
- the difference in the worstcase amount of shared memory 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.

11.4.3 The meta-object references mode

In the meta-object references mode `mmstat` basically displays which objects are currently populating the shared memory.

Mmstat meta-object references mode



For this purpose it will iterate through all datatypes known to the Domain, and for each datatype it will display the following information:

- 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).

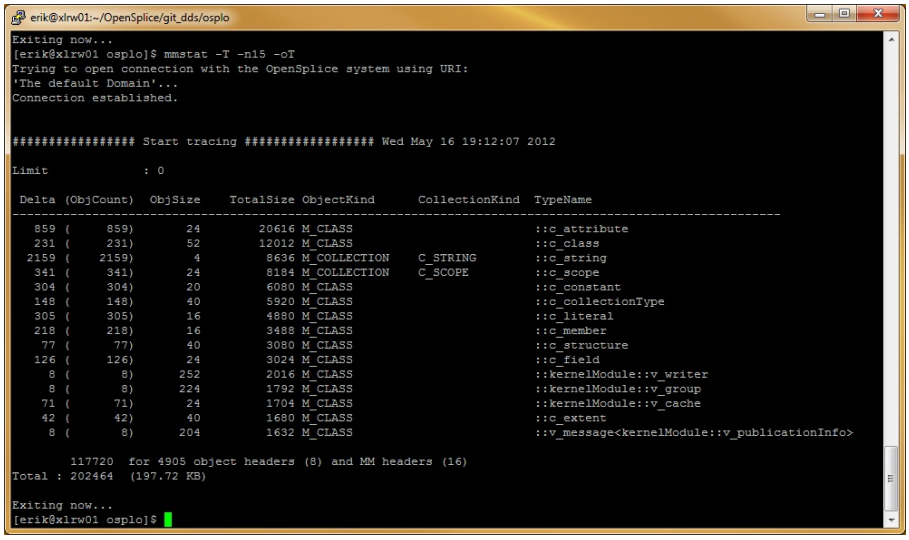
11.4.4 The meta-object references difference mode

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.

Mmstat meta-object references difference mode



12

Configuration

This section describes the various configuration elements and attributes available for Vortex OpenSplice. The configuration items should be added to an XML file and then the OSPL_URI environment variable should be set to point to the path of that XML file with the "file://" URI prefix.

- **e.g.**
  - Linux: export OSPL_URI=file://$OSPL_HOME/etc/ospl.xml
  - Windows: set OSPL_URI=file://%OSPL_HOME%\\etc\\ospl.xml

The ospl.xml file supplied with Vortex OpenSplice contains the following:

```
<OpenSplice>
   <Domain>
     <Name>ospl_sp_ddsi</Name>
     <Id>0</Id>
     <SingleProcess>true</SingleProcess>
     <Service name="ddsi2">
         <Command>ddsi2</Command>
     </Service>
     <Service name="durability">
         <Command>durability</Command>
     </Service>
     <Service name="cmsoap">
         <Command>cmsoap</Command>
     </Service>
   </Domain>
   <DDSI2Service name="ddsi2">
     <General>
         <NetworkInterfaceAddress>AUTO</NetworkInterfaceAddress>
         <AllowMulticast>true</AllowMulticast>
         <EnableMulticastLoopback>true</EnableMulticastLoopback>
         <CoexistWithNativeNetworking>false</CoexistWithNativeNetworking>
     </General>
     <Compatibility>
         <!-- see the release notes and/or the OpenSplice configurator on DDSI interoperability -
         <StandardsConformance>lax</StandardsConformance>
         <!-- the following one is necessary only for TwinOaks CoreDX DDS compatibility -->
         <!-- <ExplicitlyPublishQosSetToDefault>true</ExplicitlyPublishQosSetToDefault> -->
     </Compatibility>
   </DDSI2Service>
   <DurabilityService name="durability">
     <Network>
         <Alignment>
             <TimeAlignment>false</TimeAlignment>
             <RequestCombinePeriod>
                 <Initial>2.5</Initial>
                 <Operational>0.1</Operational>
             </RequestCombinePeriod>
         </Alignment>
         <WaitForAttachment maxWaitCount="100">
```

```
            <ServiceName>ddsi2</ServiceName>
        </WaitForAttachment>
    </Network>
    <NameSpaces>
        <NameSpace name="defaultNamespace">
            <Partition>*</Partition>
        </NameSpace>
        <Policy alignee="Initial" aligner="true" durability="Durable" nameSpace="defaultNamespace
    </NameSpaces>
  </DurabilityService>
  <TunerService name="cmsoap">
    <Server>
        <PortNr>Auto</PortNr>
    </Server>
  </TunerService>
  <Description>Stand-alone 'single-process' deployment and standard DDSI networking.</Description
</OpenSplice>
```

The tags in the XML file should be nested in the same way as they are in the table of contents in this configuration section. The nesting and numbering of the tags in the contents of this section allows you to see which elements are the parent or children of one another. For example, if you wanted to find a description of the NetworkInterfaceAddress attribute, you would first navigate to it's parent, the General element, and inside that you would find a heading for the child NetworkInterfaceAddress attribute along with a description and valid values. Some attributes may state that they are required and if so these elements must be present when the parent element is included in the XML file.

If you wanted to add a new element, say to enable security, you would navigate to the Security element of the section. This has a child element called SecurityProfile which should be nested within the Security element. Each element lists a number of occurences, this states how many times this element can appear in your XML file. The SecurityProfile element has three attributes, Name, which is required, and Cipher and CipherKey which are optional. Attributes are added within the parent element tag in the format name="value". Adding these new elements and attributes would result in the following XML:

```
<OpenSplice>
  <Domain>
     <Name>ospl_sp_ddsi</Name>
     <Id>0</Id>
  </Domain>
  <DDSI2E>
      <General>
          <NetworkInterfaceAddress>AUTO</NetworkInterfaceAddress>
          <AllowMulticast>true</AllowMulticast>
          <EnableMulticastLoopback>true</EnableMulticastLoopback>
      </General>
      <Compatibility>
          <StandardsConformance>lax</StandardsConformance>
      </Compatibility>
      <Tracing>
          <Verbosity>warning</Verbosity>
      </Tracing>
      <Security>
          <SecurityProfile Name="GlobalProfile" Cipher="blowfish" CipherKey="000000000000000000000
      </Security>
  </DDSI2E>
</OpenSplice>
```

12.1 OpenSplice

Vortex OpenSplice is highly configurable, even allowing the architecture of the DDS middleware to be chosen by the user at deployment time. Vortex OpenSplice can be configured to run using a **shared**

memory architecture, where both the DDS related administration (including the optional pluggable services) and DDS applications interface directly with shared memory. Alternatively, Vortex OpenSplice also supports a **single process** architecture, where one or more DDS applications, together with the Vortex 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 the Vortex OpenSplice Tuner tool

dbms service - providing a connection between the real-time and the enterprise domain by bridging data from DDS to DBMS and vice versa

Because of the pluggable architecture, the Vortex OpenSplice middleware can be easily configured on the fly by specifying which services to be used as well as specifying their optimal configuration for the application domain (networking parameters, durability levels, etc.). Typically, there are many nodes within a system.

The Vortex OpenSplice middleware including its services can be configured by means of easy maintainable XML-file(s).

- Full path: //OpenSplice
- Occurrences min-max: 1-1
- Child elements: Description
- Optional attributes: version

12.2 Domain

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 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 Vortex OpenSplice Domain service is requested, it will react by announcing the shutdown using the DDS 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 administration and finally terminate itself.

- Full path: //OpenSplice/Domain
- Occurrences min-max: 1-1
- Child elements: Name, Id, Role, ServiceTerminatePeriod, SingleProcess, Description, CPUAffinity, InProcessExceptionHandling, RetentionPeriod, y2038Ready

12.2.1 Name

This element specifies the name of the instantiated DDS domain. In general, it is recommended to change this name to a name that identifies the domain. If several different DDS domains are required to run simultaneously, then they all need to have their own domain name.

- Full path: //OpenSplice/Domain/Name
- Format: string
- Default value: OpenSpliceV6
- Occurrences min-max: 1-1

12.2.2 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 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 or v2.2, formal/2014-09-01, at http://www.omg.org/spec/DDSI for further information.

- Full path: //OpenSplice/Domain/Id
- Format: integer
- Default value: 0
- Valid values: 0 / 2147483646
- Occurrences min-max: 0-1

12.2.3 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 OpenSplice/NetworkService/Discovery[@Scope])

- Full path: //OpenSplice/Domain/Role
- Format: string
- Default value: DefaultRole
- Occurrences min-max: 0-1

12.2.4 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: ExpiryTime

12.2.4.1 ExpiryTime

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

Every Vortex OpenSplice 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 / -
- Occurrences min-max: 1-1
- Required attributes: update_factor

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
- Default value: 0.2
- Valid values: 0.01 / 1.0
- Required: true

12.2.5 GeneralWatchdog

This element controls the default characteristics of the Watchdog thread for all services. Individual services may overrule this default in their service specific settings. Every service has its own Watchdog thread, that is responsible for automatically renewing the lease for that service. Services that do not renew their lease in time will loose the liveliness of all their writers

- Full path: //OpenSplice/Domain/GeneralWatchdog
- Occurrences min-max: 0-1

12.2.5.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/Domain/GeneralWatchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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/Domain/GeneralWatchdog/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.5.1.1.1 priority_kind

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

- Full path: //OpenSplice/Domain/GeneralWatchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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/Domain/GeneralWatchdog/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.2.6 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 Vortex OpenSplice.

- Full path: //OpenSplice/Domain/ServiceTerminatePeriod
- Format: float
- Dimension: seconds
- Default value: 10.0
- Valid values: 0.0 / 60.0

- Occurrences min-max: 0-1

12.2.7 SingleProcess

The SingleProcess element specifies whether the Vortex OpenSplice Domain and other Vortex OpenSplice services and applications are intended to be all deployed within the same process, known in Vortex 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 Vortex OpenSplice database management instead of shared memory that would be used otherwise. If no database size or size 0 is configured 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. If the database size is configured with a value, a database will be allocated on heap with that size and the domain service will use it's own memory manager to manage that memory

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

Single Process Application [The user starts an application as a] new process. In this case, the DDS create_participant operation will implicitly start the Vortex OpenSplice Domain Service as a thread in the existing application process. The Vortex 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 Application 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 specified under Domain will only take effect when this SingleProcess attribute has a value of true.

- Full path: //OpenSplice/Domain/SingleProcess
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.2.8 Description

The content of this (optional) element is visualised by the Launcher tool and is meant to describe the configuration in a human-readable form.

- Full path: //OpenSplice/Domain/Description
- Format: string
- Occurrences min-max: 0-1

12.2.9 CPUAffinity

This (optional) element specifies a comma separated list of CPU numbers for the CPUs which opensplice, and its services should be restricted to use. (Supported on VxWorks kernel mode only)

- Full path: //OpenSplice/Domain/CPUAffinity
- Format: string
- Occurrences min-max: 0-1

12.2.10 InProcessExceptionHandling

The InProcessExceptionHandling element determines whether a process that uses Vortex OpenSplice will handle exceptions by itself and try to clean up shared resources or not. If the process itself refrains from cleaning up its resources, the splice-daemon will attempt to clean up the application shared resources asynchronously. If the splice-daemon during clean-up determines that shared resources have been left in an inconsistent state by the application, it will terminate the middleware.

Setting this option to false will make the middleware fail-safe. However, the downside of this approach is that there are cases in which the splice-daemon will decide to shut down the middleware, because it cannot determine with 100% certainty that shared resources are consistent (even if they are consistent in some cases). A potential approach could be to set this option to 'true' during development-phase and 'false' when application(s) have been deployed.

By default this configuration item is set to true.

- Full path: //OpenSplice/Domain/InProcessExceptionHandling
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.2.11 Daemon

Every domain is controlled by exactly one daemon: the Splice Daemon. The Splice Daemon configuration expects a root element named *OpenSplice/Daemon*. Within this root element, the Splice Daemon will look for several child-elements. Each of these child elements is listed and explained.

- Full path: //OpenSplice/Domain/Daemon
- Occurrences min-max: 0-1
- Child elements: Locking

12.2.11.1 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 Deamon. 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
- Default value: false
- Occurrences min-max: 0-1

12.2.11.2 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: StackSize

Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/Domain/Daemon/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

12.2.11.2.1.1 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/Domain/Daemon/Watchdog/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.2.1.2 priority_kind

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

- Full path: //OpenSplice/Domain/Daemon/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Occurrences min-max: 0-1
- Required: false

12.2.11.2.1.3 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/Domain/Daemon/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

StackSize

This element specifies the thread stacksize that will be used by the Watchdog thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/Watchdog/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.3 shmMonitor

This element controls the scheduling characteristics of the shmMonitor thread. This thread is responsible cleaning up resources that are left behind by unexpected termination of other dds services and applications.

- Full path: //OpenSplice/Domain/Daemon/shmMonitor
- Occurrences min-max: 0-1
- Child elements: StackSize

Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/Domain/Daemon/shmMonitor/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

12.2.11.3.1.1 Priority

This element specifies the thread priority that will be used by the shmMonitor 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/shmMonitor/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.3.1.2 priority_kind

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

- Full path: //OpenSplice/Domain/Daemon/shmMonitor/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute

- Occurrences min-max: 0-1
- Required: false

12.2.11.3.1.3 Class

This element specifies the thread scheduling class that will be used by the shmMonitor 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/shmMonitor/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

StackSize

This element specifies the thread stacksize that will be used by the shmMonitor thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/shmMonitor/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.4 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: StackSize

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: Priority, Class

12.2.11.4.1.1 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
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.4.1.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.2.11.4.1.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

StackSize

This element specifies the thread stacksize that will be used by the KernelManager thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/KernelManager/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.5 GarbageCollector

This element specifies the behaviour of the GarbageCollector.

The garbage collector is a safety mechanism and is responsible for reclaiming resources in case an application or remote node does not terminate properly.

- Full path: //OpenSplice/Domain/Daemon/GarbageCollector
- Occurrences min-max: 0-1
- Child elements: StackSize

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: Priority, Class

12.2.11.5.1.1 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
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.5.1.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.2.11.5.1.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default

- Occurrences min-max: 1-1

StackSize

This element specifies the thread stacksize that will be used by the GarbageCollector thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/GarbageCollector/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.6 ResendManager

Every service has its own ResendManager thread, that is responsible for attempting to resend samples in the history queue of its writers. A writer that cannot successfully deliver a sample to a local service or application in the first attempt (for example because the application or service ran out of queue space) will store that sample in its own history queue, from which the Resend manager will periodically try to re-transmit it.

- Full path: //OpenSplice/Domain/Daemon/ResendManager
- Occurrences min-max: 0-1
- Child elements: StackSize

Scheduling

This element specifies the type of OS scheduling class used by the thread that does local resends for the builtin participant.

- Full path: //OpenSplice/Domain/Daemon/ResendManager/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

12.2.11.6.1.1 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
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.6.1.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.2.11.6.1.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

StackSize

This element specifies the thread stacksize that will be used by the ResendManager thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/ResendManager/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.7 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: ExpiryTime, StackSize
- Optional attributes: transport_priority

transport_priority

This attribute controls the transport priority QoS setting (in seconds) that is only used by the Splice Daemom for for sending its heartbeats.

- Full path: //OpenSplice/Domain/Daemon/Heartbeat[@transport_priority]

- Format: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

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: Priority, Class

12.2.11.7.2.1 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.11.7.2.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.2.11.7.2.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

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 need of your system with respect to these aspects.

- Full path: //OpenSplice/Domain/Daemon/Heartbeat/ExpiryTime
- Default value: 4.0
- Valid values: 0.2 / -
- Occurrences min-max: 0-1
- Required attributes: update_factor

12.2.11.7.3.1 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 then 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]
- Default value: 0.25
- Valid values: 0.1 / 0.9
- Required: true

StackSize

This element specifies the thread stacksize that will be used by the Heartbeat thread. By default this is 512 Kb.

- Full path: //OpenSplice/Domain/Daemon/Heartbeat/StackSize
- Format: integer
- Dimension: bytes
- Default value: 524288
- Occurrences min-max: 0-1

12.2.11.8 Tracing

This element controls the amount and type of information that is written into the tracing log by the Splice Daemon. This is useful to track the Durability Service during application development. In the runtime system it should be turned off.

- Full path: //OpenSplice/Domain/Daemon/Tracing
- Occurrences min-max: 0-1
- Child elements: OutputFile, Timestamps, Verbosity
- Optional attributes: synchronous

synchronous

- Full path: //OpenSplice/Domain/Daemon/Tracing[@synchronous]
- Format: boolean
- Default value: FALSE
- Required: false

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/Domain/Daemon/Tracing/OutputFile
- Format: string
- Default value: durability.log
- Occurrences min-max: 0-1

Timestamps

This element specifies whether the logging must contain timestamps.

- Full path: //OpenSplice/Domain/Daemon/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

12.2.11.8.3.1 absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

- Full path: //OpenSplice/Domain/Daemon/Tracing/Timestamps[@absolute]
- Format: boolean
- Default value: true
- Required: false

Verbosity

This element specifies the verbosity level of the loggin information. The higher the level, the more (detailed) information will be logged.

- Full path: //OpenSplice/Domain/Daemon/Tracing/Verbosity
- Format: enumeration
- Default value: INFO
- Valid values: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
- Occurrences min-max: 0-1

12.2.12 Database

The Database element drives how the administration within the federation is managed. In federated deployment this administration is managed in a shared memory segment and in single-process deployment it is managed on heap.

- Full path: //OpenSplice/Domain/Database
- Occurrences min-max: 0-1
- Child elements: Size, Threshold, Address, Locking

12.2.12.1 Size

This element specifies the maximum size of the memory segment for maintaining the nodal administration. The memory management behaviour depends on the deployment mode (federated vs single-process, driven by the //OpenSplice/Domain/SingleProcess element) and the size configured within this element:

- **Single process mode In this mode the database is managed on heap.** If the size is set to zero, the heap is managed by the operating system memory manager and limited to the operating system limit. If a non-zero size is configured, the database is still allocated on heap, but the domain service pre-allocates the entire size at once and will use its own memory manager to manage this memory segment. The default value in this mode is 0 bytes.
- **Federated mode In this mode, the database is managed in shared memory segment** and the domain service will use its own memory manager to manage the shared memory segment. The default value in this mode is 10 Mbytes.

Change this value if your system requires more memory than the default. Please note that the operating system should be configured support the requested size. On most platforms you need 'root' 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
- Dimension: bytes
- Default value: 10485760
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.2.12.2 Threshold

This element specifies the threshold size used by Vortex OpenSplice. Whenever there is less free shared memory then indicated by the threshold then no new allocations will be allowed within shared memory. Services are allowed to continue allocating shared memory until less then 50% of the threshold value is available. It is strongly discouraged to configure a threshold value of less then the default value, but for some embedded systems it might be needed as only limited memory 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.

- Full path: //OpenSplice/Domain/Database/Threshold
- Dimension: bytes
- Default value: 1048576
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.2.12.3 Address

This element specifies the start address where the nodal shared administration is mapped into the virtual memory for each process that attaches to the current Domain. 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.

default values per platform:

- 00x20000000 (Linux)
- 00x140000000 (Linux 64-bit)
- 00x40000000 (Windows)
- 00x140000000 (Windows 64-bit)
- 00xA0000000 (Solaris)
- 00xA0000000 (AIX5.3)
- 00x0 (VxWorks 5.5.1)
- 00x60000000 (VxWorks 6.x)
- 00x20000000 (Integrity)
- 00x20000000 (LynxOS)
- 00x140000000 (LynxOS 64-bit)
- 00x0 (PikeOS)
- 00x60000000 (QXN)
- 00x0 (RTEMS)
- 00x00220000 (Windows CE)

- Full path: //OpenSplice/Domain/Database/Address
- Format: string
- Default value: 0x40000000
- Valid values: 0x0 / -
- Occurrences min-max: 0-1

12.2.12.4 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
- Default value: Default
- Valid values: True, False, Default
- Occurrences min-max: 0-1

12.2.13 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: StackSize

12.2.13.1 StackSize

This element specifies the stack size of the listener thread.

- Full path: //OpenSplice/Domain/Listeners/StackSize
- Default value: 128000
- Valid values: 64000 / -
- Occurrences min-max: 1-1

12.2.14 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: 0-*
- Child elements: Command, MemoryPoolSize, HeapSize, StackSize, Configuration, Locking, FailureAction
- Required attributes: name
- Optional attributes: enabled

12.2.14.1 name

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

In the Vortex OpenSplice 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

- Default value: durability
- Required: true

12.2.14.2 enabled

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

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

- Full path: //OpenSplice/Domain/Service[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.2.14.3 Command

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

In shared memory mode, Command element specifies the name of the actual service executable or a script to launch this service (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 Vortex 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
- Default value: durability
- Occurrences min-max: 1-1

12.2.14.4 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. 0xa00000 for spliced, 0x280000 for durability, 0x280000 for networking, 0x100000 for cmsoap

- Full path: //OpenSplice/Domain/Service/MemoryPoolSize
- Format: string
- Default value: 0
- Occurrences min-max: 0-1

12.2.14.5 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. 0x800000 for spliced, 0x240000 for durability, 0x240000 for networking, 0x200000 for cmsoap

- Full path: //OpenSplice/Domain/Service/HeapSize
- Format: string
- Default value: 0
- Occurrences min-max: 0-1

12.2.14.6 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 0x10000 for spliced 0x10000 for durability 0x10000 for networking 0x10000 for cmsoap

- Full path: //OpenSplice/Domain/Service/StackSize
- Format: string
- Default value: 0
- Occurrences min-max: 0-1

12.2.14.7 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
- Default value: ${OSPL_URI}
- Occurrences min-max: 0-1

12.2.14.8 Scheduling

This element specifies the type of OS scheduling class will be used by the Domain service to create the service process. Services can only be started within the scheduling classes that are supported by the underlying operating system.

- Full path: //OpenSplice/Domain/Service/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.2.14.8.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.2.14.9 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
- Default value: false
- Occurrences min-max: 0-1

12.2.14.10 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 DDS 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 DomainService 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 Vortex OpenSplice services including the Domain service (for the current DDS Domain on this computing node).

- Full path: //OpenSplice/Domain/Service/FailureAction
- Format: enumeration
- Default value: skip
- Valid values: kill, restart, skip, systemhalt
- Occurrences min-max: 0-1

12.2.15 GIDKey

- Full path: //OpenSplice/Domain/GIDKey
- Occurrences min-max: 0-*
- Required attributes: groups

12.2.15.1 groups

This attribute specifies the namespace by which the corresponding groups will use the writer instance gid as key so that data from different writers don merge as historical data for late joiners.

- Full path: //OpenSplice/Domain/GIDKey[@groups]
- Format: string
- Default value: n/a
- Required: true

12.2.16 Application

When in the single process deployment 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 Vortex OpenSplice 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.>/p> **Note that Applications only take effect when the single process configuration is enabled by way of the SingleProcess element.**

- Full path: //OpenSplice/Domain/Application
- Occurrences min-max: 0-*
- Child elements: Command, Arguments, Library
- Required attributes: name
- Optional attributes: enabled

12.2.16.1 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
- Default value: n/a
- Required: true

12.2.16.2 enabled

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

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

- Full path: //OpenSplice/Domain/Application[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.2.16.3 Command

Command is the name of both the entry point function to 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.

For example, if Command is "HelloWorld", Vortex OpenSplice 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 is able to override the name of the library by using the Application's Library attribute

The shared library is located by way of the current working directory, or LD_LIBRARY_PATH for Unix systems, and PATH for 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
- Default value: application1

- Occurrences min-max: 1-1

12.2.16.4 Arguments

This optional attribute allows 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", Vortex OpenSplice will invoke the HelloWorld function with the argc = 3 and argv = {"HelloWorld", "arg1", "arg2"}

- Full path: //OpenSplice/Domain/Application/Arguments
- Format: string
- Occurrences min-max: 0-1

12.2.16.5 Library

This optional attribute allows the user to override the name of the shared library if it is different to the name of the entry point specified by Command.

The shared library is located by way of the current working directory, or LD_LIBRARY_PATH for Unix systems, and PATH for Windows systems.

- Full path: //OpenSplice/Domain/Application/Library
- Format: string
- Occurrences min-max: 0-1

12.2.17 BuiltinTopics

This element specifies the granularity of the builtin topics.

- Full path: //OpenSplice/Domain/BuiltinTopics
- Occurrences min-max: 0-1
- Required attributes: enabled
- Optional attributes: logfile

12.2.17.1 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 builtin topics, regardless of the value specified here.

- Full path: //OpenSplice/Domain/BuiltinTopics[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.2.17.2 logfile

This attribute specifies the log file for entity lifecycle messages. When a logfile is specified the service will open the logfile and write a corresponding log line for each update of discovered entities.

- Full path: //OpenSplice/Domain/BuiltinTopics[@logfile]

- Format: string
- Default value: builtin.log
- Required: false

12.2.18 PriorityInheritance

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

- Full path: //OpenSplice/Domain/PriorityInheritance
- Occurrences min-max: 0-1
- Required attributes: enabled

12.2.18.1 enabled

This attribute enables or disables priority inheritance for mutexes, if that is supported by the underlying Operating System. If the parent Element PriorityInheritance is not specified (does not exist), then the 'default' value of this attribute is false.

- Full path: //OpenSplice/Domain/PriorityInheritance[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.2.19 Report

The Report element controls some aspects of the Vortex OpenSplice domain logging functionality.

- Full path: //OpenSplice/Domain/Report
- Occurrences min-max: 0-1
- Optional attributes: append, verbosity

12.2.19.1 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: boolean
- Default value: true
- Required: false

12.2.19.2 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: enumeration
- Default value: INFO
- Valid values: DEBUG, INFO, WARNING, API_INFO, ERROR, CRITICAL, FATAL, REPAIRED, NONE
- Required: false

12.2.20 Statistics

This element specifies the policies regarding statistics. Various statistics can be generated by Vortex OpenSplice 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

12.2.20.1 Category

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

- Full path: //OpenSplice/Domain/Statistics/Category
- Occurrences min-max: 0-*
- Required attributes: name
- Optional attributes: enabled

enabled

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

- Full path: //OpenSplice/Domain/Statistics/Category[@enabled]
- Format: boolean
- Default value: true
- Required: false

name

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

- Full path: //OpenSplice/Domain/Statistics/Category[@name]
- Format: enumeration

- Default value: reader
- Valid values: durability, reader, writer, networking
- Required: true

12.2.21 RetentionPeriod

This element specifies how long the administration for unregistered instances is retained in both readers and the durability service before it is definitively removed. (For the durability service this time is added to the service_cleanup_delay configured for each TopicQos.) By default unregistered instances are retained for 500 ms prior to removal, to avoid the revival of 'forgotten' instances when receiving delayed samples written prior to the instance's unregistration. This value should only be decreased when the expected lifetime of an instance is extremely short while the instance generation frequency is extremely high, to avoid running out of resources. The value should be increased when you know you can expect out-of-order deliveries with a maximum delay higher than the currently configured RetentionPeriod.

- Full path: //OpenSplice/Domain/RetentionPeriod
- Format: integer
- Default value: 500
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.2.22 ReportPlugin

This Tag specifies user defined report functionality to be used by the domain. All services and applications will load a user provides report library that will implement the report plugin interface. The report interface consists of three operations; initialize, report and finalize.

- Full path: //OpenSplice/Domain/ReportPlugin
- Occurrences min-max: 0-*
- Child elements: SuppressDefaultLogs, ServicesOnly

12.2.22.1 Library

This tag specifies the library to be loaded.

- Full path: //OpenSplice/Domain/ReportPlugin/Library
- Occurrences min-max: 1-1
- Required attributes: file_name

file_name

This attribute specifies the library to be loaded.

- Full path: //OpenSplice/Domain/ReportPlugin/Library[@file_name]
- Format: string
- Default value: n/a
- Required: true

12.2.22.2 Initialize

This tag 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
- Required attributes: symbol_name
- Optional attributes: argument

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 it is 0 then the operation was successful. If it is not 0 then there was an error and details of the error and the result value are reported to the Vortex OpenSplice 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
- Default value: n/a
- Required: true

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
- Default value: “”
- Required: false

12.2.22.3 Report

This tag 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
- Required attributes: symbol_name

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 function is invoked slightly differently for applications using the DDS API and Vortex OpenSplice services. Vortex OpenSplice services invoke the function once for every report. For applications using the DDS API the function is invoked once per API call, in which case the XML is extended by one or more DETAIL elements. 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 it is 0 then the operation was successful. If it is not 0 then there was an error and details of the error and the result value are reported to the Vortex OpenSplice 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 when invoked by a service:

<ERROR>

<DESCRIPTION>The object “my_topic” not found</DESCRIPTION>

<CONTEXT>c_base::resolve</CONTEXT>

<FILE>c_base.c</FILE>

<LINE>1234</LINE>

<CODE>0</CODE>

<PROCESS>

<ID>1234</ID>

<NAME>ProcessName</NAME>

</PROCESS>

<THREAD>

<ID>1234</ID>

<NAME>ThreadName</NAME>

</THREAD>

<COMMERCIAL>6.5</COMMERCIAL>

<REVISION>...</REVISION>

</ERROR>

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

<ERROR>

<DESCRIPTION>Operation failed.</DESCRIPTION>

<CONTEXT>c_base::resolve</CONTEXT>

<FILE>c_base.c</FILE>

<LINE>1234</LINE>

<CODE>0</CODE>

<PROCESS>

<ID>1234</ID>

<NAME>ProcessName</NAME>

</PROCESS>

<THREAD>

<ID>1234</ID>

<NAME>ThreadName</NAME>

</THREAD>

<COMMERCIAL>6.5</COMMERCIAL>

<REVISION>...</REVISION>

<DETAIL>

<REPORT>The object “my_topic” not found.</REPORT>

<INTERNALS>...</INTERNALS>

</DETAIL>

<!– optionally more DETAIL elements –>

</ERROR>

- Full path: //OpenSplice/Domain/ReportPlugin/Report[@symbol_name]
- Format: string
- Default value: n/a
- Required: true

12.2.22.4 TypedReport

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

- Full path: //OpenSplice/Domain/ReportPlugin/TypedReport
- Occurrences min-max: 0-1
- Required attributes: symbol_name

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 it is 0 then the operation was successful. If it is not 0 then there was an error and details of the error and the result value are reported to the Vortex OpenSplice default report service.

- Full path: //OpenSplice/Domain/ReportPlugin/TypedReport[@symbol_name]
- Format: string
- Default value: n/a
- Required: true

12.2.22.5 Finalize

This tag 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
- Required attributes: symbol_name

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 it is 0 then the operation was successful. If it is not 0 then there was an error and details of the error and the result value are reported to the Vortex OpenSplice default report service. The context parameter is a reference to the plugin-specific data retrieved from the initialize operation.

- Full path: //OpenSplice/Domain/ReportPlugin/Finalize[@symbol_name]
- Format: string
- Default value: n/a
- Required: true

12.2.22.6 SuppressDefaultLogs

This attribute specifies whether the default error and info report logs are to be produced when a user Report Plugin has been defined. If registration of the Report Plugin fails the default error and info logs will not be suppressed regardless of the value of this attribute

- Full path: //OpenSplice/Domain/ReportPlugin/SuppressDefaultLogs
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.2.22.7 ServicesOnly

This attribute specifies whether the log plug-in is to be effective only for processes that are exclusively Vortex OpenSplice services. If this value is true then the plug-in will not be used for user applications and/or Vortex OpenSplice services collocated with user applications in single process mode.

- Full path: //OpenSplice/Domain/ReportPlugin/ServicesOnly
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.2.23 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

12.2.23.1 MaxSamples

This configuration tag allows for the specification of certain characteristics of the max samples resource limit that will be applied throughout the domain

- Full path: //OpenSplice/Domain/ResourceLimits/MaxSamples
- Occurrences min-max: 0-1
- Child elements: WarnAt

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
- Default value: 5000
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.2.23.2 MaxInstances

This configuration tag allows for the specification of certain characteristics of the max instances resource limit that will be applied throughout the domain

- Full path: //OpenSplice/Domain/ResourceLimits/MaxInstances
- Occurrences min-max: 0-1
- Child elements: WarnAt

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
- Default value: 5000
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.2.23.3 MaxSamplesPerInstance

This configuration tag allows for the specification of certain characteristics of the max samples per instance resource limit that will be applied throughout the domain

- Full path: //OpenSplice/Domain/ResourceLimits/MaxSamplesPerInstance
- Occurrences min-max: 0-1
- Child elements: WarnAt

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
- Default value: 5000
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.2.24 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-*
- Required attributes: partition_expression, access_mode

12.2.24.1 partition_expression

This attribute specifies the partitions by name. The wildcards '*' and '?' are allowed and the specified access rights will be applied to all matching partitions. In case partitions match multiple rules the rules will be applied in sequence of declaration.

- Full path: //OpenSplice/Domain/PartitionAccess[@partition_expression]
- Format: string
- Default value: *
- Required: true

12.2.24.2 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: enumeration
- Default value: readwrite
- Valid values: none, read, write, readwrite
- Required: true

12.2.25 SystemId

This configures the generation of the unique System IDs

- Full path: //OpenSplice/Domain/SystemId
- Occurrences min-max: 0-1
- Child elements: UserEntropy

12.2.25.1 Range

This configures the range of the generated System IDs

- Full path: //OpenSplice/Domain/SystemId/Range
- Occurrences min-max: 0-1
- Optional attributes: min, max

min

This attribute specifies the minimum allowed System ID. In addition to being within range, it must be less than or equal to the "max" attribute.

- Full path: //OpenSplice/Domain/SystemId/Range[@min]
- Format: integer
- Default value: 1
- Valid values: 1 / 2147483647
- Required: false

max

This attribute specifies the maximum allowed System ID. In addition to being within range, it must be greater than or equal to the "min" attribute.

- Full path: //OpenSplice/Domain/SystemId/Range[@max]
- Format: integer

- Default value: 2147483647
- Valid values: 1 / 2147483647
- Required: false

12.2.25.2 UserEntropy

This attribute specifies a string that is used as an additional source of entropy in the System ID generation. The string is not interpreted.

- Full path: //OpenSplice/Domain/SystemId/UserEntropy
- Format: string
- Occurrences min-max: 0-1

12.2.26 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

- Full path: //OpenSplice/Domain/TopicAccess
- Occurrences min-max: 0-*
- Required attributes: topic_expression, access_mode

12.2.26.1 topic_expression

This attribute specifies the topics by name. The wildcards '*' and '?' are allowed and the specified access rights will be applied to all matching topics. In case topics match multiple rules the rules will be applied in sequence of declaration.

- Full path: //OpenSplice/Domain/TopicAccess[@topic_expression]
- Format: string
- Default value: *
- Required: true

12.2.26.2 access_mode

This attribute specifies the access rights that will be applied to the specified topics. The following values are applicable:

  - none
  - read
  - write
  - readwrite

- Full path: //OpenSplice/Domain/TopicAccess[@access_mode]
- Format: enumeration

- Default value: readwrite
- Valid values: none, read, write, readwrite
- Required: true

12.2.27 UserClock

The UserClock Service allows you to plug in a custom clock library, allowing Vortex OpenSplice to read the time from an external clock source. It expects a root element named *OpenSplice/Domain/UserClock*. Within this root element, the userclock will look for several child-elements. Each of these is listed and explained.

- Full path: //OpenSplice/Domain/UserClock
- Occurrences min-max: 0-1
- Child elements: UserClockModule, UserClockStart, UserClockStop, UserClockQuery
- Optional attributes: y2038Ready

12.2.27.1 y2038Ready

This element specifies whether or not the registered user clock is returning a 64-bit seconds field.

Default this setting is following the y2038Ready setting on domain level. (default /domain/y2038Ready is FALSE) The user is able to make some mixed environments by using this configuration option. For example a 64-bit user clock can already be implemented (by setting this option to true) while the node must be compatible with older versions, so domain/y2038Ready can not be set. A warning is printed when the Domain/y2038Ready element is true and the Domain/UserClock/y2038Ready attribute is false. The 64-bit layout returned by the userclock must be:

```
struct dds_userclock_t {
    os_int64 seconds;
    os_int32 nanoseconds;
};
```

See 'Time stamps and year 2038 limit' for more background information.

- Full path: //OpenSplice/Domain/UserClock[@y2038Ready]
- Format: boolean
- Default value: False
- Required: false

12.2.27.2 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/UserClock/UserClockModule
- Format: string
- Occurrences min-max: 1-1

12.2.27.3 UserClockStart

This element specifies if the user clock requires a start function to be called when the process first creates a participant. This element 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/UserClock/UserClockStart
- Format: string
- Default value: clockStart
- Occurrences min-max: 0-1
- Required attributes: enabled

enabled

This attribute specifies if the start function is enabled and should be used.

- Full path: //OpenSplice/Domain/UserClock/UserClockStart[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.2.27.4 UserClockStop

This element specifies if the user clock requires a stop function to be called when the process deletes the last participant. 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/UserClock/UserClockStop
- Format: string
- Default value: clockStop
- Occurrences min-max: 0-1
- Required attributes: enabled

enabled

This attribute specifies if the stop function is enabled and should be used.

- Full path: //OpenSplice/Domain/UserClock/UserClockStop[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.2.27.5 UserClockQuery

This element specifies the clock query function. 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*.

- Full path: //OpenSplice/Domain/UserClock/UserClockQuery
- Format: string
- Default value: clockGet
- Occurrences min-max: 0-1
- Required attributes: enabled

enabled

This attribute specifies if the query function is enabled and should be used.

- Full path: //OpenSplice/Domain/UserClock/UserClockQuery[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.2.28 DurablePolicies

There are two ways for a late joining reader to retrieve historical data as a result of calling <tt>wait_for_historical_data()</tt> or <tt>DDS_wait_for_historical_data()</tt>.

- **The first way is to configure a local durability service** that is responsible for the alignment of historical data (see //OpenSplice/DurabilityService). In this case the local durability service will provide historical data to interested readers on the local federations.
- **The second way is NOT to run a local durability service, but** to request data from a durability service on a remote federation using the client-durability feature. This makes sense in environments where running a full-fledged durability service is not possible or unwanted, e.g., due to resource limitations.

The element //OpenSplice/Domain/DurablePolicies specifies how historical data that is retrieved from a remote durability federation using the client-durability feature is handled locally. More specifically, this element allows the user to indicate whether historical data that is requested by a late joining reader using the client-durability feature will be made available to the requesting reader only, or to all readers in the federation. The behaviour can be specified per partition/topic combination in the //OpenSplice/Domain/DurablePolicies/Policy elements.

In most situations it is sufficient to either configure a local durability service to acquire historical data, or to use the client-durability feature, but not both. However, it is not forbidden to use both methods concurrently. In that case it is advised to configure the system in such a way that historical data for a particular partition/topic combination is either provided using method 1 or 2. In the event that historical data for the same partition/topic combination can be requested via both methods, methods 1 will be used.

- Full path: //OpenSplice/Domain/DurablePolicies
- Occurrences min-max: 0-1

12.2.28.1 Policy

The policy specifies whether historical data that matches the pattern specified in the //OpenSplice/Domain/DurablePolicies/Policy[@obtain] attribute is cached or not. Caching means that the same data is available for late joining readers on the local federation.

Multiple policies can be specified. The order of these policies is important: the first policy that matches (from top to bottom) will be applied.

If no policy is specified then the default policy is applied. The default policy is to deliver requested historical data to all readers.

- Full path: //OpenSplice/Domain/DurablePolicies/Policy
- Occurrences min-max: 0-*
- Required attributes: obtain
- Optional attributes: cache

obtain

This element specifies the pattern of the partition/topic for which the policy must be applied. The default is *.*.

- Full path: //OpenSplice/Domain/DurablePolicies/Policy[@obtain]
- Format: string
- Default value: .
- Required: true

cache

This element specifies whether or not to cache historical data. If set to TRUE historical data is cached in the local federation for potential other late joiners in the same federation. In this case historical data will only be aligned once for the federation, no matter how many readers are interested. This saves bandwidth and CPU when more than one readers are expected with (partly) the same interest. The downside is that caching historical data will require additional memory even after the reader has taken all its data. If set to FALSE data is not cached.

The recommended setting is TRUE when more than one readers for the data are expected and the cost of extra memory can be afforded. In all other circumstances FALSE should be used. The default is TRUE.

- Full path: //OpenSplice/Domain/DurablePolicies/Policy[@cache]
- Format: boolean
- Default value: True
- Required: false

12.2.29 y2038Ready

The y2038Ready element determines that Vortex OpenSplice will send timestamps to other nodes in a 64-bit format which is capable to contain time stamps beyond the year 2038.

Setting this option to true will force the middleware to communicate with time stamps able to go beyond 2038. Other nodes before version 6.7 are not able to handle these new time stamps and will no longer interact correctly with the node configured with this option.

From version 6.7 the middleware is internally prepared and calculating with time stamps capable with timestamps beyond 2038. From version 6.7 nodes are able to detect the timestamps containing time beyond 2038 and the older time format. So these nodes will interact correctly with all other nodes. In a mixed environment containing versions before 6.7, communication must be in the old timestamp format, so this configuration option can not be set. In a mixed environment with only versions 6.7 and higher, this configuration option may be set. The nodes with this configuration will send timestamps capable beyond 2038. All other nodes will be able to interpret these values correctly.

See ‘ time stamps and year 2038 limit’ for more background information.

By default this configuration item is set to false.

- Full path: //OpenSplice/Domain/y2038Ready
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.2.30 Filters

Domain Filters are expressions that are applied on the local Domain Service meaning that it affects all local readers including services, i.e. filtered out data will not exist on the node. By using filters the amount of data that is being send or received can be reduced.

Different filters can be specified for different partition-topic combinations using the <Filter>-elements.

Note: that when a filter is set on a partition topic combination the Durability Service can no longer be master for the namespace that contains the given partition-topic combination as it is incomplete by definition.

- Full path: //OpenSplice/Domain/Filters
- Occurrences min-max: 0-1

12.2.30.1 Filter

This element specifies the filter that is used. The <PartitionTopic>-child elements specify the partition-topic combinations to which the filter expression is applied.

If no filter is defined for a particular partition-topic then all data will be send for this partition-topic combination. If multiple filters are defined then the first one that matches will be applied.

- Full path: //OpenSplice/Domain/Filters/Filter
- Occurrences min-max: 0-*
- Child elements: PartitionTopic
- Required attributes: content

content

This attribute specifies the expression that is used for filtering. The expression is a string that is used as-is without any parameters. The filter expression is essentially the where-clauses of an sql expression. When data is requested for matching partition-topics only the data that matches the filter expression will be aligned. The content attribute defaults to the empty string, which means that no filter is applied at all.

The following escape sequence can be used in expressions: < (less than), > (greater than), " (double quote) ' (apostrophe) and & (ampersand).

Examples of expressions are:

- x=1 or x=2
- (id<10) and (name="test")

If an invalid filter expression is provided an error will be logged in ospl-error.log.

- Full path: //OpenSplice/Domain/Filters/Filter[@content]
- Format: string

- Default value: n/a
- Required: true

PartitionTopic

This element specifies the partition-topics to which the filter expression is applied. A partition-topic expression may contain the wildcards '*' to match any number of characters and '?' to match one single character.

- Full path: //OpenSplice/Domain/Filters/Filter/PartitionTopic
- Format: string
- Default value: .
- Occurrences min-max: 0-*

12.3 DurabilityService

The responsibilities of the durability service are to realize the durable properties of data in an Vortex 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: 0-1
- Required attributes: name

12.3.1 name

This attribute identifies the configuration for the Durability service. Multiple Durability 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 *OpenSplice/Domain/Service[@name]* in the configuration of the DomainService.

- Full path: //OpenSplice/DurabilityService[@name]
- Format: string
- Default value: durability
- Required: true

12.3.2 ClientDurability

Client-durability is a feature that allows clients of this durability service to acquire historical data from a remote durability service without having to run their own durability service. This element controls the characteristics of the client-durability feature. When enabled, this durability server will be able to responds to requests for historical data from such clients. If the *OpenSplice/DurabilityService/ClientDurability* element is not provided, then the durability service will not respond to historical data requests from clients.

- Full path: //OpenSplice/DurabilityService/ClientDurability
- Occurrences min-max: 0-1
- Optional attributes: enabled

12.3.2.1 enabled

This attribute enables or disables the ability of the durability service to respond to requests for historical data from clients. When the *OpenSplice/DurabilityService/ClientDurability[@enabled]* attribute is not provided then it is assumed to be TRUE.

- Full path: //OpenSplice/DurabilityService/ClientDurability[@enabled]
- Format: boolean
- Default value: TRUE
- Required: false

12.3.2.2 EntityNames

This element specifies the names of the various entities used by the client-durability feature of this DurabilityService. The names specified here will be displayed in the Vortex OpenSplice Tuner when viewing the DurabilityService.

- Full path: //OpenSplice/DurabilityService/ClientDurability/EntityNames
- Occurrences min-max: 0-1
- Child elements: Publisher, Subscriber, Partition

Publisher

This element specifies the name of the client-durability publisher.

- Full path: //OpenSplice/DurabilityService/ClientDurability/EntityNames/Publisher
- Format: string
- Default value: durabilityPublisher
- Occurrences min-max: 0-1

Subscriber

This element specifies the name of the client-durability subscriber.

- Full path: //OpenSplice/DurabilityService/ClientDurability/EntityNames/Subscriber
- Format: string
- Default value: durabilitySubscriber
- Occurrences min-max: 0-1

Partition

This element specifies the name of the partition used for client-durability. The default is the same partition as the partition specified for durability (see OpenSplice/DurabilityService/EntityNames/Partition)

- Full path: //OpenSplice/DurabilityService/ClientDurability/EntityNames/Partition
- Format: string
- Default value: durabilityPartition
- Occurrences min-max: 0-1

12.3.3 Watchdog

This element controls the characteristics of the Watchdog thread.

- Full path: //OpenSplice/DurabilityService/Watchdog
- Occurrences min-max: 0-1
- Optional attributes: deadlockDetection

12.3.3.1 deadlockDetection

This attribute drives whether the Watchdog will check for deadlocks and refrain from updating its lease and heartbeat in case one or more of its threads do not assert their liveliness. Typically this should not be enabled, but it can be helpful to ensure certain responsiveness of the durability service and the detection of potential deadlocks.

- Full path: //OpenSplice/DurabilityService/Watchdog[@deadlockDetection]
- Format: boolean
- Default value: False
- Required: false

12.3.3.2 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DurabilityService/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.3.3.2.1.1 priority_kind

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

- Full path: //OpenSplice/DurabilityService/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.3.4 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 Networking 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 behavior of the communication between Durability services on different computing nodes on network level. These settings only apply when the Networking service is active.

- Full path: //OpenSplice/DurabilityService/Network
- Occurrences min-max: 0-1
- Child elements: InitialDiscoveryPeriod
- Optional attributes: latency_budget, transport_priority

12.3.4.1 latency_budget

This attribute controls the latency_budget QoS setting that is used by the Durability service for its communication with other Durability services.

It specfies the maximum acceptable delay (in seconds) from the time the data is written until the data is insterted in the cache of the receiving Durability service(s) and the receiver is notified of the fact. The default value is zero, indicating the delay should be minimized.

- Full path: //OpenSplice/DurabilityService/Network[@latency_budget]
- Dimension: seconds
- Default value: 0.0
- Valid values: 0.0 / -
- Required: false

12.3.4.2 transport_priority

This attribute controls the transport priority QoS setting that is used by the Durability service for its communication with other Durability services.

It indicates the importance of the communication of the Durability service with other Durability services in the system. The transport priority specified here will be interpreted by the Networking service and should be used to differentiate the priority between communication of user applications and communication of the Durability service.

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: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

12.3.4.3 Heartbeat

During startup and at runtime, the network topology can change dynamically. This happens when Vortex 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 nodes that are leaving, the Durability service uses a hearbeat 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: ExpiryTime
- Optional attributes: latency_budget, transport_priority

latency_budget

This attribute controls the latency budget QoS setting 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]
- Default value: 0.0
- Valid values: 0.0 / -
- Required: false

transport_priority

This attribute controls the transport priority QoS setting (in seconds) that is only used by the Durability service for for sending its heartbeats. It overrules the value of the *DurabilityService/Network[@transport_priorrity]*.

- Full path: //OpenSplice/DurabilityService/Network/Heartbeat[@transport_priority]
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

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: Priority, Class

12.3.4.4.3.3.1 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.3.4.4.3.3.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.3.4.4.3.3.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

ExpiryTime

This element specifies the maximum amount of time(in seconds) 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 networking traffic and overhead but also to less responsiveness with respect to the liveliness of a Durability service. Change this value according to the need of your system with respect to these aspects.

- Full path: //OpenSplice/DurabilityService/Network/Heartbeat/ExpiryTime
- Default value: 10.0
- Valid values: 0.2 / -
- Occurrences min-max: 1-1
- Required attributes: update_factor

12.3.4.3.4.1 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 then 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]
- Default value: 0.2
- Valid values: 0.1 / 0.9
- Required: true

12.3.4.4 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. 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. .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'.

- Full path: //OpenSplice/DurabilityService/Network/InitialDiscoveryPeriod
- Dimension: seconds
- Default value: 3.0
- Valid values: 0.1 / 10.0
- Occurrences min-max: 0-1

12.3.4.5 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: TimeAlignment, TimeToWaitForAligner
- Optional attributes: latency_budget, transport_priority

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 OpenSplice/DurabilityService/Network[@latency_budget].

- Full path: //OpenSplice/DurabilityService/Network/Alignment[@latency_budget]
- Default value: 0.0
- Valid values: 0.0 / -
- Required: false

transport_priority

This attribute specifies the transport priority QoS setting that is used by the Durability service for the alignment of data. It overrules the value of the *DurabilityService/Network[@transport_priorrity]* for the alignment of data only.

- Full path: //OpenSplice/DurabilityService/Network/Alignment[@transport_priority]
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

TimeAlignment

This attribute specifies whether time on all nodes 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 for possible timing gaps between different nodes in the domain.

When using DDSI2(e) networking service it is strongly recommended to set this value to 'false'. The asynchronous nature of the DDSI2 discovery protocol in combination with TimeAlignment could lead to a gap in transient/persistent data when Durability alignment and DDSI2 discovery coincide, which is normal behavior.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/TimeAlignment
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

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: Priority, Class

12.3.4.5.4.1 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.3.4.5.4.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.3.4.5.4.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

AligneeScheduling

This element specifies the scheduling parameters used to control the thread that makes sure the local node becomes and stays aligned.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/AligneeScheduling
- Occurrences min-max: 0-1

- Child elements: Priority, Class

12.3.4.5.5.1 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.3.4.5.5.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.3.4.5.5.3 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

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 networking 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 Vortex OpenSplice is started at the same time with more than two computing nodes.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/RequestCombinePeriod
- Occurrences min-max: 0-1
- Child elements: Initial, Operational

12.3.4.5.6.1 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
- Default value: 0.5
- Valid values: 0.01 / 5.0
- Occurrences min-max: 0-1

12.3.4.5.6.2 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
- Default value: 0.01
- Valid values: 0.01 / 5.0
- Occurrences min-max: 0-1

Partition

This tag specified an inter durability communication partition with specific qos settings. Alignment between durability services managed through this partition. In case multiple partitions are defined the partition with the highest alignment_priority that can provide the requested data will be used as data source. By default (no Partition specified) the durability services will use an internal partition that inherits the default qos policies and has the default alignment_priority.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/Partition
- Occurrences min-max: 0-*
- Required attributes: Name
- Optional attributes: alignment_priority, latency_budget, transport_priority

12.3.4.5.7.1 Name

The name of the partition to use for alignment.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/Partition[@Name]
- Format: string
- Default value: partition
- Required: true

12.3.4.5.7.2 alignment_priority

This attribute specifies the alignment priority of the partition used by the durability service to select the preferred partition to align from. If no alignment_priority is configured, the service uses 0 as default.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/Partition[@alignment_priority]
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

12.3.4.5.7.3 latency_budget

This attribute overrules the latency budget for this partition specified at this point.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/Partition[@latency_budget]
- Default value: 0.0
- Valid values: 0.0 / -
- Required: false

12.3.4.5.7.4 transport_priority

This attribute overrules the transport priority for this partition specified at this point.

- Full path: //OpenSplice/DurabilityService/Network/Alignment/Partition[@transport_priority]
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Required: false

TimeToWaitForAligner

When all Durability Services in the domain have configured their aligner element as false , 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 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
- Default value: 1.0
- Valid values: 0.0 / 1.0
- Occurrences min-max: 0-1

12.3.4.6 WaitForAttachment

The Durability service depends on the Networking service for its communication with other Durability services. Before it starts communicating, it must make sure the Networking 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: ServiceName
- Optional attributes: maxWaitCount

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, so a maxWaitCount of 100 represents 10 seconds. 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: integer
- Default value: 200
- Valid values: 1 / 1000
- Required: false

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
- Default value: networking
- Occurrences min-max: 1-*

12.3.5 MasterElection

For every namespace and role a single node is elected master, the one to provide historical data. This optional element can be used to fine-fune the characteristics of the master election algorithm.

This element is optional, and when used only applies to master election for namespaces with non-legacy masterPriority (see //OpenSplice/DurabilityService/NameSpaces/Policy[@masterPriority]).

- Full path: //OpenSplice/DurabilityService/MasterElection
- Occurrences min-max: 0-1
- Child elements: WaitTime

12.3.5.1 WaitTime

Before the durability service elects a master it can waits some time to detect other nodes in the system. Waiting might increase the possibility to elect a better candidate, but slows down the master election process. Electing a master too early may lead to additional alignment actions when a better candidate appears later, because a handover of mastership is needed. When the chance that a better master candicate appears within the WaitTime is small it is advised to set this value to 0.0 (the default).

- Full path: //OpenSplice/DurabilityService/MasterElection/WaitTime
- Dimension: seconds
- Default value: 0.0
- Valid values: 0.0 / 10.0
- Occurrences min-max: 0-1

12.3.6 Persistent

Durable data is divided in transient and persistent data. Transient data must stay available for as long as at least one Durability service is available in the system. For persistent data it is the same, but that type of data must also outlive the downtime of the system. The Durability service stores the persistent data on permanent storage to realize this. This element can be used to fine-tune the behaviour of the Durability service concerning the persistent properties of the data.

Note these elements are only available as part of the DDS persistence profile of Vortex OpenSplice.

- Full path: //OpenSplice/DurabilityService/Persistent
- Occurrences min-max: 0-1
- Child elements: StoreDirectory, StoreSessionTime, StoreSleepTime, StoreMode, StoreOptimizeInterval, QueueSize
- Optional attributes: SmpCount

12.3.6.1 StoreDirectory

This element determines the location where the persistent data will be stored on disk. If this parameter is not configured, the Durability service will not manage persistent data.

- Full path: //OpenSplice/DurabilityService/Persistent/StoreDirectory
- Format: string
- Default value: /tmp/pstore
- Occurrences min-max: 1-1

12.3.6.2 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
- Dimension: seconds
- Default value: 20.0
- Valid values: 0.001 / 60.0

- Occurrences min-max: 0-1

12.3.6.3 StoreSleepTime

This element specifies the period of time (in seconds) the persistency thread sleeps between two sessions. This allows influencing the CPU load of the persistency thread.

In most use cases there is no need to change the default value. Only in case the persistency thread takes up too much CPU time so that it prevents other threads from progressing a non-zero value should be used.

- Full path: //OpenSplice/DurabilityService/Persistent/StoreSleepTime
- Default value: 0.0
- Valid values: 0.0 / 10.0
- Occurrences min-max: 0-1

12.3.6.4 StoreMode

This element specifies the plug-in that is used to store the persistent data on disk. With "XML" mode, the service will store persistent data in XML files. With "KV" mode the service will store persistent data in a key-value store using either sqlite of leveldb to store the data on disk. !!! 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
- Default value: XML
- Valid values: XML, KV
- Occurrences min-max: 0-1

12.3.6.5 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! !!! The "MMF" store is deprecated from version 6.3 and was only implemented on linux !!!,

- Full path: //OpenSplice/DurabilityService/Persistent[@SmpCount]
- Format: integer
- Default value: 1
- Valid values: 1 / -
- Required: false

12.3.6.6 KeyValueStore

This element specifies the key-value store mode parameters. The Storage element specifies the which storage type is used to implement the key-value store. The storage type defaults to Sqlite3 when the storage type is not specified. Using the optional StorageParameters element parameters specific to the used storage implementation can be defined which are passed to the selected storage implementation. This element is only valid when the Persistent/StoreMode element is set to "KV". The "KV" store is currenly only supported on linux (SQLite and LevelDB), Windows (SQLite), and Solaris (SQLite).

- Full path: //OpenSplice/DurabilityService/Persistent/KeyValueStore
- Occurrences min-max: 0-1
- Child elements: StorageParameters
- Required attributes: type

type

This attribute specifies the third-party product that is used to implement the KV store. Products currently supported are SQLite and LevelDB. The following types can be selected:

- **sqlite3 -use sqlite3 as the key value store. Data in the store is stored efficiently** as binary blobs. This is the default.
- sqlite - same as sqlite3. This option is present for backward compatibility.
- **sqlitemt - use sqlite3 as the key value store implementation, but** stores the information as readable data. This mode allows a user to inspect the contents of the store, but the store performance will be lower compared to sqlite3 because each piece of data must be translated to readable format. This option is typically used for testing purposes to inspect the contents of the store.

  (Note: the abbreviation 'mt' stands for 'multi-table').
- leveldb - use leveldb as the key value store

- Full path: //OpenSplice/DurabilityService/Persistent/KeyValueStore[@type]
- Format: enumeration
- Default value: sqlite3
- Valid values: sqlite3, sqlite, sqlitemt, leveldb
- Required: true

StorageParameters

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/StorageParameters
- Format: string
- Default value: 0
- Occurrences min-max: 0-1

Compression

This element specifies compression settings for the key-value store.

Compression for the persistent store can be used to reduce disk I/O and lower disk space usage at the cost of processing power required to compress and uncompress the persistent data.

Compression is set at store creation time. Changing these settings after the store is created will result in an error.

- Full path: //OpenSplice/DurabilityService/Persistent/KeyValueStore/Compression
- Occurrences min-max: 0-1
- Required attributes: algorithm
- Optional attributes: enabled

12.3.6.6.3.1 algorithm

This attribute specifies the compression algorithm that is used to store the persistent data in the key-value store.

The lzf and snappy compression algorithms are built into the Vortex OpenSplice installation. To use the zlib algorithm a shared library needs to be available on the system, and it must be locatable by way of the current working directory, or via LD_LIBRARY_PATH (on Unix systems) or PATH (on Windows systems).

- Full path: //OpenSplice/DurabilityService/Persistent/KeyValueStore/Compression[@algorithm]
- Format: enumeration
- Default value: lzf
- Valid values: lzf, snappy, zlib
- Required: true

12.3.6.6.3.2 enabled

This attribute specifies whether the key-value store will apply compression for storing persistent data to limit disk usage.

- Full path: //OpenSplice/DurabilityService/Persistent/KeyValueStore/Compression[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.3.6.7 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: integer
- Default value: 0
- Valid values: 0 / 1000000000
- Occurrences min-max: 0-1

12.3.6.8 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: 0-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.3.6.8.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.3.7 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.

- Full path: //OpenSplice/DurabilityService/NameSpaces
- Occurrences min-max: 1-1

12.3.7.1 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: Partition, PartitionTopic
- Optional attributes: name, durabilityKind, alignmentKind, mergePolicy

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
- Default value: defaultNameSpace
- Required: false

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 the different namespaces 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 one single character.

- Full path: //OpenSplice/DurabilityService/NameSpaces/NameSpace/Partition
- Format: string
- Default value: *
- Occurrences min-max: 0-*

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 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 one single character.

- Full path: //OpenSplice/DurabilityService/NameSpaces/NameSpace/PartitionTopic
- Format: string
- Default value: .
- Occurrences min-max: 0-*

12.3.7.2 Policy

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.

A template specifies behaviour for a namespace. It matches a namespace name with an expression that may contain wildcards, thereby allowing dynamic configuration for namespaces. The order in which

templates are specified is important, as the first matching template will be the one that is selected for a namespace.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy
- Occurrences min-max: 0-*
- Required attributes: nameSpace, durability, aligner, alignee
- Optional attributes: delayedAlignment, equalityCheck, masterPriority

Merge

This tag specifies the merge policy applicable for a specific namespace. The following merge policy values are applicable:

- Ignore - No alignment will take place. This is also the default value if not specified.
- Merge - Existing data will remain and data from others will be aligned.
- Delete - Existing data is removed.
- Replace - Existing data is removed and data from others will be aligned.
- **Catchup - Existing data that is not available from others is removed, and data** that is added or changed by others will made available.

Note that the Replace and Catchup merge policies result in the same data set, but their instance states may differ after the merge policy has completed. In the Replace merge policy all instances present both before and after the merge transitioned through NOT_ALIVE_DISPOSED and end up as NEW instances with changes to the instance generation counters. In the Catchup merge policy the instance state of the instances that are not changed will remain untouched.

The scope attribute specifies for which role(s) the mergepolicy will be applied. A scope may contain the wildcards '*' to match any number of characters and '?' to match one single character.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy/Merge
- Occurrences min-max: 0-*
- Required attributes: type, scope

12.3.7.2.1.1 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.
- Catchup: When a state mismatch occurs, instances that do not exist any more are disposed, and instances that have been added or changed are updated. Instances for which no state mismatch occurs are left untouched. Immediately after successful completion of the Catchup merge action the data will be available to existing readers and late-joining readers.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy/Merge[@type]
- Format: enumeration
- Default value: Ignore
- Valid values: Ignore, Merge, Delete, Replace, Catchup
- Required: true

12.3.7.2.1.2 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
- Default value: *
- Required: true

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
- Default value: *
- Required: true

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**: Data is maximally handled as persistent. In practice this means a sample is handled exactly as specified in the DataWriter durability QoS that wrote it.
- **Transient**: A sample is maximally handled as if it were published with a transient durability QoS.
- **Transient_Local**: Data is maximally handled as if it were published with a transient_local durability QoS.
- **Durable**: Convenience value that behaves equal to Persistent.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy[@durability]
- Format: enumeration
- Default value: Durable

- Valid values: Durable, Persistent, Transient, Transient_Local
- Required: true

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
- Required: true

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 on each node may not be feasible. Often nodes 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).

The durability service provides the following mechanisms to request and provide historical data:

- **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 on the node, 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
- Default value: Initial
- Valid values: Initial, Lazy, On_Request
- Required: true

delayedAlignment

This element determines if the durability allows delayed alignment of initial data. This can be usefull for systems where there can be late-joining nodes 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 node when no writers have been created in the partitions matched by the namespace!

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy[@delayedAlignment]
- Format: boolean
- Default value: False
- Required: false

equalityCheck

This element specifies whether or not the durability service should compare its current data set with the data set of its aligner before applying the merge policy. If this option is enabled the aligner will align its data sets only in case there is a difference between his data sets and the data sets of this durability service. By default this option is NOT enabled, so data is always aligned even if there is no difference between data sets.

This option applies to all merge policies except for the IGNORE and DELETE merge policy.

NOTE Enabling this option may lead to less alignment data at the expense of processing power required to calculate hashes to compare the data sets. It is recommended to enable this option only for large data sets that do not change often. If this option is enabled for data sets that change often then chances of set equality are small, while the penalty to calculate hashes still exists.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy[@equalityCheck]
- Format: boolean
- Default value: False
- Required: false

masterPriority

This attribute sets the master selection priority of the namespace. The default value is 255, this allows to modify a single durability service in an existing system to become both preferred master as well as non-preferred master without having to change the configuration of all durability services. The Durability Service will never become master for a namespace when the masterPriority is set to zero. When set to 255 the durability Service will not use priorities at all, and falls back to the legacy master selection algorithm. If the masterPriority is set other than 0 and 255 the Durability Service will become master if it has the highest masterPriority of all discovered Durability Services for the namespace. If multiple Durability Services exist having equally highest masterPriority a further selection will be made based on the highest namespace quality of each Durability Service (but only if persistent data has not been injected before). Then if there are still multiple equally suitable Durability Services the Durability Service with the highest system id will be selected.

- Full path: //OpenSplice/DurabilityService/NameSpaces/Policy[@masterPriority]
- Format: integer
- Default value: 255
- Valid values: 0 / 255
- Required: false

12.3.8 EntityNames

This element specifies the names of the various entities used by the DurabilityService. The names specified here will be displayed in the Vortex OpenSplice Tuner when viewing the DurabilityService.

- Full path: //OpenSplice/DurabilityService/EntityNames
- Occurrences min-max: 0-1

- Child elements: Publisher, Subscriber, Partition

12.3.8.1 Publisher

This element specifies the name of the durability publisher.

- Full path: //OpenSplice/DurabilityService/EntityNames/Publisher
- Format: string
- Default value: durabilityPublisher
- Occurrences min-max: 0-1

12.3.8.2 Subscriber

This element specifies the name of the durability subscriber.

- Full path: //OpenSplice/DurabilityService/EntityNames/Subscriber
- Format: string
- Default value: durabilitySubscriber
- Occurrences min-max: 0-1

12.3.8.3 Partition

This element specifies the name of the durability partition.

- Full path: //OpenSplice/DurabilityService/EntityNames/Partition
- Format: string
- Default value: durabilityPartition
- Occurrences min-max: 0-1

12.3.9 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: OutputFile, Timestamps, Verbosity
- Optional attributes: synchronous

12.3.9.1 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 when the service crashes.

- Full path: //OpenSplice/DurabilityService/Tracing[@synchronous]
- Format: boolean
- Default value: FALSE

- Required: false

12.3.9.2 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
- Default value: durability.log
- Occurrences min-max: 0-1

12.3.9.3 Timestamps

This element specifies whether the logging must contain timestamps.

- Full path: //OpenSplice/DurabilityService/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

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
- Default value: true
- Required: false

12.3.9.4 Verbosity

This element specifies the verbosity level of the logging information. The higher the level, the more (detailed) information will be logged.

- Full path: //OpenSplice/DurabilityService/Tracing/Verbosity
- Format: enumeration
- Default value: INFO
- Valid values: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, NONE
- Occurrences min-max: 0-1

12.4 SNetworkService

When communication endpoints are located on different computing nodes, the data produced using the local DDS service must be communicated to the remote DDS service and the other way around. The Networking service provides a bridge between the local DDS service and a network interface. Multiple Networking services can exist next to each other; each serving one (or more) physical network interface(s). The Secure Networking 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.

- Full path: //OpenSplice/SNetworkService
- Occurrences min-max: 0-*
- Required attributes: name

12.4.1 name

This attribute identifies the configuration for the Secure Networking 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 *//OpenSplice/Domain/Service[@name]* in the configuration of the DomainService.

- Full path: //OpenSplice/SNetworkService[@name]
- Format: string
- Default value: snetworking
- Required: true

12.4.2 Watchdog

This element controls the characteristics of the Watchdog thread.

- Full path: //OpenSplice/SNetworkService/Watchdog
- Occurrences min-max: 0-1

12.4.2.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/SNetworkService/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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/SNetworkService/Watchdog/Scheduling/Priority
- Format: integer

- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.4.2.1.1.1 priority_kind

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

- Full path: //OpenSplice/SNetworkService/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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/SNetworkService/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.4.3 General

This element contains general parameters that concern the secure networking service as a whole.

- Full path: //OpenSplice/SNetworkService/General
- Occurrences min-max: 0-1
- Child elements: NetworkInterfaceAddress, EnableMulticastLoopback, LegacyCompression

12.4.3.1 NetworkInterfaceAddress

This element specifies which network interface card should be used. Every Secure Networking service is bound to only one network interface card (NIC). 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, the Vortex OpenSplice middleware will try to look up an interface that has the required capabilities.

- Full path: //OpenSplice/SNetworkService/General/NetworkInterfaceAddress
- Format: string
- Default value: first available
- Occurrences min-max: 0-1
- Optional attributes: forced, ipv6, bind, allowReuse

forced

This attribute specifies whether only the selected NetworkInterfaceAddress should be used or others can be used too.

- false - Specifies that the NetworkInterfaceAddress is first used but when not available another, when available, is used. (default).
- true - Specifies that only the selected NetworkInterfaceAddress can be used.

- Full path: //OpenSplice/SNetworkService/General/NetworkInterfaceAddress[@forced]
- Format: boolean
- Default value: false
- Required: false

ipv6

This attribute specifies whether IPv6 should be used for communication.

- false - specifies that IPv4 should be used (default).
- true - Specifies that IPv6 should be used.

This setting will be overriden & ignored if the element NetworkInterfaceAddress has an explicit value that is unequivocally either an IPv4 or IPv6 address. This attribute is therefore only optionally required to specify IPv6 communication when special values like "first available" or an interface name are used instead of IP addresses.

- Full path: //OpenSplice/SNetworkService/General/NetworkInterfaceAddress[@ipv6]
- Format: boolean
- Default value: false
- Required: false

bind

Specifies the bind strategy to be used by the networking service for its sockets.

- any - Specifies that the service should bind to the wildcard-address (INADDR_ANY) (default).
- strict - Specifies that the service should bind to the NetworkingInterfaceAddress.

- Full path: //OpenSplice/SNetworkService/General/NetworkInterfaceAddress[@bind]
- Format: enumeration
- Default value: any
- Valid values: any, strict
- Required: false

allowReuse

By default the networking service will bind to a port allowing other services to bind to the same port as well (so reuse of the port is allowed). By setting this option to 'false', the port is bound exclusively (SO_REUSEADDR disabled).

- true - Ports can be reused (SO_REUSEADDR enabled) (default).
- false - Ports are bound exclusively.

- Full path: //OpenSplice/SNetworkService/General/NetworkInterfaceAddress[@allowReuse]
- Format: boolean
- Default value: true
- Required: false

12.4.3.2 EnableMulticastLoopback

EnableMulticastLoopback specifies whether the secure networking service will allow IP multicast packets within the node to be visible to all secure networking participants in the node, including itself. It must be TRUE for intra-node multicast communications, but if a node runs only a single Vortex OpenSplice secure networking service and does not host any other networking-capable programs, it may be set to FALSE for improved performance.

- Full path: //OpenSplice/SNetworkService/General/EnableMulticastLoopback
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.4.3.3 LegacyCompression

This element specifies if compression is applied after of before fragmentations. When set to TRUE compression is applied after fragmentation which is provided for backward compatibility. When set to FALSE compression is applied before fragmentation. The default value is TRUE.

- Full path: //OpenSplice/SNetworkService/General/LegacyCompression
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.4.3.4 Reconnection

This element specifies the desired secure networking-behavior with respect to the validity of restoring lost connectivity with remote nodes. Here 'lost connectivity' means a prolonged inability to communicate with a known and still active remote node (typically because of network-issues) that has resulted in such a node 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 node will be restored, even though reliable data might have been lost during the disconnection period.

- Full path: //OpenSplice/SNetworkService/General/Reconnection
- Occurrences min-max: 0-1
- Required attributes: allowed

allowed

This attribute specifies whether the network service must resume communication with an other 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. (default).

- true - Specifies that the network service must resume communication.

- Full path: //OpenSplice/SNetworkService/General/Reconnection[@allowed]
- Format: boolean
- Default value: false
- Required: true

12.4.4 Partitioning

The Vortex OpenSplice Secure Networking 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 secure networking services in the system can be explicitly instructed to use specific unicast or multicast addresses for its networking traffic. This is done by means of so-called network partitions

A network partition is defined by one or more unicast, multicast of broadcast IP addresses. Any secure networking 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 networking service, both its partition and topic name will be inspected. In combination with a set of network partition mapping rules, the service will determine to which network partition the sample is written. The mapping rules are configurable as well.

Using networking configuration, nodes can be disconnected from any networking 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 on its turn is mapped onto a specific networking partition, which on its turn is only connected to those nodes that are capable of handling high volume data.

- Full path: //OpenSplice/SNetworkService/Partitioning
- Occurrences min-max: 0-1

12.4.4.1 GlobalPartition

This element specifies the global or default secure networking partition.

- Full path: //OpenSplice/SNetworkService/Partitioning/GlobalPartition
- Occurrences min-max: 0-1
- Required attributes: Address
- Optional attributes: SecurityProfile, MulticastTimeToLive

Address

The global networking partition transports data that is either meant to be global, like discovery heartbeats, or that is not mapped onto any other networking partition. 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 are separated by a colon (,) semicolon (;) or space ( ). 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 notation or as the symbolic hostname, in which case the middleware will try to resolve the corresponding IP address.

- Full path: //OpenSplice/SNetworkService/Partitioning/GlobalPartition[@Address]
- Format: string
- Default value: broadcast
- Required: true

SecurityProfile

In the context of secure networking, the GlobalPartition element provides support for the attribute SecurityProfile. The attribute is referencing 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/SNetworkService/Partitioning/GlobalPartition[@SecurityProfile]
- Format: string
- Default value: nullProfile
- Required: false

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/SNetworkService/Partitioning/GlobalPartition[@MulticastTimeToLive]
- Default value: 32
- Valid values: 0 / 255
- Required: false

12.4.4.2 NetworkPartitions

Networking configuration can contain a set of networking partitions, which are grouped under the NetworkPartitions element.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions
- Occurrences min-max: 0-1

NetworkPartition

Every NetworkPartition has a name, an address and a connected flag.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition
- Occurrences min-max: 1-*
- Required attributes: Address
- Optional attributes: Name, Connected, Compression, SecurityProfile, MulticastTimeToLive

12.4.4.2.1.1 Name

The Name attribute identifies a Network Partition; it must be unique to create associations with Element PartitionMappings.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Name]
- Format: string
- Default value: networkPartition
- Required: false

12.4.4.2.1.2 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 Vortex OpenSplice will try to resolve the corresponding IP address.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Address]
- Format: string
- Default value: broadcast
- Required: true

12.4.4.2.1.3 Connected

A node can choose to be not connected to a networking partition by setting the Connected attribute.

If a node is connected to a networking partition, it will join the corresponding multicast group and it will receive data distributed over the partition. If it is not connected, data distributed over the partition will not reach the node but will be filtered by the networking interface or multicast enabled switches.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Connected]
- Format: boolean
- Default value: true
- Required: false

12.4.4.2.1.4 Compression

This attribute specifies if networking will apply compression to limit bandwidth for a specific network partition. This provides great flexibility as network partition are dynamically bind to logical partitions. Compression is performed before fragmentation of the messages. To provide backward compatibility the option LegacyCompression (see General options) can be set to provide compression after fragmentation. The following compression values are allowed:

- false - No compression is applied. This is also the default value if not specified.
- true - Compression is applicable

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Compression]
- Format: boolean
- Default value: false
- Required: false

12.4.4.2.1.5 SecurityProfile

In the context of secure networking, the NetworkPartition element provides support for the attribute SecurityProfile. The attribute is referencing a security profile 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 Vortex 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 Vortex OpenSplice node in question.

- Full path: //OpenSplice/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@SecurityProfile]

- Format: string
- Default value: nullProfile
- Required: false

12.4.4.2.1.6 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/SNetworkService/Partitioning/NetworkPartitions/NetworkPartition[@MulticastTimeToLive]
- Default value: 32
- Valid values: 0 / 255
- Required: false

12.4.4.3 IgnoredPartitions

This element is used to group the set of IgnoredPartition elements.

- Full path: //OpenSplice/SNetworkService/Partitioning/IgnoredPartitions
- Occurrences min-max: 0-1

IgnoredPartition

This element can be used to create a "Local Partition" that is only available on the node 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 Networking service.

- Full path: //OpenSplice/SNetworkService/Partitioning/IgnoredPartitions/IgnoredPartition
- Occurrences min-max: 1-*
- Required attributes: DCPSPartitionTopic

12.4.4.3.1.1 DCPSPartitionTopic

The Networking 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/SNetworkService/Partitioning/IgnoredPartitions/IgnoredPartition[@DCPSPartitionTopic]
- Format: string
- Default value: .
- Required: true

12.4.4.4 PartitionMappings

This element is used to group the set of PartitionMapping elements.

- Full path: //OpenSplice/SNetworkService/Partitioning/PartitionMappings
- Occurrences min-max: 0-1

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. Networking allows for a set of partition mappings to be defined.

- Full path: //OpenSplice/SNetworkService/Partitioning/PartitionMappings/PartitionMapping
- Occurrences min-max: 1-*
- Required attributes: NetworkPartition, DCPSPartitionTopic

12.4.4.4.1.1 NetworkPartition

The NetworkPartition attribute of a partition mapping defines that networking partitition that data in a specific DCPS partition of a specific DCPS topic should be sent to.

- Full path: //OpenSplice/SNetworkService/Partitioning/PartitionMappings/PartitionMapping[@NetworkPartition]
- Format: string
- Default value: networkPartition
- Required: true

12.4.4.4.1.2 DCPSPartitionTopic

The Networking 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. 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/SNetworkService/Partitioning/PartitionMappings/PartitionMapping[@DCPSPartitionTopic]
- Format: string
- Default value: .
- Required: true

12.4.5 Security

The Security section defines the parameters relevant for secure networking. Declaring this element in the OSPL configuration file will activate the secure networking feature. Without any additional security settings, all network partitions of the node would use the NULL cipher encoding. If confidentiality and integrity is required for a network partition, the network partition must be associated with a security profile

- Full path: //OpenSplice/SNetworkService/Security
- Occurrences min-max: 0-1
- Optional attributes: enabled

12.4.5.1 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, but message items will be sent unencrypted. In case of enabled="false" the security feature will not be activated, and the node acts like any other Vortex 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/SNetworkService/Security[@enabled]
- Format: boolean
- Default value: false
- Required: false

12.4.5.2 SecurityProfile

This element defines the security profile which can be applied to one or more network partitions. This element is optional.

- Full path: //OpenSplice/SNetworkService/Security/SecurityProfile
- Occurrences min-max: 0-*
- Required attributes: Name, Cipher, CipherKey

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/SNetworkService/Security/SecurityProfile[@Name]
- Format: string
- Default value: aSecurityProfile
- Required: true

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 the 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 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/SNetworkService/Security/SecurityProfile[@Cipher]
- Format: string
- Default value: null

- Valid values: aes128, aes192, aes256, blowfish, null, rsa-aes128, rsa-aes192, rsa-aes256, rsa-blowfish, rsa-null
- Required: true

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 Vortex 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 Vortex OpenSplice node. Because of this, the 'in-place' method is strongly discouraged.

- Full path: //OpenSplice/SNetworkService/Security/SecurityProfile[@CipherKey]
- Format: string
- Default value: n/a
- Required: true

12.4.5.3 AccessControl

The optional AccessControl element defines settings for access control enforcement and which access control module shall be used.

- Full path: //OpenSplice/SNetworkService/Security/AccessControl
- Occurrences min-max: 0-1
- Optional attributes: enabled, policy

enabled

The access control feature will be activated when enabled="true"

- Full path: //OpenSplice/SNetworkService/Security/AccessControl[@enabled]
- Format: boolean
- Default value: false
- Required: false

policy

The policy attribute references a file containing the access control policy.

- Full path: //OpenSplice/SNetworkService/Security/AccessControl[@policy]
- Format: string

- Default value: “”
- Required: false

AccessControlModule

The AccessControlModule element defines which access control module will be used. More than one module may be defined. All defined and enabled modules will be used to determine if access should be granted.

- Full path: //OpenSplice/SNetworkService/Security/AccessControl/AccessControlModule
- Occurrences min-max: 0-*
- Optional attributes: enabled, type

12.4.5.3.3.1 enabled

The module specified in the type attribute is used to evaluate access control rules when enabled=”true”.

- Full path: //OpenSplice/SNetworkService/Security/AccessControl/AccessControlModule[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.4.5.3.3.2 type

The type attribute defines the access control model type. Currently, Vortex OpenSplice only supports mandatory access control, accordingly the only valid value for this attribute is “MAC”.

- Full path: //OpenSplice/SNetworkService/Security/AccessControl/AccessControlModule[@type]
- Format: string
- Default value: none
- Required: false

12.4.5.4 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/SNetworkService/Security/Authentication
- Occurrences min-max: 0-1
- Optional attributes: enabled

enabled

Authentication is performed when enabled is set to true.

- Full path: //OpenSplice/SNetworkService/Security/Authentication[@enabled]
- Format: boolean
- Default value: true
- Required: false

X509Authentication

The X509Authentication element defines where keys and certificates required for X509 authentication may be found.

- Full path: //OpenSplice/SNetworkService/Security/Authentication/X509Authentication
- Occurrences min-max: 0-1
- Child elements: TrustedCertificates

12.4.5.4.2.1 Credentials

The Credentials element is an optional element. If it is missing, then the node does not sign messages (in other words, does not send credentials).

- Full path: //OpenSplice/SNetworkService/Security/Authentication/X509Authentication/Credentials
- Occurrences min-max: 0-1
- Child elements: Key, Cert

12.4.5.4.2.2 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 Vortex OpenSplice daemon is started.

- Full path: //OpenSplice/SNetworkService/Security/Authentication/X509Authentication/Credentials/Key
- Format: string
- Occurrences min-max: 1-1

12.4.5.4.2.3 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 Vortex OpenSplice daemon is started.

- Full path: //OpenSplice/SNetworkService/Security/Authentication/X509Authentication/Credentials/Cert
- Format: string
- Occurrences min-max: 1-1

12.4.5.4.2.4 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 Vortex OpenSplice daemon is started.

- Full path: //OpenSplice/SNetworkService/Security/Authentication/X509Authentication/TrustedCertificates
- Format: string
- Occurrences min-max: 1-1

12.4.6 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, reliability 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/SNetworkService/Channels
- Occurrences min-max: 1-1
- Child elements: AllowedPorts

12.4.6.1 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/SNetworkService/Channels/Channel
- Occurrences min-max: 1-42
- Child elements: PortNr, FragmentSize, Resolution, AdminQueueSize, CompressionBufferSize, CompressionThreshold, AllowedPorts
- Required attributes: name, reliable, enabled
- Optional attributes: default, priority

name

The name uniquely identifies the channel.

- Full path: //OpenSplice/SNetworkService/Channels/Channel[@name]
- Format: string
- Default value: aChannel
- Required: true

reliable

If this attribute is set to true, the channel sends all messages reliably. If not, data is sent only once (fire-and-forget).

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.

- Full path: //OpenSplice/SNetworkService/Channels/Channel[@reliable]
- Format: boolean
- Default value: false
- Required: true

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 networking 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.

Note that only one channel is allowed to have this attribute set to true

- Full path: //OpenSplice/SNetworkService/Channels/Channel[@default]
- Format: boolean
- Default value: false
- Required: false

enabled

This attribute toggles a channel on or off. Toggling a channel between enabled and disabled is a quick alternative for commenting out the corresponding lines in the configuration file.

- Full path: //OpenSplice/SNetworkService/Channels/Channel[@enabled]
- Format: boolean
- Default value: false
- Required: true

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 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/SNetworkService/Channels/Channel[@priority]
- Format: integer
- Default value: 0
- Required: false

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/SNetworkService/Channels/Channel/PortNr
- Format: integer
- Default value: 53400
- Valid values: 1 / 65535
- Occurrences min-max: 1-1

FragmentSize

The networking module will fragment large message into smaller fragments with size **FragmentSize**. These fragments are sent as datagrams to the UDP stack. OS-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/SNetworkService/Channels/Channel/FragmentSize
- Format: unsigned int
- Dimension: bytes
- Default value: 1300
- Valid values: 200 / 65535
- Occurrences min-max: 0-1

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 ThrottleTreshold consistently throughout the system.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Resolution
- Format: unsigned int
- Dimension: milliseconds
- Default value: 10
- Valid values: 1 / -
- Occurrences min-max: 0-1

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/SNetworkService/Channels/Channel/AdminQueueSize
- Format: integer
- Default value: 4000
- Valid values: 400 / -
- Occurrences min-max: 0-1

CompressionBufferSize

When compression on messages is enabled then the CompressionBufferSize specifies the initial size of the compression/decompression buffer. The compression buffer is used to store the messages before they are compressed and send on the network. The decompression buffer is used to decompress the received compressed messages. Note that the actual size of these buffers may be increased when needed.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/CompressionBufferSize
- Format: integer
- Default value: 131072
- Valid values: 65536 / -
- Occurrences min-max: 0-1

CompressionThreshold

When compression on messages is enabled then the CompressionThreshold provides a threshold to start compressing the accumulated data and sending the compressed data on the network. The CompressionThreshold is used to estimate the size of the compressed messages.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/CompressionThreshold
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

Sending

This element describes all properties for the transmitting side of the Channel.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending
- Occurrences min-max: 0-1
- Child elements: CrcCheck, QueueSize, MaxBurstSize, ThrottleLimit, ThrottleThreshold, MaxRetries, RecoveryFactor, DiffServField, DontRoute, DontFragment, TimeToLive

12.4.6.1.12.1 CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.4.6.1.12.2 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/SNetworkService/Channels/Channel/Sending/QueueSize
- Format: integer

- Default value: 400
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.4.6.1.12.3 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 considered "unlimited" as this far exceeds the capacity of modern 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/SNetworkService/Channels/Channel/Sending/MaxBurstSize
- Dimension: bytes/(resolution interval)
- Default value: 1073741823
- Valid values: 1024 / 1073741823
- Occurrences min-max: 0-1

12.4.6.1.12.4 ThrottleLimit

Throttling will enable you to further limit (below MaxBurstSize) the amount of data that is sent every Resolution interval. This happens if one of the receiving nodes in the network 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 (or higher) as 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. The human-readable option lets the user postfix the value with K(ilobyte), M(egabyte) or G(igabtye). For example, 10K results in 10240 bytes

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/ThrottleLimit
- Dimension: bytes/(resolution interval)
- Default value: 10240
- Valid values: - / 4294967295
- Occurrences min-max: 0-1

12.4.6.1.12.5 ThrottleThreshold

This is the number of unprocessed network fragments that a node will store before it will inform the other nodes in the network that it has trouble processing the incoming data. Those other nodes 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 node is able to catch up.

It is considered good practice to specify the ThrottleTreshold consistently throughout the system.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/ThrottleThreshold
- Format: integer
- Dimension: fragments
- Default value: 50
- Valid values: 2 / -
- Occurrences min-max: 0-1

12.4.6.1.12.6 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/SNetworkService/Channels/Channel/Sending/MaxRetries
- Format: integer
- Default value: 100
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.4.6.1.12.7 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 218.) The lost message is resent after Resolution * RecoveryFactor milliseconds.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/RecoveryFactor
- Format: integer
- Default value: 3
- Valid values: 2 / -
- Occurrences min-max: 0-1

12.4.6.1.12.8 DiffServField

This element describes the DiffServ setting the channel will apply to the networking messages. This parameter determines the value of the diffserv field of the IP version 4 packets send on this channel which allows QoS setting to be applied to the network traffic send on this channel.

Windows platform support for setting the diffserv field is dependent on the OS version. For Windows versions XP SP2 and 2003 to use the diffserv field the following parameter should be added to the register:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting

The type of this parameter is a DWORD and it's value should be set to 0 to allow setting of the diffserv field.

For Windows version 7 or higher a new API (qWAVE) has been introduced For these platforms the specified diffserv value is mapped to one of the support traffic types. The mapping is as follows: 1-8 background traffic; 9-40 excellent traffic; 41-55 audio/video traffic; 56 voice traffic; 57-63 control traffic. When Vortex OpenSplice is run without Administrative priveleges then only the diffserv value of 0, 8, 40 or 56 is allowed.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/DiffServField
- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

12.4.6.1.12.9 DontRoute

The IP DONTROUTE socket option is set to the value specified.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/DontRoute
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.4.6.1.12.10 DontFragment

Controls whether the "don't fragment" bit (DF) is set on outgoing UDP IPv4 packets. Note that not all operating systems support setting this bit. When the operating system does not support setting the DF bit this option is ignored.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/DontFragment
- Format: boolean
- Default value: False
- Valid values: True, False
- Occurrences min-max: 0-1

12.4.6.1.12.11 TimeToLive

For each UDP packet sent out, the IP Time To Live header value is set to the value specified.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/TimeToLive
- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

12.4.6.1.12.12 Scheduling

This element specifies the scheduling policies used to control the transmitter thread of the current Channel.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.4.6.1.12.13 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/SNetworkService/Channels/Channel/Sending/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1

- Optional attributes: priority_kind

12.4.6.1.12.14 priority_kind

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

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Sending/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.4.6.1.12.15 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/SNetworkService/Channels/Channel/Sending/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

Receiving

This element describes all properties for the receiving side of the Channel.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving
- Occurrences min-max: 0-1
- Child elements: CrcCheck, ReceiveBufferSize, DefragBufferSize, MaxReliabBacklog, PacketRetentionPeriod, ReliabilityRecoveryPeriod

12.4.6.1.13.1 CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the receiving side is enabled only network packets that contain a valid crc field are accepted.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.4.6.1.13.2 ReceiveBufferSize

The UDP receive buffer of the best effort 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/SNetworkService/Channels/Channel/Receiving/ReceiveBufferSize
- Default value: 1000000
- Valid values: 1024 / -
- Occurrences min-max: 0-1

12.4.6.1.13.3 Scheduling

This element specifies the scheduling policies used to control the receiver thread of the current Channel.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.4.6.1.13.4 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/SNetworkService/Channels/Channel/Receiving/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.4.6.1.13.5 priority_kind

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

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.4.6.1.13.6 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/SNetworkService/Channels/Channel/Receiving/Scheduling/Class
- Format: enumeration
- Default value: Default

- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.4.6.1.13.7 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 Element FragmentSize)

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/DefragBufferSize
- Default value: 5000
- Valid values: 500 / -
- Occurrences min-max: 0-1

12.4.6.1.13.8 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/SNetworkService/Channels/Channel/Receiving/SMPOptimization
- Occurrences min-max: 0-1
- Required attributes: enabled

12.4.6.1.13.9 enabled

This attribute toggles the Optimization on or off.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/SMPOptimization[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.4.6.1.13.10 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 Element Reconnection).

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/MaxReliabBacklog
- Format: integer
- Default value: 1000
- Valid values: 100 / -
- Occurrences min-max: 0-1

12.4.6.1.13.11 PacketRetentionPeriod

This element specifies the number of milliseconds 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) nodes, even though some nodes might not have received some packets at the moment a sending node disappears (for

whatever reason). The protocol implies that each node retains sufficient received data so that it can be (re-)distributed if a publishing node disappears before all receiving nodes are "up-to-date". When the PacketRetentionPeriod element is set to 0 (the default value), the alignment amongst receiving nodes will not occur. To activate the extended realibility protocol, 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 networking services in the system.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/PacketRetentionPeriod
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.4.6.1.13.12 ReliabilityRecoveryPeriod

This element specifies a timeout period (in milliseconds) for the alignment phase of the extended reliability protocol. 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 nodes (following the disappearance or crash of a publishing node) 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 nodes following the disappearance of a sending node in the system.

- Full path: //OpenSplice/SNetworkService/Channels/Channel/Receiving/ReliabilityRecoveryPeriod
- Format: integer
- Default value: 1000
- Valid values: 0 / -
- Occurrences min-max: 0-1

AllowedPorts

AllowedPorts 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. When also 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 a comma (,). 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/SNetworkService/Channels/Channel/AllowedPorts
- Format: string
- Occurrences min-max: 0-1

12.4.6.2 AllowedPorts

AllowedPorts 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. When also 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 a comma (,). 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/SNetworkService/Channels/AllowedPorts
- Format: string
- Occurrences min-max: 0-1

12.4.7 Discovery

This element is used to configure the various parameters of the Discovery Channel, which is used to discover all relevant participating entities in the current Domain. The purpose of the discovery process is to build-up and maintain a notion of all relevant active nodes within the domain. The relevance of discovered remote nodes can be defined statically (by definition of the so-called Global Partition) and/or can be dynamically expanded and maintained by the dynamic-discovery process driven by the node's Role and Scope.

- Full path: //OpenSplice/SNetworkService/Discovery
- Occurrences min-max: 0-1
- Child elements: PortNr, ProbeList
- Optional attributes: enabled, Scope

12.4.7.1 enabled

This element can be used to enable or disable the Discovery Channel. In case the Discovery Channel is disabled, entities will only detect each others presence implicitly once messages are received for the first time.

- Full path: //OpenSplice/SNetworkService/Discovery[@enabled]
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Required: false

12.4.7.2 Scope

This attribute controls the dynamic discovery behaviour of this node within the current Domain. If it is not set, dynamic discovery will be disabled and the networking service will only communicate with nodes that can be reached through the predefined Global Partition. If the Scope attribute is specified, dynamic discovery is enabled and the networking service will be able to communicate with all nodes in the system that have a Role that matches the Scope expression. The Scope expression can contain a comma separated list of wild-card role-expressions. If the role of any discovered node matches any of the wild-card expressions, the remote node is considered a match and will become part of the communication reach (i.e. the Global Partition) of the current domain.

- Full path: //OpenSplice/SNetworkService/Discovery[@Scope]
- Format: string

- Occurrences min-max: 0-1
- Required: false

12.4.7.3 PortNr

This element specifies the Port number used by the Discovery Channel.

- Full path: //OpenSplice/SNetworkService/Discovery/PortNr
- Format: integer
- Default value: 3369
- Valid values: 1 / 65536
- Occurrences min-max: 1-1

12.4.7.4 ProbeList

This element contains the addresses of the nodes that will be contacted to retrieve an initial list of participating nodes in the current domain that match the specified Scope. Multiple ProbeList addresses can be entered by separating them by a colon (,), semicolon (;) or space( ) . The addresses can be entered as dotted decimal notation or as the symbolic hostname, in which case the middleware will try to resolve the corresponding IP address.

- Full path: //OpenSplice/SNetworkService/Discovery/ProbeList
- Format: string
- Occurrences min-max: 0-1

12.4.7.5 Sending

This element describes all properties for the transmitting side of the Discovery Channel.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending
- Occurrences min-max: 0-1
- Child elements: CrcCheck, DiffServField, DontRoute, DontFragment, TimeToLive, Interval, SafetyFactor, SalvoSize

CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

DiffServField

This element describes the DiffServ setting the channel will apply to the networking messages. This parameter determines the value of the diffserv field of the IP version 4 packets send on this channel which allows QoS setting to be applied to the network traffic send on this channel.

Windows platform support for setting the diffserv field is dependent on the OS version. For Windows versions XP SP2 and 2003 to use the diffserv field the following parameter should be added to the register:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting

The type of this parameter is a DWORD and it's value should be set to 0 to allow setting of the diffserv field.

For Windows version 7 or higher a new API (qWAVE) has been introduced For these platforms the specified diffserv value is mapped to one of the support traffic types. The mapping is as follows: 1-8 background traffic; 9-40 excellent traffic; 41-55 audio/video traffic; 56 voice traffic; 57-63 control traffic. When Vortex OpenSplice is run without Administrative priveleges then only the diffserv value of 0, 8, 40 or 56 is allowed.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/DiffServField
- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

DontRoute

The IP DONTROUTE socket option is set to the value specified.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/DontRoute
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

DontFragment

Controls whether the "don't fragment" bit (DF) is set on outgoing UDP IPv4 packets. Note that not all operating systems support setting this bit. When the operating system does not support setting the DF bit this option is ignored.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/DontFragment
- Format: boolean
- Default value: False
- Valid values: True, False
- Occurrences min-max: 0-1

TimeToLive

For each UDP packet sent out, the IP Time To Live header value is set to the value specified.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/TimeToLive
- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

Scheduling

This element specifies the scheduling policies used to control the transmitter thread of the Discovery Channel.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.4.7.5.6.1 Priority

This element specifies the thread priority that will be used by the transmitter thread of the Discovery Channel. 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/SNetworkService/Discovery/Sending/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.4.7.5.6.2 priority_kind

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

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.4.7.5.6.3 Class

This element specifies the thread scheduling class that will be used by the transmitter thread of the Discovery Channel. 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/SNetworkService/Discovery/Sending/Scheduling/Class
- Format: enumeration
- Default value: Default

- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

Interval

This element describes the interval(in milliseconds) at which remote nodes will expect heartbeats from this node.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/Interval
- Format: integer
- Default value: 333
- Valid values: 10 / -
- Occurrences min-max: 0-1

SafetyFactor

The SafetyFactor is used to set a margin on the discovery sending. This avoids tight timing issues.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/SafetyFactor
- Default value: 0.9
- Valid values: 0.2 / 1.0
- Occurrences min-max: 0-1

SalvoSize

During starting and stopping, discovery messages are sent at higher frequency. This SalvoSize sets the number of messages to send during these phases.

- Full path: //OpenSplice/SNetworkService/Discovery/Sending/SalvoSize
- Format: integer
- Default value: 3
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.4.7.6 Receiving

This element describes all properties for the receiving side of the Discovery Channel.

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving
- Occurrences min-max: 0-1
- Child elements: CrcCheck, DeathDetectionCount, ReceiveBufferSize

CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

Scheduling

This element specifies the scheduling policies used to control the receiver thread of the Discovery Channel.

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.4.7.6.2.1 Priority

This element specifies the thread priority that will be used by the receiver thread of the Discovery Channel. 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/SNetworkService/Discovery/Receiving/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.4.7.6.2.2 priority_kind

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

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.4.7.6.2.3 Class

This element specifies the thread scheduling class that will be used by the receiver thread of the Discovery Channel. 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/SNetworkService/Discovery/Receiving/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

DeathDetectionCount

This element specifies how often a heartbeat from a remote node must miss its Interval before that remote node is considered dead.

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving/DeathDetectionCount
- Format: integer
- Default value: 6
- Valid values: 1 / -
- Occurrences min-max: 0-1

ReceiveBufferSize

The UDP receive buffer of the Discovery Channel socket is set to the value given. If many message are lost, the receive buffer size has to be increased.

- Full path: //OpenSplice/SNetworkService/Discovery/Receiving/ReceiveBufferSize
- Default value: 1000000
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.4.8 Tracing

This element controls the amount and type of information that is written into the tracing log by the Networking Service. This is useful to track the Networking Service during application development. In the runtime system it should be turned off.

- Full path: //OpenSplice/SNetworkService/Tracing
- Occurrences min-max: 0-1
- Child elements: OutputFile, Timestamps, Verbosity
- Optional attributes: enabled

12.4.8.1 enabled

This attribute controls whether the tracing option is enabled or not.

- Full path: //OpenSplice/SNetworkService/Tracing[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.4.8.2 OutputFile

This option specifies where the logging is printed to. Note that "stdout" is considered a legal value that represents "standard out". The default value is an empty string, indicating that the tracing log will be written to standard out.

- Full path: //OpenSplice/SNetworkService/Tracing/OutputFile
- Format: string
- Default value: networking.log

- Occurrences min-max: 1-1

12.4.8.3 Timestamps

This element specifies whether the logging must contain timestamps.

- Full path: //OpenSplice/SNetworkService/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

absolute

This attribute specifies whether the timestamps are absolute or relative to the startup time of the service.

- Full path: //OpenSplice/SNetworkService/Tracing/Timestamps[@absolute]
- Format: boolean
- Default value: true
- Required: false

12.4.8.4 Categories

This element contains the logging properties for various networking categories.

- Full path: //OpenSplice/SNetworkService/Tracing/Categories
- Occurrences min-max: 1-1
- Child elements: Default, Configuration, Construction, Destruction, Mainloop, Groups, Send, Receive, Throttling, Test, Discovery

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/SNetworkService/Tracing/Categories/Default
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Configuration

- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Construction
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Destruction
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Mainloop
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Groups

- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Send
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Receive
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

Throttling

This element specifies the tracing level for the *Throttling* category. This includes information about throttling. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

- Full path: //OpenSplice/SNetworkService/Tracing/Categories/Throttling
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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/SNetworkService/Tracing/Categories/Test
- Format: integer
- Default value: 0
- Valid values: 0 / 6

- Occurrences min-max: 0-1

Discovery

This element specifies the tracing level for the *Discovery* category. This includes all activity related to the discovery channel. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

- Full path: //OpenSplice/SNetworkService/Tracing/Categories/Discovery
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

12.4.8.5 Verbosity

The Verbosity attribute sets the level of tracing for all Tracing Catagories. This attribute is an additional method besides the Tracing/Catagories tag to specify trace levels. The difference is that Verbosity sets the level for all catagories similar as by other services whereas the Tracing/Catagories element allows to set the trace level per catagory. The verbosity levels are mapped to Catagory levels as following:

- *none*: level 0 (no Networking log)
- *severe*: level 1
- *warning*: level 2
- *info*: level 3
- *config*: level 3
- *fine*: level 4
- *finer*: level 5
- *finest*: level 6
- Full path: //OpenSplice/SNetworkService/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, finer, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.4.9 Compression

This group of attributes specifies a compression method to use within the service in partitions where it is enabled. The networking service includes (depending on platform) implementations of zlib, lzf and snappy. Others may be implemented by writing a dynamically-loadable library and configuring it here. See the Vortex OpenSplice release notes for details of how to write such a library.

It is imperative that all nodes exchanging compressed data have the same configuration in this section.

- Full path: //OpenSplice/SNetworkService/Compression
- Occurrences min-max: 0-1
- Optional attributes: PluginLibrary, PluginInitFunction, PluginParameter

12.4.9.1 PluginLibrary

This attribute names a dynamically loadable library which must contain the code for compressing and decompressing the network data. This may be left blank for the built-in compressors.

- Full path: //OpenSplice/SNetworkService/Compression[@PluginLibrary]
- Format: string
- Default value: “”
- Required: false

12.4.9.2 PluginInitFunction

This attribute specifies an initialization function for a compression plugin to be used within the service. The functions for the built-in compressors are named ospl_comp_zlib_init, ospl_comp_lzf_init and ospl_comp_snappy_init but for convenience they may be specified here as as zlib, lzf or snappy.

- Full path: //OpenSplice/SNetworkService/Compression[@PluginInitFunction]
- Format: string
- Default value: “”
- Required: false

12.4.9.3 PluginParameter

Some compression implementations are configurable with respect to the tradeoff between speed and effectiveness. A parameter may be specified here to control this. For example the zlib compressor is configured with an integer between 0 (for no compression) to 9 (for maximum compression).

- Full path: //OpenSplice/SNetworkService/Compression[@PluginParameter]
- Format: string
- Default value: “”
- Required: false

12.5 NetworkService

The Network Service provides a bridge between the local DDS service and a network interface. The Vortex 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 Applications which operate in multiple domains, 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 (Element Application). * <Service name=”networking”>

<Command>networking</Command>

</Service>*

The network configuration expects a root element named Vortex 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-*
- Required attributes: name

12.5.1 name

This attribute identifies the configuration for the Networking 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 *//OpenSplice/Domain/Service[@name]* in the configuration of the DomainService.

- Full path: //OpenSplice/NetworkService[@name]
- Format: string
- Default value: networking
- Required: true

12.5.2 Watchdog

This element controls the characteristics of the Watchdog thread.

- Full path: //OpenSplice/NetworkService/Watchdog
- Occurrences min-max: 0-1

12.5.2.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/NetworkService/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.5.2.1.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.5.3 General

This element contains general parameters that concern the networking service as a whole.

- Full path: //OpenSplice/NetworkService/General
- Occurrences min-max: 0-1
- Child elements: NetworkInterfaceAddress, EnableMulticastLoopback, LegacyCompression

12.5.3.1 NetworkInterfaceAddress

This element specifies which network interface card should be used. Every Networking service is bound to only one network interface card (NIC). 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, the Vortex OpenSplice middleware will try to look up an interface that has the required capabilities.

- Full path: //OpenSplice/NetworkService/General/NetworkInterfaceAddress
- Format: string
- Default value: first available
- Occurrences min-max: 0-1
- Optional attributes: forced, ipv6, bind, allowReuse

forced

This attribute specifies whether only the selected NetworkInterfaceAddress should be used or others can be used too.

- false - Specifies that the NetworkInterfaceAddress is first used but when not available another, when available, is used. (default).

- true - Specifies that only the selected NetworkInterfaceAddress can be used.

- Full path: //OpenSplice/NetworkService/General/NetworkInterfaceAddress[@forced]
- Format: boolean
- Default value: false
- Required: false

ipv6

This attribute specifies whether IPv6 should be used for communication.

- false - specifies that IPv4 should be used (default).
- true - Specifies that IPv6 should be used.

This setting will be overriden & ignored if the element NetworkInterfaceAddress has an explicit value that is unequivocally either an IPv4 or IPv6 address. This attribute is therefore only optionally required to specify IPv6 communication when special values like "first available" or an interface name are used instead of IP addresses.

- Full path: //OpenSplice/NetworkService/General/NetworkInterfaceAddress[@ipv6]
- Format: boolean
- Default value: false
- Required: false

bind

Specifies the bind strategy to be used by the networking service for its sockets.

- any - Specifies that the service should bind to the wildcard-address (INADDR_ANY) (default).
- strict - Specifies that the service should bind to the NetworkingInterfaceAddress.

- Full path: //OpenSplice/NetworkService/General/NetworkInterfaceAddress[@bind]
- Format: enumeration
- Default value: any
- Valid values: any, strict
- Required: false

allowReuse

By default the networking service will bind to a port allowing other services to bind to the same port as well (so reuse of the port is allowed). By setting this option to 'false', the port is bound exclusively (SO_REUSEADDR disabled).

- true - Ports can be reused (SO_REUSEADDR enabled) (default).
- false - Ports are bound exclusively.

- Full path: //OpenSplice/NetworkService/General/NetworkInterfaceAddress[@allowReuse]
- Format: boolean
- Default value: true
- Required: false

12.5.3.2 EnableMulticastLoopback

EnableMulticastLoopback specifies whether the networking service will allow IP multicast packets within the node to be visible to all networking participants in the node, including itself. It must be TRUE for intra-node multicast communications, but if a node runs only a single Vortex OpenSplice networking service and does not host any other networking-capable programs, it may be set to FALSE for improved performance.

- Full path: //OpenSplice/NetworkService/General/EnableMulticastLoopback
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.5.3.3 LegacyCompression

This element specifies if compression is applied after of before fragmentations. When set to TRUE compression is applied after fragmentation which is provided for backward compatibility. When set to FALSE compression is applied before fragmentation. The default value is TRUE.

- Full path: //OpenSplice/NetworkService/General/LegacyCompression
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.5.3.4 Reconnection

This element specifies the desired networking-behavior with respect to the validity of restoring lost connectivity with remote nodes. Here 'lost connectivity' means a prolonged inability to communicate with a known and still active remote node (typically because of network-issues) that has resulted in such a node 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 node 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
- Required attributes: allowed

allowed

This attribute specifies whether the network service must resume communication with an other 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. (default).
  - true - Specifies that the network service must resume communication.

- Full path: //OpenSplice/NetworkService/General/Reconnection[@allowed]
- Format: boolean
- Default value: false

- Required: true

12.5.4 Partitioning

The Vortex OpenSplice Networking 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 networking services in the system can be explicitly instructed to use specific unicast or multicast addresses for its networking traffic. This is done by means of so-called network partitions

A network partition is defined by one or more unicast, multicast of broadcast IP addresses. Any networking service that is started will read the network partition settings and, if applicable, join the required multicast groups. For every sample distributed by the networking service, both its partition and topic name will be inspected. In combination with a set of network partition mapping rules, the service will determine to which network partition the sample is written. The mapping rules are configurable as well.

Using networking configuration, nodes can be disconnected from any networking 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 on its turn is mapped onto a specific networking partition, which on its turn is only connected to those nodes that are capable of handling high volume data.

- Full path: //OpenSplice/NetworkService/Partitioning
- Occurrences min-max: 0-1

12.5.4.1 GlobalPartition

This element specifies the global or default networking 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
- Required attributes: Address
- Optional attributes: MulticastTimeToLive

Address

The global networking partition transports data that is either meant to be global, like discovery heartbeats, or that is not mapped onto any other networking partition. 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 are separated by a colon (,) semicolon (;) or space ( ). 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 notation or as the symbolic hostname, in which case the middleware 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: string
- Default value: broadcast
- Required: true

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/GlobalPartition[@MulticastTimeToLive]
- Default value: 32
- Valid values: 0 / 255
- Required: false

12.5.4.2 NetworkPartitions

Networking configuration can contain a set of networking partitions, which are grouped under the NetworkPartitions element.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions
- Occurrences min-max: 0-1

NetworkPartition

Every NetworkPartition has a name, an address and a connected flag.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition
- Occurrences min-max: 1-*
- Required attributes: Address
- Optional attributes: Name, Connected, Compression, SecurityProfile, MulticastTimeToLive

12.5.4.2.1.1 Name

A networking partition is uniquely identified by its name.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Name]
- Format: string
- Default value: networkPartition
- Required: false

12.5.4.2.1.2 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 are separated by a colon (,) semicolon (;) or space ( ). Samples for this network 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 notation or as the symbolic hostname, in which case the middleware will try to resolve the corresponding IP address.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Address]
- Format: string
- Default value: broadcast
- Required: true

12.5.4.2.1.3 Connected

A node can choose to be not connected to a networking partition by setting the Connected attribute.

If a node is connected to a networking partition, it will join the corresponding multicast group and it will receive data distributed over the partition. If it is not connected, data distributed over the partition will not reach the node but will be filtered by the networking interface or multicast enabled switches.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Connected]
- Format: boolean
- Default value: true
- Required: false

12.5.4.2.1.4 Compression

This attribute specifies if networking will apply compression to limit bandwidth for a specific network partition. This provides great flexibility as network partition are dynamically bind to logical partitions. Compression is performed before fragmentation of the messages. To provide backward compatibility the option LegacyCompression (see General options) can be set to provide compression after fragmentation. The following compression values are allowed:

- false - No compression is applied. This is also the default value if not specified.
- true - Compression is applicable

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@Compression]
- Format: boolean
- Default value: false
- Required: false

12.5.4.2.1.5 SecurityProfile

In the context of secure networking, the NetworkPartition element provides support for the attribute SecurityProfile. The attribute is referencing a security profile 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 Vortex 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 Vortex OpenSplice node in question.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@SecurityProfile]
- Format: string
- Default value: nullProfile
- Required: false

12.5.4.2.1.6 MulticastTimeToLive For each UDP packet sent out, The TimeToLive header value is set to this value for Multicast packets.

- Full path: //OpenSplice/NetworkService/Partitioning/NetworkPartitions/NetworkPartition[@MulticastTimeToLive]
- Default value: 32
- Valid values: 0 / 255

- Required: false

12.5.4.3 IgnoredPartitions

This element is used to group the set of IgnoredPartition elements.

- Full path: //OpenSplice/NetworkService/Partitioning/IgnoredPartitions
- Occurrences min-max: 0-1

IgnoredPartition

This element can be used to create a "Local Partition" that is only available on the node 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 Networking service.

- Full path: //OpenSplice/NetworkService/Partitioning/IgnoredPartitions/IgnoredPartition
- Occurrences min-max: 1-*
- Required attributes: DCPSPartitionTopic

12.5.4.3.1.1 DCPSPartitionTopic

The Networking 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: string
- Default value: .
- Required: true

12.5.4.4 PartitionMappings

This element is used to group the set of PartitionMapping elements.

- Full path: //OpenSplice/NetworkService/Partitioning/PartitionMappings
- Occurrences min-max: 0-1

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. Networking allows for a set of partition mappings to be defined.

- Full path: //OpenSplice/NetworkService/Partitioning/PartitionMappings/PartitionMapping
- Occurrences min-max: 1-*
- Required attributes: NetworkPartition, DCPSPartitionTopic

12.5.4.4.1.1 NetworkPartition

The NetworkPartition attribute of a partition mapping defines that networking partitition 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
- Default value: networkPartition
- Required: true

12.5.4.4.1.2 DCPSPartitionTopic

The Networking 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. 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/NetworkService/Partitioning/PartitionMappings/PartitionMapping[@DCPSPartitionTopic]
- Format: string
- Default value: .
- Required: true

12.5.5 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, reliability 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/NetworkService/Channels
- Occurrences min-max: 1-1
- Child elements: AllowedPorts

12.5.5.1 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/NetworkService/Channels/Channel
- Occurrences min-max: 1-42
- Child elements: PortNr, FragmentSize, Resolution, AdminQueueSize, CompressionBufferSize, CompressionThreshold, AllowedPorts
- Required attributes: name, reliable, enabled
- Optional attributes: default, priority

name

The name uniquely identifies the channel.

- Full path: //OpenSplice/NetworkService/Channels/Channel[@name]
- Format: string
- Default value: aChannel
- Required: true

reliable

If this attribute is set to true, the channel sends all messages reliably. If not, data is sent only once (fire-and-forget).

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.

- Full path: //OpenSplice/NetworkService/Channels/Channel[@reliable]
- Format: boolean
- Default value: false
- Required: true

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 networking 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.

Note that only one channel is allowed to have this attribute set to true

- Full path: //OpenSplice/NetworkService/Channels/Channel[@default]
- Format: boolean
- Default value: false
- Required: false

enabled

This attribute toggles a channel on or off. Toggling a channel between enabled and disabled is a quick alternative for commenting out the corresponding lines in the configuration file.

- Full path: //OpenSplice/NetworkService/Channels/Channel[@enabled]
- Format: boolean
- Default value: false
- Required: true

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 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/NetworkService/Channels/Channel[@priority]
- Format: integer
- Default value: 0
- Required: false

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: integer
- Default value: 53400
- Valid values: 1 / 65535
- Occurrences min-max: 1-1

FragmentSize

The networking module will fragment large message into smaller fragments with size **FragmentSize**. These fragments are sent as datagrams to the UDP stack. OS-settings determine the maximum datagram size.

- Full path: //OpenSplice/NetworkService/Channels/Channel/FragmentSize
- Default value: 1300
- Valid values: 200 / 65535
- Occurrences min-max: 0-1

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 ThrottleTreshold consistently throughout the system.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Resolution
- Format: integer
- Default value: 10
- Valid values: 1 / -
- Occurrences min-max: 0-1

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: integer
- Default value: 4000
- Valid values: 400 / -
- Occurrences min-max: 0-1

CompressionBufferSize

When compression on messages is enabled then the CompressionBufferSize specifies the initial size of the compression/decompression buffer. The compression buffer is used to store the messages before they are compressed and send on the network. The decompression buffer is used to decompress the received compressed messages. Note that the actual size of these buffers may be increased when needed.

- Full path: //OpenSplice/NetworkService/Channels/Channel/CompressionBufferSize
- Format: integer
- Default value: 131072
- Valid values: 65536 / -
- Occurrences min-max: 0-1

CompressionThreshold

When compression on messages is enabled then the CompressionThreshold provides a threshold to start compressing the accumulated data and sending the compressed data on the network. The CompressionThreshold is used to estimate the size of the compressed messages.

- Full path: //OpenSplice/NetworkService/Channels/Channel/CompressionThreshold
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

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: CrcCheck, QueueSize, MaxBurstSize, ThrottleLimit, ThrottleThreshold, MaxRetries, RecoveryFactor, DiffServField, DontRoute, DontFragment, TimeToLive, ReportInterval

12.5.5.1.12.1 CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.5.5.1.12.2 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/NetworkService/Channels/Channel/Sending/QueueSize
- Format: integer
- Default value: 400
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.5.5.1.12.3 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 considered "unlimited" as this far exceeds the capacity of modern physical networks.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/MaxBurstSize
- Default value: 1073741823
- Valid values: 1024 / 1073741823
- Occurrences min-max: 0-1

12.5.5.1.12.4 ThrottleLimit

Throttling will enable you to further limit (below MaxBurstSize) the amount of data that is sent every Resolution interval. This happens if one of the receiving nodes in the network 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 (or higher) as 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
- Default value: 10240
- Occurrences min-max: 0-1

12.5.5.1.12.5 ThrottleThreshold

This is the number of unprocessed network fragments that a node will store before it will inform the other nodes in the network that it has trouble processing the incoming data. Those other nodes 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 node 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: integer
- Default value: 50
- Valid values: 2 / -
- Occurrences min-max: 0-1

12.5.5.1.12.6 MaxRetries

The number of retransmissions the service has to execute before considering the addressed node as not responding.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/MaxRetries
- Format: integer
- Default value: 100
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.5.5.1.12.7 RecoveryFactor A lost message is resent after Resolution * RecoveryFactor milliseconds.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/RecoveryFactor
- Format: integer
- Default value: 3
- Valid values: 2 / -
- Occurrences min-max: 0-1

12.5.5.1.12.8 DiffServField

This element describes the DiffServ setting the channel will apply to the networking messages. This parameter determines the value of the diffserv field of the IP version 4 packets send on this channel which allows QoS setting to be applied to the network traffic send on this channel.

Windows platform support for setting the diffserv field is dependent on the OS version. For Windows versions XP SP2 and 2003 to use the diffserv field the following parameter should be added to the register:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting

The type of this parameter is a DWORD and it's value should be set to 0 to allow setting of the diffserv field.

For Windows version 7 or higher a new API (qWAVE) has been introduced For these platforms the specified diffserv value is mapped to one of the support traffic types. The mapping is as follows: 1-8 background traffic; 9-40 excellent traffic; 41-55 audio/video traffic; 56 voice traffic; 57-63 control traffic. When Vortex OpenSplice is run without Administrative priveleges then only the diffserv value of 0, 8, 40 or 56 is allowed.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/DiffServField

- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

12.5.5.1.12.9 DontRoute

The IP DONTROUTE socket option is set to the value specified.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/DontRoute
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

12.5.5.1.12.10 DontFragment

Controls whether the "don't fragment" bit (DF) is set on outgoing UDP IPv4 packets. Note that not all operating systems support setting this bit. When the operating system does not support setting the DF bit this option is ignored.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/DontFragment
- Format: boolean
- Default value: False
- Valid values: True, False
- Occurrences min-max: 0-1

12.5.5.1.12.11 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: integer
- Default value: 1
- Valid values: 1 / 255
- Occurrences min-max: 0-1

12.5.5.1.12.12 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: Priority, Class

12.5.5.1.12.13 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/Channels/Channel/Sending/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.5.5.1.12.14 priority_kind

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

- Full path: //OpenSplice/NetworkService/Channels/Channel/Sending/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.5.5.1.12.15 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/Channels/Channel/Sending/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

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: CrcCheck, ReceiveBufferSize, DefragBufferSize, MaxReliabBacklog, PacketRetentionPeriod, ReliabilityRecoveryPeriod

12.5.5.1.13.1 CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the receiving side is enabled only network packets that contain a valid crc field are accepted.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.5.5.1.13.2 ReceiveBufferSize

The UDP receive buffer of the best effort channel socket is set to the value given. If many message are lost, the receive buffer size has to be increased.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/ReceiveBufferSize
- Default value: 1000000
- Valid values: 1024 / -
- Occurrences min-max: 0-1

12.5.5.1.13.3 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: 0-1
- Child elements: Priority, Class, ReportInterval

12.5.5.1.13.4 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/Channels/Channel/Receiving/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.5.5.1.13.5 priority_kind

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

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.5.5.1.13.6 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.5.5.1.13.7 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.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/DefragBufferSize
- Default value: 5000
- Valid values: 500 / -
- Occurrences min-max: 0-1

12.5.5.1.13.8 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
- Required attributes: enabled

12.5.5.1.13.9 enabled

This attribute toggles the Optimization on or off.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/SMPOptimization[@enabled]
- Format: boolean
- Default value: true
- Required: true

12.5.5.1.13.10 MaxReliabBacklog

This is a lower limit to the DefragBufferSize that specifies the number of received fragments from a single remote node allocated for the purpose of order preservation because an earlier fragment from that remote node is missing. If this number is exceeded, then that particular remote node that didn't resend the missing fragent in time is considered dead for this channel.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/MaxReliabBacklog
- Format: integer
- Default value: 1000
- Valid values: 100 / -
- Occurrences min-max: 0-1

12.5.5.1.13.11 PacketRetentionPeriod

This element specifies the number of milliseconds 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) nodes, even though some nodes might not have received some packets at the moment a sending node disappears (for whatever reason). The protocol implies that each node retains sufficient received data so that it can be (re-)distributed if a publishing node disappears before all receiving nodes are "up-to-date". When the PacketRetentionPeriod element is set to 0 (the default value), the alignment amongst receiving nodes will not occur. To activate the extended realibility protocol, 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 networking services in the system.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/PacketRetentionPeriod
- Format: integer
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.5.5.1.13.12 ReliabilityRecoveryPeriod

This element specifies a timeout period (in milliseconds) for the alignment phase of the extended reliability protocol. 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 nodes (following the disappearance or crash of a publishing node) 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 nodes following the disappearance of a sending node in the system.

- Full path: //OpenSplice/NetworkService/Channels/Channel/Receiving/ReliabilityRecoveryPeriod
- Format: integer
- Default value: 1000
- Valid values: 0 / -
- Occurrences min-max: 0-1

AllowedPorts

AllowedPorts 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. When also 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 a comma (,). 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
- Occurrences min-max: 0-1

12.5.5.2 AllowedPorts

AllowedPorts 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. When also 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 a comma (,). 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/AllowedPorts
- Format: string
- Occurrences min-max: 0-1

12.5.6 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 (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.) 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 (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) (see Element ProbeList). 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 (The Global Partition contains all the addresses that the Network Service communicate with) 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 Attribute Scope) 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: PortNr, ProbeList
- Optional attributes: enabled, Scope

12.5.6.1 enabled

This element can be used to enable or disable the Discovery Channel. In case the Discovery Channel is disabled, entities will only detect each others presence implicitly once messages are received for the first time.

- Full path: //OpenSplice/NetworkService/Discovery[@enabled]
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

- Required: false

12.5.6.2 Scope

This attribute controls the dynamic discovery behaviour of this node within the current Domain. If it is not set, dynamic discovery will be disabled and the networking service will only communicate with nodes that can be reached through the predefined Global Partition. If the Scope attribute is specified, dynamic discovery is enabled and the networking service will be able to communicate with all nodes in the system that have a Role that matches the Scope expression. The Scope expression can contain a comma separated list of wild-card role-expressions. If the role of any discovered node matches any of the wild-card expressions, the remote node is considered a match and will become part of the communication reach (i.e. the Global Partition) of the current domain.

- Full path: //OpenSplice/NetworkService/Discovery[@Scope]
- Format: string
- Occurrences min-max: 0-1
- Required: false

12.5.6.3 PortNr

This element specifies the Port number used by the Discovery Channel.

- Full path: //OpenSplice/NetworkService/Discovery/PortNr
- Format: integer
- Default value: 3369
- Valid values: 1 / 65536
- Occurrences min-max: 1-1

12.5.6.4 ProbeList

This element contains the addresses of the nodes that will be contacted to retrieve an initial list of participating nodes in the current domain that match the specified Scope. Multiple ProbeList addresses can be entered by separating them by a colon (,), semicolon (;) or space( ) . The addresses can be entered as dotted decimal 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
- Occurrences min-max: 0-1

12.5.6.5 Sending

This element describes all properties for the transmitting side of the Discovery Channel.

- Full path: //OpenSplice/NetworkService/Discovery/Sending
- Occurrences min-max: 0-1
- Child elements: CrcCheck, DiffServField, DontRoute, DontFragment, TimeToLive, Interval, SafetyFactor, SalvoSize

CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

DiffServField

This element describes the DiffServ setting the channel will apply to the networking messages. This parameter determines the value of the diffserv field of the IP version 4 packets send on this channel which allows QoS setting to be applied to the network traffic send on this channel.

Windows platform support for setting the diffserv field is dependent on the OS version. For Windows versions XP SP2 and 2003 to use the diffserv field the following parameter should be added to the register:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting

The type of this parameter is a DWORD and it's value should be set to 0 to allow setting of the diffserv field.

For Windows version 7 or higher a new API (qWAVE) has been introduced For these platforms the specified diffserv value is mapped to one of the support traffic types. The mapping is as follows: 1-8 background traffic; 9-40 excellent traffic; 41-55 audio/video traffic; 56 voice traffic; 57-63 control traffic. When Vortex OpenSplice is run without Administrative priveleges then only the diffserv value of 0, 8, 40 or 56 is allowed.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/DiffServField
- Format: integer
- Default value: 0
- Valid values: 0 / 255
- Occurrences min-max: 0-1

DontRoute

The IP DONTROUTE socket option is set to the value specified.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/DontRoute
- Format: boolean
- Default value: True
- Valid values: True, False
- Occurrences min-max: 0-1

DontFragment

Controls whether the "don't fragment" bit (DF) is set on outgoing UDP IPv4 packets. Note that not all operating systems support setting this bit. When the operating system does not support setting the DF bit this option is ignored.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/DontFragment
- Format: boolean
- Default value: False
- Valid values: True, False
- Occurrences min-max: 0-1

TimeToLive

For each UDP packet sent out, the IP Time To Live header value is set to the value specified.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/TimeToLive
- Format: integer
- Default value: 1
- Valid values: 1 / 255
- Occurrences min-max: 0-1

Scheduling

This element specifies the scheduling policies used to control the transmitter thread of the Discovery Channel.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.5.6.5.6.1 Priority

This element specifies the thread priority that will be used by the transmitter thread of the Discovery Channel. 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.5.6.5.6.2 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

- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.5.6.5.6.3 Class

This element specifies the thread scheduling class that will be used by the transmitter thread of the Discovery Channel. 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

Interval

This element describes the interval(in milliseconds) at which remote nodes will expect heartbeats from this node.

- Full path: //OpenSplice/NetworkService/Discovery/Sending/Interval
- Format: integer
- Dimension: milliseconds
- Default value: 333
- Valid values: 10 / -
- Occurrences min-max: 0-1

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
- Default value: 0.9
- Valid values: 0.2 / 1.0
- Occurrences min-max: 0-1

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: integer
- Default value: 3
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.5.6.6 Receiving

This element describes all properties for the receiving side of the Discovery Channel.

- Full path: //OpenSplice/NetworkService/Discovery/Receiving
- Occurrences min-max: 0-1
- Child elements: CrcCheck, DeathDetectionCount, ReceiveBufferSize

CrcCheck

In order to protect Vortex OpenSplice network packets from malicious attack the CrcCheck(cyclic redundancy check) configuration item has been added. CRCs 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 using this feature due to the addtional overhead of carrying out the crc calculation.

When the sending side is enabled the network packet will contain a valid crc field.

- Full path: //OpenSplice/NetworkService/Discovery/Receiving/CrcCheck
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

Scheduling

This element specifies the scheduling policies used to control the receiver thread of the Discovery Channel.

- Full path: //OpenSplice/NetworkService/Discovery/Receiving/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

12.5.6.6.2.1 Priority

This element specifies the thread priority that will be used by the receiver thread of the Discovery Channel. 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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.5.6.6.2.2 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

12.5.6.6.2.3 Class

This element specifies the thread scheduling class that will be used by the receiver thread of the Discovery Channel. 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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

DeathDetectionCount

This element specifies how often a heartbeat from a remote node must miss its Interval before that remote node is considered dead.

- Full path: //OpenSplice/NetworkService/Discovery/Receiving/DeathDetectionCount
- Format: integer
- Default value: 6
- Valid values: 1 / -
- Occurrences min-max: 0-1

ReceiveBufferSize

The UDP receive buffer of the Discovery Channel socket 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
- Dimension: bytes
- Default value: 1000000
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.5.7 Tracing

This element controls the amount and type of information that is written into the tracing log by the Networking Service. This is useful to track the Networking 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: OutputFile, Timestamps, Verbosity
- Optional attributes: enabled

12.5.7.1 enabled

This attribute controls whether the tracing option is enabled or not.

- Full path: //OpenSplice/NetworkService/Tracing[@enabled]
- Format: boolean
- Default value: true
- Required: false

12.5.7.2 OutputFile

This option specifies where the logging is printed to. Note that "stdout" is considered a legal value that represents "standard out". The default value is an empty string, indicating that the tracing log will be written to standard out.

- Full path: //OpenSplice/NetworkService/Tracing/OutputFile
- Format: string
- Default value: networking.log
- Occurrences min-max: 1-1

12.5.7.3 Timestamps

This element specifies whether the logging must contain timestamps.

- Full path: //OpenSplice/NetworkService/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

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
- Default value: true
- Required: false

12.5.7.4 Categories

This element contains the logging properties for various networking categories.

- Full path: //OpenSplice/NetworkService/Tracing/Categories
- Occurrences min-max: 1-1
- Child elements: Default, Configuration, Construction, Destruction, Mainloop, Groups, Send, Receive, Throttling, Test, Discovery

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer

- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

Throttling

This element specifies the tracing level for the *Throttling* category. This includes information about throttling. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

- Full path: //OpenSplice/NetworkService/Tracing/Categories/Throttling
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

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: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

Discovery

This element specifies the tracing level for the *Discovery* category. This includes all activity related to the discovery channel. Level 0 indicates no tracing, level 6 indicates the most detailed level of tracing.

- Full path: //OpenSplice/NetworkService/Tracing/Categories/Discovery
- Format: integer
- Default value: 0
- Valid values: 0 / 6
- Occurrences min-max: 0-1

12.5.7.5 Verbosity

The Verbosity attribute sets the level of tracing for all Tracing Catagories. This attribute is an additional method besides the Tracing/Catagories tag to specify trace levels. The difference is that Verbosity sets the level for all catagories similar as by other services whereas the Tracing/Catagories element allows to set the trace level per catagory. The verbosity levels are mapped to Catagory levels as following:

- *none*: level 0 (no Networking log)
- *severe*: level 1
- *warning*: level 2

- *info*: level 3
- *config*: level 3
- *fine*: level 4
- *finer*: level 5
- *finest*: level 6
- Full path: //OpenSplice/NetworkService/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, finer, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.5.8 Compression

This group of attributes specifies a compression method to use within the service in partitions where it is enabled. The networking service includes (depending on platform) implementations of zlib, lzf and snappy. Others may be implemented by writing a dynamically-loadable library and configuring it here. See the Vortex OpenSplice release notes for details of how to write such a library.

It is imperative that all nodes exchanging compressed data have the same configuration in this section.

- Full path: //OpenSplice/NetworkService/Compression
- Occurrences min-max: 0-1
- Optional attributes: PluginLibrary, PluginInitFunction, PluginParameter

12.5.8.1 PluginLibrary

This attribute names a dynamically loadable library which must contain the code for compressing and decompressing the network data. This may be left blank for the built-in compressors.

- Full path: //OpenSplice/NetworkService/Compression[@PluginLibrary]
- Format: string
- Default value: “”
- Required: false

12.5.8.2 PluginInitFunction

This attribute specifies an initialization function for a compression plugin to be used within the service. The functions for the built-in compressors are named ospl_comp_zlib_init, ospl_comp_lzf_init and ospl_comp_snappy_init but for convenience they may be specified here as as zlib, lzf or snappy.

- Full path: //OpenSplice/NetworkService/Compression[@PluginInitFunction]
- Format: string
- Default value: “”
- Required: false

12.5.8.3 PluginParameter

Some compression implementations are configurable with respect to the tradeoff between speed and effectiveness. A parameter may be specified here to control this. For example the zlib compressor is configured with an integer between 0 (for no compression) to 9 (for maximum compression).

- Full path: //OpenSplice/NetworkService/Compression[@PluginParameter]
- Format: string
- Default value: “”
- Required: false

12.6 NetworkingBridgeService

The root element of a networking bridge service configuration.

- Full path: //OpenSplice/NetworkingBridgeService
- Occurrences min-max: 0-*
- Required attributes: name

12.6.1 name

This attribute identifies the configuration for the Networking Bridge Service. Multiple service configurations can be specified in one single XML file. The actual applicable configuration is determined by the value of the name attribute, which must match the specified under the element OpenSplice/Domain/Service[@name] in the Domain Service configuration.

- Full path: //OpenSplice/NetworkingBridgeService[@name]
- Format: string
- Default value: networkingbridge
- Required: true

12.6.2 Exclude

This element specifies which partition/topic combinations may not be forwarded.

- Full path: //OpenSplice/NetworkingBridgeService/Exclude
- Occurrences min-max: 0-*

12.6.2.1 Entry

This element configures a single partition/topic combination for exclusion in the set of forwarded partition/topic combinations.

- Full path: //OpenSplice/NetworkingBridgeService/Exclude/Entry
- Occurrences min-max: 0-*
- Required attributes: DCPSPartitionTopic

DCPSPartitionTopic

This attribute specifies a partition and a topic expression, separated by a single '.', that are used to determine if a given partition and topic will be excluded w.r.t. forwarding. The expressions may use the usual wildcards '*' and '?'.

- Full path: //OpenSplice/NetworkingBridgeService/Exclude/Entry[@DCPSPartitionTopic]
- Format: string
- Default value: n/a
- Required: true

12.6.3 Include

This element specifies which partition/topic combinations are to be forwarded, provided they are not listed in the Exclude section.

- Full path: //OpenSplice/NetworkingBridgeService/Include
- Occurrences min-max: 0-*

12.6.3.1 Entry

This element configures a single partition/topic combination for inclusion in the set of forwarded partition/topic combinations.

- Full path: //OpenSplice/NetworkingBridgeService/Include/Entry
- Occurrences min-max: 0-*
- Required attributes: DCPSPartitionTopic

DCPSPartitionTopic

This attribute specifies a partition and a topic expression, separated by a single '.', that are used to determine if a given partition and topic will be included w.r.t. forwarding. The expressions may use the usual wildcards '*' and '?'.

- Full path: //OpenSplice/NetworkingBridgeService/Include/Entry[@DCPSPartitionTopic]
- Format: string
- Default value: n/a
- Required: true

12.6.4 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/NetworkingBridgeService/Tracing
- Occurrences min-max: 0-*
- Child elements: AppendToFile, EnableCategory, OutputFile, Verbosity

12.6.4.1 AppendToFile

This option specifies whether the output is to be appended to an existing log file. The default is to create a new log file each time, which is generally the best option if a detailed log is generated.

- Full path: //OpenSplice/NetworkingBridgeService/Tracing/AppendToFile
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.6.4.2 EnableCategory

This element enables individual logging categories. These are enabled in addition to those enabled by Tracing/Verbosity. Recognised categories are: - *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

In addition, there is the keyword *trace* that enables all but *radmin*

- Full path: //OpenSplice/NetworkingBridgeService/Tracing/EnableCategory
- Format: string
- Occurrences min-max: 0-1

12.6.4.3 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/NetworkingBridgeService/Tracing/OutputFile
- Format: string
- Default value: nwbridge.log
- Occurrences min-max: 0-1

12.6.4.4 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. Recognised verbosity levels and the categories they map to are: - *none*: no NetworkingBridge log - *severe*: error and fatal - *warning*: *severe* + warning - *info*: *warning* + general information messages - *config*: *info* + config - *fine*: equivalent to *config* - *finest*: *fine* + trace

While *none* prevents any message from being written to a NetworkingBridge log file, warnings and errors are still logged in the ospl-info.log and ospl-error.log files.

- Full path: //OpenSplice/NetworkingBridgeService/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.6.5 Watchdog

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/NetworkingBridgeService/Watchdog
- Occurrences min-max: 0-*

12.6.5.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/NetworkingBridgeService/Watchdog/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

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/NetworkingBridgeService/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: default
- Valid values: realtime, timeshare, default
- Occurrences min-max: 0-1

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/NetworkingBridgeService/Watchdog/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1
- Optional attributes: priority_kind

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

- Full path: //OpenSplice/NetworkingBridgeService/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: relative
- Valid values: relative, absolute
- Required: false

12.7 DDSI2EService

The root element of a DDSI2E networking service configuration.

- Full path: //OpenSplice/DDSI2EService
- Occurrences min-max: 0-*
- Required attributes: name

12.7.1 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/Domain/Service[@name] in the Domain Service configuration.

- Full path: //OpenSplice/DDSI2EService[@name]
- Format: string
- Default value: ddsi2e
- Required: true

12.7.2 Channels

This element is used to group a set of channels. The channels are independent data paths through DDSI2E and by using separate threads and setting their priorities appropriately, chanenls can be used to map transport priorities to operating system scheduler priorities, ensuring system-wide end-to-end priority preservation.

- Full path: //OpenSplice/DDSI2EService/Channels
- Occurrences min-max: 0-1

12.7.2.1 Channel

This element defines a channel.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel
- Occurrences min-max: 0-42
- Child elements: AuxiliaryBandwidthLimit, DataBandwidthLimit, DiffServField, QueueSize
- Required attributes: Name
- Optional attributes: TransportPriority, Resolution

Name

This attribute specifies name of this channel. The name should uniquely identify the channel.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel[@Name]
- Format: string
- Default value: n/a
- Required: true

TransportPriority

This attribute sets the transport priority threshold for the channel. Each DCPS data writer has a "transport_priority" QoS and this QoS is used to select a channel for use by this writer. The selected channel is the one with the largest threshold not greater than the writer's transport priority, and if no such channel exists, the channel with the lowest threshold.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel[@TransportPriority]
- Format: integer
- Default value: 0
- Required: false

AuxiliaryBandwidthLimit

This element specifies the maximum transmit rate of auxiliary traffic on this channel (e.g. retransmits, heartbeats, etc). Bandwidth limiting uses a leaky bucket scheme. The default value "inf" means DDSI2E imposes no limitation, the underlying operating system and hardware will likely limit the maimum transmit rate.

The unit must be specified explicitly. Recognised units: *X*b/s, *X*bps for bits/s or *X*B/s, *X*Bps for bytes/s; where *X* is an optional prefix: k for 10^3, Ki for 2^{10}, M for 10^6, Mi for 2^{20}, G for 10^9, Gi for 2^{30}.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel/AuxiliaryBandwidthLimit
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

DataBandwidthLimit

This element specifies the maximum transmit rate of new samples and directly related data, for this channel. Bandwidth limiting uses a leaky bucket scheme. The default value "inf" means DDSI2E imposes no limitation, the underlying operating system and hardware will likely limit the maimum transmit rate.

The unit must be specified explicitly. Recognised units: *X*b/s, *X*bps for bits/s or *X*B/s, *X*Bps for bytes/s; where *X* is an optional prefix: k for 10^3, Ki for 2^{10}, M for 10^6, Mi for 2^{20}, G for 10^9, Gi for 2^{30}.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel/DataBandwidthLimit
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

DiffServField

This element describes the DiffServ setting the channel will apply to the networking messages. This parameter determines the value of the diffserv field of the IP version 4 packets sent on this channel which allows QoS setting to be applied to the network traffic send on this channel.

Windows platform support for setting the diffserv field is dependent on the OS version.

For Windows versions XP SP2 and 2003 to use the diffserv field the following parameter should be added to the register:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting

The type of this parameter is a DWORD and its value should be set to 0 to allow setting of the diffserv field.

For Windows version 7 or higher a new API (qWAVE) has been introduced. For these platforms the specified diffserv value is mapped to one of the support traffic types. The mapping is as follows: 1-8 background traffic; 9-40 excellent traffic; 41-55 audio/video traffic; 56 voice traffic; 57-63 control traffic. When an application is run without Administrative priveleges then only the diffserv value of 0, 8, 40 or 56 is allowed.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel/DiffServField
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

QueueSize

This element specifies the number of messages the network queue for this channel can contain. The Vortex OpenSplice kernel writes data to be transmitted to the network queue, and DDSI2E takes them from this queue. If this queue is full when an application tries to write a sample, the sample will be dropped or the writer suspended, depending on the QoS settings of the writer. Vortex OpenSplice and its services are optimised for a well-balanced system design, where the queue never becomes full.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel/QueueSize
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

Resolution

This element specifies the interval at which the DDSI2E transmit thread for this channel wakes up, and which controls the smallest latency_budget that has an effect. A shorter latency_budget is rounded to 0. The downside of a reducing this setting is that it increases the number of idle wake-ups of the transmit thread when there is no data to be sent.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Channels/Channel[@Resolution]
- Format: string
- Default value: 1s
- Valid values: 0 / 1s
- Required: false

12.7.3 Compatibility

The Compatibility elements allows specifying various settings related to compatability with standards and with other DDSI implementations.

- Full path: //OpenSplice/DDSI2EService/Compatibility
- Occurrences min-max: 0-1
- Child elements: AckNackNumbitsEmptySet, ArrivalOfDataAssertsPpAndEpLiveliness, AssumeRtiHasPmdEndpoints, ExplicitlyPublishQosSetToDefault, ManySocketsMode, RespondToRtiInitZeroAckWithInvalidHeartbeat, StandardsConformance

12.7.3.1 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 reject by the strict and pedantic conformance modes, but several other implementation require this setting for smooth interoperation.

If set to 1, all acknowledgements sent by DDSI2E adhere 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 an AckNackNumbitsEmptySet=0 to prevent the transmitting of invalid messages.

- Full path: //OpenSplice/DDSI2EService/Compatibility/AckNackNumbitsEmptySet
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.3.2 ArrivalOfDataAssertsPpAndEpLiveliness

When set to true, arrival of a message from a peer asserts liveliness of that peer. When set to false, only SPDP and explicit lease renewals have this effect.

- Full path: //OpenSplice/DDSI2EService/Compatibility/ArrivalOfDataAssertsPpAndEpLiveliness
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.3.3 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/DDSI2EService/Compatibility/AssumeRtiHasPmdEndpoints
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.3.4 ExplicitlyPublishQosSetToDefault

This element 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 significant 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/DDSI2EService/Compatibility/ExplicitlyPublishQosSetToDefault
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.3.5 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 will ensure this to be the case. This is not the defeault.

Disabling it slightly improves performance and reduces network traffic somewhat. It also causes the set of port numbers needed by DDSI2E to become predictable, which may be useful for firewall and NAT configuration.

- Full path: //OpenSplice/DDSI2EService/Compatibility/ManySocketsMode
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.3.6 RespondToRtiInitZeroAckWithInvalidHeartbeat

This element 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/DDSI2EService/Compatibility/RespondToRtiInitZeroAckWithInvalidHeartbeat
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.3.7 StandardsConformance

This element sets the level of standards conformance of this instance of the DDSI2E Service. Stricter conformance typically means less interoperability with other implementations. Currently three modes are defined:

- *pedantic*: very strictly conform 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*: a slightly less strict view of the standard than does pedantic: it follows the established behaviour where the standard is obviously in error.
- *lax*: attempt 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/DDSI2EService/Compatibility/StandardsConformance
- Format: enumeration
- Default value: lax
- Valid values: lax, strict, pedantic
- Occurrences min-max: 0-1

12.7.4 Discovery

The Discovery element allows specifying various parameters related to the discovery of peers.

- Full path: //OpenSplice/DDSI2EService/Discovery
- Occurrences min-max: 0-1
- Child elements: AdvertiseBuiltinTopicWriters, DSGracePeriod, DefaultMulticastAddress, DomainId, GenerateBuiltinTopics, LocalDiscoveryPartition, MaxAutoParticipantIndex, ParticipantIndex, SPDPInterval, SPDPMulticastAddress

12.7.4.1 AdvertiseBuiltinTopicWriters

This element controls whether or not DDSI2E advertises writers for the built-in topics from its discovery for backwards compatibility with older Vortex OpenSplice versions.

- Full path: //OpenSplice/DDSI2EService/Discovery/AdvertiseBuiltinTopicWriters
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.4.2 DSGracePeriod

This setting controls for how long endpoints discovered via a Cloud discovery service will survive after the discovery service disappeared, allowing reconnect without loss of data when the discovery service restarts (or another instance takes over).

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Discovery/DSGracePeriod
- Format: string
- Default value: 30 s
- Occurrences min-max: 0-1

12.7.4.3 DefaultMulticastAddress

This element specifies the default multicast address for all traffic other than participant discovery packets. It defaults to Discovery/SPDPMulticastAddress.

- Full path: //OpenSplice/DDSI2EService/Discovery/DefaultMulticastAddress
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.7.4.4 DomainId

This element allows overriding of the DDS Domain Id that is used for DDSI2E.

- Full path: //OpenSplice/DDSI2EService/Discovery/DomainId
- Format: string
- Default value: default

- Occurrences min-max: 0-1

12.7.4.5 GenerateBuiltinTopics

This element controls whether or not DDSI2E generates built-in topics from its discovery. When disabled, it relies on the durability service.

- Full path: //OpenSplice/DDSI2EService/Discovery/GenerateBuiltinTopics
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.4.6 LocalDiscoveryPartition

This element controls which partition is monitored by DDSI2E for built-in topics describing entities the it mirrors in DDSI.

- Full path: //OpenSplice/DDSI2EService/Discovery/LocalDiscoveryPartition
- Format: string
- Default value: __BUILT-IN PARTITION__
- Occurrences min-max: 0-1

12.7.4.7 MaxAutoParticipantIndex

This element specifies the maximum DDSI participant index selected by this instance of the DDSI2E service if the Discovery/ParticipantIndex is "auto".

- Full path: //OpenSplice/DDSI2EService/Discovery/MaxAutoParticipantIndex
- Format: integer
- Default value: 9
- Occurrences min-max: 0-1

12.7.4.8 ParticipantIndex

This element specifies the DDSI participant index used by this instance of the DDSI2E 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*: which will attempt to automatically determine an available participant index (see also Discovery/MaxAutoParticipantIndex), 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.

The default is *auto*. The participant index is part of the port number calculation and if predictable port numbers are needed and fixing the participant index has no adverse effects, it is recommended that the second be option be used.

- Full path: //OpenSplice/DDSI2EService/Discovery/ParticipantIndex
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.7.4.9 Peers

This element statically configures addresses for discovery.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers
- Occurrences min-max: 0-1

Group

This element statically configures a fault tolerant group of addresses for discovery. Each member of the group is tried in sequence until one succeeds.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers/Group
- Occurrences min-max: 0-*

12.7.4.9.1.1 Peer This element statically configures an addresses for discovery.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers/Group/Peer
- Occurrences min-max: 0-*
- Required attributes: Address

12.7.4.9.1.2 Address This element specifies an IP address to which discovery packets must be sent, in addition to the default multicast address (see also General/AllowMulticast). Both a hostnames and a numerical IP address is accepted; the hostname or IP address may be suffixed with :PORT to explicitly set the port to which it must be sent. Multiple Peers may be specified.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers/Group/Peer[@Address]
- Format: string
- Default value: n/a
- Required: true

Peer

This element statically configures an addresses for discovery.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers/Peer
- Occurrences min-max: 0-*
- Required attributes: Address

12.7.4.9.2.1 Address This element specifies an IP address to which discovery packets must be sent, in addition to the default multicast address (see also General/AllowMulticast). Both a hostnames and a numerical IP address is accepted; the hostname or IP address may be suffixed with :PORT to explicitly set the port to which it must be sent. Multiple Peers may be specified.

- Full path: //OpenSplice/DDSI2EService/Discovery/Peers/Peer[@Address]
- Format: string
- Default value: n/a
- Required: true

12.7.4.10 Ports

The Ports element allows specifying various parameters related to the port numbers used for discovery. These all have default values specified by the DDSI 2.1 (and 2.2) specification and rarely need to be changed.

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports
- Occurrences min-max: 0-1
- Child elements: Base, DomainGain, MulticastDataOffset, MulticastMetaOffset, ParticipantGain, UnicastDataOffset, UnicastMetaOffset

Base

This element specifies the base port number (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant PB).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/Base
- Format: integer
- Default value: 7400
- Valid values: 1 / 65535
- Occurrences min-max: 0-1

DomainGain

This element specifies the domain gain, relating domain ids to sets of port numbers (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant DG).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/DomainGain
- Format: integer
- Default value: 250
- Occurrences min-max: 0-1

MulticastDataOffset

This element specifies the port number for multicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d2).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/MulticastDataOffset
- Format: integer
- Default value: 1
- Occurrences min-max: 0-1

MulticastMetaOffset

This element specifies the port number for multicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d0).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/MulticastMetaOffset
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

ParticipantGain

This element specifies the participant gain, relating p0, articipant index to sets of port numbers (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant PG).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/ParticipantGain
- Format: integer
- Default value: 2
- Occurrences min-max: 0-1

UnicastDataOffset

This element specifies the port number for unicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d3).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/UnicastDataOffset
- Format: integer
- Default value: 11
- Occurrences min-max: 0-1

UnicastMetaOffset

This element specifies the port number for unicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d1).

- Full path: //OpenSplice/DDSI2EService/Discovery/Ports/UnicastMetaOffset
- Format: integer
- Default value: 10
- Occurrences min-max: 0-1

12.7.4.11 SPDPInterval

This element specifies the interval between spontaneous transmissions of participant discovery packets.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Discovery/SPDPInterval
- Format: string
- Default value: 30 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.4.12 SPDPMulticastAddress

This element specifies the multicast address that is used as the destination for the participant discovery packets. In IPv4 mode the default is the (standardised) 239.255.0.1, in IPv6 mode it becomes ff02::ffff:239.255.0.1, which is a non-standardised link-local multicast address.

- Full path: //OpenSplice/DDSI2EService/Discovery/SPDPMulticastAddress
- Format: string

- Default value: 239.255.0.1
- Occurrences min-max: 0-1

12.7.5 General

The General element specifies overall DDSI2E service settings.

- Full path: //OpenSplice/DDSI2EService/General
- Occurrences min-max: 0-1
- Child elements: AllowMulticast, CoexistWithNativeNetworking, DontRoute, EnableMulticastLoopback, ExternalNetworkAddress, ExternalNetworkMask, FragmentSize, MaxMessageSize, MulticastRecvNetworkInterfaceAddresses, MulticastTimeToLive, NetworkInterfaceAddress, StartupModeCoversTransient, StartupModeDuration, UseIPv6

12.7.5.1 AllowMulticast

This element controls whether DDSI2E uses multicasts for data traffic.

It is a comma-separated list of some of the following keywords: "spdp", "asm", "ssm", or either of "false" or "true".

- *spdp*: enables the use of ASM (any-source multicast) for participant discovery
- *asm*: enables the use of ASM for all traffic (including SPDP)
- *ssm*: enables the use of SSM (source-specific multicast) for all non-SPDP traffic (if supported)

When set to "false" all multicasting is disabled. The default, "true" enables full use of multicasts. Listening for multicasts can be controlled by General/MulticastRecvNetworkInterfaceAddresses.

- Full path: //OpenSplice/DDSI2EService/General/AllowMulticast
- Format: string
- Default value: true
- Occurrences min-max: 0-1

12.7.5.2 CoexistWithNativeNetworking

This element specifies whether the DDSI2E service operates in conjunction with the Vortex OpenSplice RT Networking service. When "false", the DDSI2E service will take care of all communications, including those between Vortex OpenSplice nodes; when "true", the DDSI2E service only communicates with DDS implementations other than Vortex OpenSplice. In this case the RT Networking service should be configured as well.

- Full path: //OpenSplice/DDSI2EService/General/CoexistWithNativeNetworking
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.5.3 DontRoute

This element allows setting the SO_DONTROUTE option for outgoing packets, to bypass the local routing tables. This is generally useful only when the routing tables cannot be trusted, which is highly unusual.

- Full path: //OpenSplice/DDSI2EService/General/DontRoute
- Format: boolean

- Default value: false
- Occurrences min-max: 0-1

12.7.5.4 EnableMulticastLoopback

This element specifies whether DDSI2E allows IP multicast packets to be visible to all DDSI participants in the same node, including itself. It must be "true" for intra-node multicast communications, but if a node runs only a single DDSI2E service and does not host any other DDSI-capable programs, it should be set to "false" for improved performance.

- Full path: //OpenSplice/DDSI2EService/General/EnableMulticastLoopback
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.5.5 ExternalNetworkAddress

This element allows explicitly overruling the network address DDSI2E advertises in the discovery protocol, which by default is the address of the preferred network interface (General/NetworkInterfaceAddress), to allow DDSI2E to communicate across a Network Address Translation (NAT) device.

- Full path: //OpenSplice/DDSI2EService/General/ExternalNetworkAddress
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.7.5.6 ExternalNetworkMask

This element specifies the network mask of the external network address. This element is relevant only when an external network address (General/ExternalNetworkAddress) is explicitly configured. In this case locators received via the discovery protocol that are within the same external subnet (as defined by this mask) will be translated to an internal address by replacing the network portion of the external address with the corresponding portion of the preferred network interface address. This option is IPv4-only.

- Full path: //OpenSplice/DDSI2EService/General/ExternalNetworkMask
- Format: string
- Default value: 0.0.0.0
- Occurrences min-max: 0-1

12.7.5.7 FragmentSize

This element specifies the size of DDSI sample fragments generated by DDSI2E. Samples larger than FragmentSize are fragmented into fragments of FragmentSize bytes each, except the last one, which may be smaller. The DDSI spec mandates a minimum fragment size of 1025 bytes, but DDSI2E will do whatever size is requested, accepting fragments of which the size is at least the minimum of 1025 and FragmentSize.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/General/FragmentSize
- Format: string
- Default value: 1280 B

- Occurrences min-max: 0-1

12.7.5.8 MaxMessageSize

This element specifies the maximum size of the UDP payload that DDSI2E will generate. DDSI2E will try to maintain this limit within the bounds of the DDSI specification, which means that in some cases (especially for very low values of MaxMessageSize) larger payloads may sporadically be observed (currently up to 1192 B).

On some networks it may be necessary to set this item to keep the packetsize below the MTU to prevent IP fragmentation. In those cases, it is generally advisable to also consider reducing Internal/FragmentSize.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/General/MaxMessageSize
- Format: string
- Default value: 4096 B
- Occurrences min-max: 0-1

12.7.5.9 MulticastRecvNetworkInterfaceAddresses

This element specifies on which network interfaces DDSI2E listens to multicasts. The following options are available:

- *all*: listen for multicasts on all multicast-capable interfaces; or
- *any*: listen for multicasts on the operating system default interface; or
- *preferred*: listen for multicasts on the preferred interface (General/NetworkInterfaceAddress); or
- *none*: does not listen for multicasts on any interface; or
- a comma-separated list of network addresses: configures DDSI2E to listen for multicasts on all of the listed addresses.

If DDSI2E is in IPv6 mode and the address of the preferred network interface is a link-local address, "all" is treated as a synonym for "preferred" and a comma-separated list is treated as "preferred" if it contains the preferred interface and as "none" if not.

- Full path: //OpenSplice/DDSI2EService/General/MulticastRecvNetworkInterfaceAddresses
- Format: string
- Default value: preferred
- Occurrences min-max: 0-1

12.7.5.10 MulticastTimeToLive

This element specifies the time-to-live setting for outgoing multicast packets.

- Full path: //OpenSplice/DDSI2EService/General/MulticastTimeToLive
- Format: integer
- Default value: 32
- Valid values: 0 / 255
- Occurrences min-max: 0-1

12.7.5.11 NetworkInterfaceAddress

This element specifies the preferred network interface for use by DDSI2E. The preferred network interface determines the IP address that DDSI2E advertises in the discovery protocol (but see also General/ExternalNetworkAddress), and is also the only interface over which multicasts are transmitted. The interface can be identified by its IP address, network interface name or network portion of the address. If the value "auto" is entered here, DDSI2E will select what it considers the most suitable interface.

- Full path: //OpenSplice/DDSI2EService/General/NetworkInterfaceAddress
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.7.5.12 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. Configurations without defined merge policies best leave this enabled.

- Full path: //OpenSplice/DDSI2EService/General/StartupModeCoversTransient
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.5.13 StartupModeDuration

This element specifies how long the DDSI2E 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, allowing existing readers on remote nodes to obtain 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, DDSI2E keeps track of the existence of remote readers whether or not matching writers exist locally, avoiding this discovery delay and ensuring this is merely a node startup issue.

Setting General/StartupModeDuration to 0s will disable it.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/General/StartupModeDuration
- Format: string
- Default value: 2 s
- Valid values: 0 / 60000
- Occurrences min-max: 0-1

12.7.5.14 UseIPv6

This element can be used to DDSI2E use IPv6 instead of IPv4. This is currently an either/or switch.

- Full path: //OpenSplice/DDSI2EService/General/UseIPv6
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6 Internal

The Internal elements deal with a variety of settings that evolving and that are not necessarily fully supported. For the vast majority of the Internal settings, the functionality per-se is supported, but the right to change the way the options control the functionality is reserved. This includes renaming or moving options.

- Full path: //OpenSplice/DDSI2EService/Internal
- Occurrences min-max: 0-1
- Child elements: AccelerateRexmitBlockSize, AggressiveKeepLastWhc, AggressiveKeepLastWhc, AssumeMulticastCapable, AutoReschedNackDelay, AuxiliaryBandwidthLimit, BuiltinEndpointSet, ConservativeBuiltinReaderStartup, DDSI2DirectMaxThreads, DefragReliableMaxSamples, DefragUnreliableMaxSamples, DeliveryQueueMaxSamples, ForwardAllMessages, ForwardRemoteData, GenerateKeyhash, HeartbeatInterval, LateAckMode, LeaseDuration, LegacyFragmentation, LogStackTraces, MaxParticipants, MaxQueuedRexmitBytes, MaxQueuedRexmitMessages, MaxSampleSize, MeasureHbToAckLatency, MinimumSocketReceiveBufferSize, MinimumSocketSendBufferSize, MirrorRemoteEntities, MonitorPort, NackDelay, PreEmptiveAckDelay, PrimaryReorderMaxSamples, PrioritizeRetransmit, RediscoveryBlacklistDuration, ResponsivenessTimeout, RetransmitMerging, RetransmitMergingPeriod, RetryOnRejectBestEffort, RetryOnRejectDuration, SPDPResponseMaxDelay, ScheduleTimeRounding, SecondaryReorderMaxSamples, SquashParticipants, SuppressSPDPMulticast, SynchronousDeliveryLatencyBound, SynchronousDeliveryPriorityThreshold, UnicastResponseToSPDPMessages, UseMulticastIfMreqn, WriterLingerDuration

12.7.6.1 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/DDSI2EService/Internal/AccelerateRexmitBlockSize
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.6.2 AggressiveKeepLastWhc

Internal This element controls whether to drop a reliable sample from a DDSI2E WHC before all readers have acknowledged it as soon as a later sample becomes available. It only affects DCPS data writers with a history QoS setting of KEEP_LAST with depth 1. The default setting, *false*, mimics the behaviour of the Vortex OpenSplice RT networking and is necessary to make the behaviour of wait_for_acknowledgements() consistent across the networking services.

- Full path: //OpenSplice/DDSI2EService/Internal/AggressiveKeepLastWhc
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.6.3 AggressiveKeepLastWhc

Internal This element controls whether to drop a reliable sample from a DDSI2E WHC before all readers have acknowledged it as soon as a later sample becomes available. It only affects DCPS data writers with a history QoS setting of KEEP_LAST with depth 1. The default setting, *false*, mimics the behaviour of the Vortex OpenSplice RT networking and is necessary to make the behaviour of wait_for_acknowledgements() consistent across the networking services.

- Full path: //OpenSplice/DDSI2EService/Internal/AggressiveKeepLastWhc
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.6.4 AssumeMulticastCapable

Internal This element controls which network interfaces are assumed to be capable of multicasting even when the interface flags returned by the operating system state it is not (this provides a workaround for some platforms). It is a comma-separated lists of patterns (with ? and * wildcards) against which the interface names are matched.

- Full path: //OpenSplice/DDSI2EService/Internal/AssumeMulticastCapable
- Format: string
- Occurrences min-max: 0-1

12.7.6.5 AutoReschedNackDelay

Internal This setting controls the interval with which a reader will continue NACK'ing missing samples in the absence of a response from the writer, as a protection mechanism against writers incorrectly stopping the sending of HEARTBEAT messages.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/AutoReschedNackDelay
- Format: string
- Default value: 1 s
- Occurrences min-max: 0-1

12.7.6.6 AuxiliaryBandwidthLimit

Internal This element specifies the maximum transmit rate of auxiliary traffic not bound to a specific channel, such as discovery traffic, as well as auxiliary traffic related to a certain channel if that channel has elected to share this global AuxiliaryBandwidthLimit. Bandwidth limiting uses a leaky bucket scheme. The default value "inf" means DDSI2E imposes no limitation, the underlying operating system and hardware will likely limit the maimum transmit rate.

The unit must be specified explicitly. Recognised units: *X*b/s, *X*bps for bits/s or *X*B/s, *X*Bps for bytes/s; where *X* is an optional prefix: k for 10^3, Ki for 2^{10}, M for 10^6, Mi for 2^{20}, G for 10^9, Gi for 2^{30}.

- Full path: //OpenSplice/DDSI2EService/Internal/AuxiliaryBandwidthLimit
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

12.7.6.7 BuiltinEndpointSet

Internal This element 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. See also Internal/ConservativeBuiltinReaderStartup.

- Full path: //OpenSplice/DDSI2EService/Internal/BuiltinEndpointSet
- Format: enumeration
- Default value: writers
- Valid values: full, writers, minimal
- Occurrences min-max: 0-1

12.7.6.8 ConservativeBuiltinReaderStartup

Internal This element forces all DDSI2E built-in discovery-related readers to request all historical data, instead of just one for each "topic". There is no indication that any of the current DDSI implementations requires changing of this setting, but it is conceivable that an implementation might track which participants have been informed of the existence of endpoints and which have not been, refusing communication with those that have "can't" know.

Should it be necessary to hide DDSI2E's shared discovery behaviour, set this to *true* and Internal/BuiltinEndpointSet to *full*.

- Full path: //OpenSplice/DDSI2EService/Internal/ConservativeBuiltinReaderStartup
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.9 ControlTopic

Internal The ControlTopic element allows configured whether DDSI2E provides a special control interface via a predefined topic or not.

- Full path: //OpenSplice/DDSI2EService/Internal/ControlTopic
- Occurrences min-max: 0-1
- Child elements: Deaf, Mute
- Optional attributes: enable, initialreset

enable

Internal This attribute controls whether the DDSI2E control topic is defined and acted upon by DDSI2

- Full path: //OpenSplice/DDSI2EService/Internal/ControlTopic[@enable]
- Format: boolean
- Default value: false
- Required: false

initialreset

Internal This attribute sets the time until the deaf and mute settings are automatically reset to false

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/ControlTopic[@initialreset]
- Format: string
- Default value: inf
- Required: false

Deaf

Internal This element controls whether DDSI2E defaults to deaf mode or to normal mode. This controls both the initial behaviour and what behaviour it auto-reverts to.

- Full path: //OpenSplice/DDSI2EService/Internal/ControlTopic/Deaf
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

Mute

Internal This element controls whether DDSI2E defaults to mute mode or to normal mode. This controls both the initial behaviour and what behaviour it auto-reverts to.

- Full path: //OpenSplice/DDSI2EService/Internal/ControlTopic/Mute
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.10 DDSI2DirectMaxThreads

Internal This element sets the maximum number of extra threads for an experimental, undocumented and unsupported direct mode.

- Full path: //OpenSplice/DDSI2EService/Internal/DDSI2DirectMaxThreads
- Format: integer
- Default value: 1
- Occurrences min-max: 0-1

12.7.6.11 DefragReliableMaxSamples

Internal This element sets the maximum number of samples that can be defragmented simultaneously for a reliable writer. This has to be large enough to handle retransmissions of historical data in addition to new samples.

- Full path: //OpenSplice/DDSI2EService/Internal/DefragReliableMaxSamples
- Format: integer
- Default value: 16
- Occurrences min-max: 0-1

12.7.6.12 DefragUnreliableMaxSamples

Internal This element sets the maximum number of samples that can be defragmented simultaneously for a best-effort writers.

- Full path: //OpenSplice/DDSI2EService/Internal/DefragUnreliableMaxSamples
- Format: integer
- Default value: 4
- Occurrences min-max: 0-1

12.7.6.13 DeliveryQueueMaxSamples

Internal This element controls the Maximum size of a delivery queue, expressed in samples. Once a delivery queue is full, incoming samples destined for that queue are dropped until space becomes available again.

- Full path: //OpenSplice/DDSI2EService/Internal/DeliveryQueueMaxSamples
- Format: integer
- Default value: 256
- Occurrences min-max: 0-1

12.7.6.14 ForwardAllMessages

Internal Forward all messages from a writer, rather than trying to forward each sample only once. The default of trying to forward each sample only once filters out duplicates for writers in multiple partitions under nearly all circumstances, but may still publish the odd duplicate. Note: the current implementation also can lose in contrived test cases, that publish more than 2**32 samples using a single data writer in conjunction with carefully controlled management of the writer history via cooperating local readers.

- Full path: //OpenSplice/DDSI2EService/Internal/ForwardAllMessages
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.15 ForwardRemoteData

Internal This element controls whether DDSI2E forwards data received from other network services in the domain.

- Full path: //OpenSplice/DDSI2EService/Internal/ForwardRemoteData
- Format: enumeration
- Default value: default
- Valid values: false, true, default
- Occurrences min-max: 0-1

12.7.6.16 GenerateKeyhash

Internal When true, include keyhashes in outgoing data for topics with keys.

- Full path: //OpenSplice/DDSI2EService/Internal/GenerateKeyhash
- Format: boolean

- Default value: true
- Occurrences min-max: 0-1

12.7.6.17 HeartbeatInterval

Internal This element sets the base interval for the asynchronous, periodic writer heartbeats when unacknowledged data is present in its writer history cache. The actual interval is dynamically adjusted, the attributes of this element allow further configuration.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/HeartbeatInterval
- Format: string
- Default value: 100 ms
- Occurrences min-max: 0-1
- Optional attributes: max, min, minsched

max

Internal This attribute sets the maximum interval for periodic heartbeats.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/HeartbeatInterval[@max]
- Format: string
- Default value: 8 s
- Required: false

min

Internal This attribute sets the minimum interval that must have passed since the most recent heartbeat from a writer, before another asynchronous (not directly related to writing) will be sent.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/HeartbeatInterval[@min]
- Format: string
- Default value: 5 ms
- Required: false

minsched

Internal This attribute sets the minimum interval for periodic heartbeats. Other events may still cause heartbeats to go out.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/HeartbeatInterval[@minsched]
- Format: string

- Default value: 20 ms
- Required: false

12.7.6.18 LateAckMode

Internal Ack a sample only when it has been delivered, instead of when committed to delivering it.

- Full path: //OpenSplice/DDSI2EService/Internal/LateAckMode
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.19 LeaseDuration

Internal This setting controls the default participant lease duration. The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/LeaseDuration
- Format: string
- Default value: 0 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.6.20 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 DDSI2E for large messages.

- Full path: //OpenSplice/DDSI2EService/Internal/LegacyFragmentation
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.21 LogStackTraces

Internal This element controls whether or not to write stack traces to the DDSI2 trace when a thread fails to make progress (on select platforms only).

- Full path: //OpenSplice/DDSI2EService/Internal/LogStackTraces
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.6.22 MaxParticipants

Internal This elements configures the maximum number of DCPS domain participants this DDSI2E instance is willing to service. 0 is unlimited.

- Full path: //OpenSplice/DDSI2EService/Internal/MaxParticipants
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.6.23 MaxQueuedRexmitBytes

Internal This setting limits the maximum number of bytes queued for retransmission. The default value of 0 is unlimited unless an AuxiliaryBandwidthLimit has been set, in which case it becomes NackDelay * AuxiliaryBandwidthLimit. It must be large enough to contain the largest sample that may need to be retransmitted.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/MaxQueuedRexmitBytes
- Format: string
- Default value: 50 kB
- Occurrences min-max: 0-1

12.7.6.24 MaxQueuedRexmitMessages

Internal This settings limits the maximum number of samples queued for retransmission.

- Full path: //OpenSplice/DDSI2EService/Internal/MaxQueuedRexmitMessages
- Format: integer
- Default value: 200
- Occurrences min-max: 0-1

12.7.6.25 MaxSampleSize

Internal This setting controls the maximum (CDR) serialised size of samples that DDSI2E will forward in either direction. Samples larger than this are discarded with a warning.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/MaxSampleSize
- Format: string
- Default value: 2147483647 B
- Occurrences min-max: 0-1

12.7.6.26 MeasureHbToAckLatency

Internal This element enables heartbeat-to-ack latency among DDSI2E services by prepending timestamps to Heartbeat and AckNack messages and calculating round trip times. This is non-standard behaviour. The measured latencies are quite noisy and are currently not used anywhere.

- Full path: //OpenSplice/DDSI2EService/Internal/MeasureHbToAckLatency

- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.27 MinimumSocketReceiveBufferSize

Internal This setting controls the minimum size of socket receive buffers. The operating system provides some size receive buffer upon creation of the socket, this option can be used to increase the size of the buffer beyond that initially provided by the operating system. If the buffer size cannot be increased to the specified size, an error is reported.

The default setting is the word "default", which means DDSI2E will attempt to increase the buffer size to 1MB, but will silently accept a smaller buffer should that attempt fail.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/MinimumSocketReceiveBufferSize
- Format: string
- Default value: default
- Occurrences min-max: 0-1

12.7.6.28 MinimumSocketSendBufferSize

Internal This setting controls the minimum size of socket send buffers. This setting can only increase the size of the send buffer, if the operating system by default creates a larger buffer, it is left unchanged.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/MinimumSocketSendBufferSize
- Format: string
- Default value: 64 KiB
- Occurrences min-max: 0-1

12.7.6.29 MirrorRemoteEntities

Internal This element controls whether DDSI2 mirrors all entities in the domain in DDSI or only local ones. Default is to discover remote ones iff General/LocalDiscoveryPartition is not the built-in partition.

- Full path: //OpenSplice/DDSI2EService/Internal/MirrorRemoteEntities
- Format: enumeration
- Default value: default
- Valid values: false, true, default
- Occurrences min-max: 0-1

12.7.6.30 MonitorPort

Internal This element allows configuring a service that dumps a text description of part the internal state to TCP clients. By default (-1), this is disabled; specifying 0 means a kernel-allocated port is used; a positive number is used as the TCP port number.

- Full path: //OpenSplice/DDSI2EService/Internal/MonitorPort

- Format: integer
- Default value: -1
- Occurrences min-max: 0-1

12.7.6.31 NackDelay

Internal This setting controls the delay between receipt of a HEARTBEAT indicating missing samples and a NACK (ignored when the HEARTBEAT requires an answer). However, no NACK is sent if a NACK had been scheduled already for a response earlier than the delay requests: then that NACK will incorporate the latest information.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/NackDelay
- Format: string
- Default value: 10 ms
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.6.32 PreEmptiveAckDelay

Internal This setting controls the delay between the discovering a remote writer and sending a pre-emptive AckNack to discover the range of data available.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/PreEmptiveAckDelay
- Format: string
- Default value: 10 ms
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.6.33 PrimaryReorderMaxSamples

Internal This element sets the maximum size in samples of a primary re-order administration. Each proxy writer has one primary re-order administration to buffer the packet flow in case some packets arrive out of order. Old samples are forwarded to secondary re-order administrations associated with readers in need of historical data.

- Full path: //OpenSplice/DDSI2EService/Internal/PrimaryReorderMaxSamples
- Format: integer
- Default value: 64
- Occurrences min-max: 0-1

12.7.6.34 PrioritizeRetransmit

Internal This element controls whether retransmits are prioritized over new data, speeding up recovery.

- Full path: //OpenSplice/DDSI2EService/Internal/PrioritizeRetransmit
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.6.35 RediscoveryBlacklistDuration

Internal This element controls for how long a remote participant that was previously deleted will remain on a blacklist to prevent rediscovery, giving the software on a node time to perform any cleanup actions it needs to do. To some extent this delay is required internally by DDSI2E, but in the default configuration with the 'enforce' attribute set to false, DDSI2E will reallow rediscovery as soon as it has cleared its internal administration. Setting it to too small a value may result in the entry being pruned from the blacklist before DDSI2E is ready, it is therefore recommended to set it to at least several seconds.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/RediscoveryBlacklistDuration
- Format: string
- Default value: 10s
- Occurrences min-max: 0-1
- Optional attributes: enforce

enforce

Internal This attribute controls whether the configured time during which recently deleted participants will not be rediscovered (i.e., "black listed") is enforced and following complete removal of the participant in DDSI2E, or whether it can be rediscovered earlier provided all traces of that participant have been removed already.

- Full path: //OpenSplice/DDSI2EService/Internal/RediscoveryBlacklistDuration[@enforce]
- Format: boolean
- Default value: false
- Required: false

12.7.6.36 ResponsivenessTimeout

Internal This element controls for how long an unresponsive reader can block the transmit thread by failing to acknowledge data when a writer's DDSI2E write cache is full. If after this time the writer's cache has not shrunk to below the low-water mark, the reader is considered unresponsive and degraded to unreliable. It will be restored to reliable service once it resumes acknowledging samples.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/ResponsivenessTimeout
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

12.7.6.37 RetransmitMerging

Internal This elements controls the addressing and timing of retransmits. Possible values are:

- *never*: retransmit only to the NACK-ing reader;
- *adaptive*: attempt to combine retransmits needed 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*. See also Internal/RetransmitMergingPeriod.

- Full path: //OpenSplice/DDSI2EService/Internal/RetransmitMerging
- Format: enumeration
- Default value: adaptive
- Valid values: never, adaptive, always
- Occurrences min-max: 0-1

12.7.6.38 RetransmitMergingPeriod

Internal This setting determines the size of the time window in which a NACK of some sample is ignored because a retransmit of that sample has been multicasted too recently. This setting has no effect on unicasted retransmits.

See also Internal/RetransmitMerging.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/RetransmitMergingPeriod
- Format: string
- Default value: 5 ms
- Valid values: 0 / 1s
- Occurrences min-max: 0-1

12.7.6.39 RetryOnRejectBestEffort

Internal Whether or not to locally retry pushing a received best-effort sample into the reader caches when resource limits are reached.

- Full path: //OpenSplice/DDSI2EService/Internal/RetryOnRejectBestEffort
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.40 RetryOnRejectDuration

Internal How long to keep locally retrying pushing a received sample into the reader caches when resource limits are reached. Default is dependent on Internal/LateAckMode: if the latter is false, it is 80% of Internal/ResponsivenessTimeout, otherwise it is 0.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/RetryOnRejectDuration
- Format: string
- Default value: default
- Occurrences min-max: 0-1

12.7.6.41 SPDPResponseMaxDelay

Internal Maximum pseudo-random delay in milliseconds between discovering a remote participant and responding to it.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/SPDPResponseMaxDelay
- Format: string
- Default value: 0 ms
- Valid values: 0 / 1s
- Occurrences min-max: 0-1

12.7.6.42 ScheduleTimeRounding

Internal This setting allows 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 and causes no rounding at all, i.e. are scheduled exactly, whereas a value of 10ms would mean that events are rounded up to the nearest 10 milliseconds.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/ScheduleTimeRounding
- Format: string
- Default value: 0 ms
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.6.43 SecondaryReorderMaxSamples

Internal This element sets the maximum size in samples of a secondary re-order administration. The secondary re-order administration is per reader in need of historical data.

- Full path: //OpenSplice/DDSI2EService/Internal/SecondaryReorderMaxSamples
- Format: integer
- Default value: 16
- Occurrences min-max: 0-1

12.7.6.44 SquashParticipants

Internal This element controls whether DDSI2E advertises all the domain participants it serves in DDSI (when set to *false*), or rather only one domain participant (the one corresponding to the DDSI2E process; when set to *true*). In the latter case DDSI2E becomes the virtual owner of all readers and writers of all domain participants, dramatically reducing discovery traffic (a similar effect can be obtained by setting Internal/BuiltinEndpointSet to "minimal" but with less loss of information).

- Full path: //OpenSplice/DDSI2EService/Internal/SquashParticipants
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.45 SuppressSPDPMulticast

Internal The element controls whether the mandatory multicasting of the participant discovery packets occurs. Completely disabling multicasting requires this element be set to *true*, and generally requires explicitly listing peers to ping for unicast discovery.

See also General/AllowMulticast.

- Full path: //OpenSplice/DDSI2EService/Internal/SuppressSPDPMulticast
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.6.46 SynchronousDeliveryLatencyBound

Internal This element controls whether samples sent by a writer with QoS settings transport_priority >= SynchronousDeliveryPriorityThreshold and a latency_budget at most this element's value will be delivered synchronously from the "recv" thread, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of aggregate bandwidth.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/SynchronousDeliveryLatencyBound
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

12.7.6.47 SynchronousDeliveryPriorityThreshold

Internal This element controls whether samples sent by a writer with QoS settings latency_budget <= SynchronousDeliveryLatencyBound and transport_priority greater than or equal to this element's value will be delivered synchronously from the "recv" thread, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of aggregate bandwidth.

- Full path: //OpenSplice/DDSI2EService/Internal/SynchronousDeliveryPriorityThreshold
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.6.48 Test

Internal Testing options.

- Full path: //OpenSplice/DDSI2EService/Internal/Test
- Occurrences min-max: 0-1
- Child elements: XmitLossiness

XmitLossiness

Internal This element controls the fraction of outgoing packets to drop, specified as samples per thousand.

- Full path: //OpenSplice/DDSI2EService/Internal/Test/XmitLossiness
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.6.49 UnicastResponseToSPDPMessages

Internal This element controls whether the response to a newly discovered participant is sent as a unicasted SPDP packet, instead of rescheduling the periodic multicasted one. There is no known benefit to setting this to *false*.

- Full path: //OpenSplice/DDSI2EService/Internal/UnicastResponseToSPDPMessages
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.6.50 UseMulticastIfMreqn

Internal Do not use.

- Full path: //OpenSplice/DDSI2EService/Internal/UseMulticastIfMreqn
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.7.6.51 Watermarks

Internal Watermarks for flow-control.

- Full path: //OpenSplice/DDSI2EService/Internal/Watermarks
- Occurrences min-max: 0-1
- Child elements: WhcAdaptive, WhcHigh, WhcHighInit, WhcLow

WhcAdaptive

Internal This element controls whether DDSI2E will adapt the high-water mark to current traffic conditions, based on retransmit requests and transmit pressure.

- Full path: //OpenSplice/DDSI2EService/Internal/Watermarks/WhcAdaptive
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

WhcHigh

Internal This element sets the maximum allowed high-water mark for the DDSI2E WHCs, expressed in bytes. A writer is suspended when the WHC reaches this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/Watermarks/WhcHigh
- Format: string
- Default value: 100 kB
- Occurrences min-max: 0-1

WhcHighInit

Internal This element sets the initial level of the high-water mark for the DDSI2E WHCs, expressed in bytes.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/Watermarks/WhcHighInit
- Format: string
- Default value: 30 kB
- Occurrences min-max: 0-1

WhcLow

Internal This element sets the low-water mark for the DDSI2E WHCs, expressed in bytes. A suspended writer resumes transmitting when its DDSI2E WHC shrinks to this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Internal/Watermarks/WhcLow
- Format: string
- Default value: 1 kB
- Occurrences min-max: 0-1

12.7.6.52 WriterLingerDuration

Internal This setting controls the maximum duration for which actual deletion of a reliable writer with unacknowledged data in its history will be postponed to provide proper reliable transmission. The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/Internal/WriterLingerDuration
- Format: string
- Default value: 1 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.7 Partitioning

The Partitioning element specifies DDSI2E network partitions and how DCPS partition/topic combinations are mapped onto the network partitions.

- Full path: //OpenSplice/DDSI2EService/Partitioning
- Occurrences min-max: 0-1

12.7.7.1 IgnoredPartitions

The IgnoredPartitions element specifies DCPS partition/topic combinations that are not distributed over the network.

- Full path: //OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions
- Occurrences min-max: 0-*

IgnoredPartition

This element can be used to prevent certain combinations of DCPS partition and topic from being transmitted over the network. DDSI2E will complete ignore readers and writers for which all DCPS partitions as well as their topic is ignored, not even creating DDSI readers and writers to mirror the DCPS ones.

- Full path: //OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions/IgnoredPartition
- Occurrences min-max: 0-*
- Required attributes: DCPSPartitionTopic

12.7.7.1.1.1 DCPSPartitionTopic This attribute specifies a partition and a topic expression, separated by a single '.', that are used to determine if a given partition and topic will be ignored or not. The expressions may use the usual wildcards '*' and '?'. DDSI2E will consider an wildcard DCPS partition to match an expression iff there exists a string that satisfies both expressions.

- Full path: //OpenSplice/DDSI2EService/Partitioning/IgnoredPartitions/IgnoredPartition[@DCPSPartitionTopic]
- Format: string
- Default value: n/a
- Required: true

12.7.7.2 NetworkPartitions

The NetworkPartitions element specifies the DDSI2E network partitions.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions
- Occurrences min-max: 0-*

NetworkPartition

This element defines a DDSI2E network partition.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions/NetworkPartition
- Occurrences min-max: 0-*
- Required attributes: Address, Name
- Optional attributes: Connected, SecurityProfile

12.7.7.2.1.1 Address This attribute specifies the multicast addresses associated with the network partition as a comma-separated list. Readers matching this network partition (cf. Partitioning/PartitionMappings) will listen for multicasts on all of these addresses and advertise them in the discovery protocol. The writers will select the most suitable address from the addresses advertised by the readers.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions/NetworkPartition[@Address]
- Format: string
- Default value: n/a
- Required: true

12.7.7.2.1.2 Connected This attribute is a placeholder.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions/NetworkPartition[@Connected]
- Format: boolean
- Default value: true
- Required: false

12.7.7.2.1.3 Name This attribute specifies the name of this DDSI2E network partition. Two network partitions cannot have the same name.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions/NetworkPartition[@Name]
- Format: string
- Default value: n/a
- Required: true

12.7.7.2.1.4 SecurityProfile This attribute selects the DDSI2E security profile for encrypting the traffic mapped to this DDSI2E network partition. The default "null" means the network partition is unsecured; any other name refers to a security profile defined using the Security/SecurityProfile elements.

- Full path: //OpenSplice/DDSI2EService/Partitioning/NetworkPartitions/NetworkPartition[@SecurityProfile]
- Format: string
- Default value: null
- Required: false

12.7.7.3 PartitionMappings

The PartitionMappings element specifies the mapping from DCPS partition/topic combinations to DDSI2E network partitions.

- Full path: //OpenSplice/DDSI2EService/Partitioning/PartitionMappings
- Occurrences min-max: 0-*

PartitionMapping

This element defines a mapping from a DCPS partition/topic combination to a DDSI2E network partition. This allows partitioning data flows by using special multicast addresses for part of the data and possibly also encrypting the data flow.

- Full path: //OpenSplice/DDSI2EService/Partitioning/PartitionMappings/PartitionMapping
- Occurrences min-max: 0-*

- Required attributes: DCPSPartitionTopic, NetworkPartition

12.7.7.3.1.1 DCPSPartitionTopic This attribute specifies a partition and a topic expression, separated by a single '.', that are used to determine if a given partition and topic maps to the DDSI2E network partition named by the NetworkPartition attribute in this PartitionMapping element. The expressions may use the usual wildcards '*' and '?'. DDSI2E will consider a wildcard DCPS partition to match an expression if there exists a string that satisfies both expressions.

- Full path: //OpenSplice/DDSI2EService/Partitioning/PartitionMappings/PartitionMapping[@DCPSPartitionTopic]
- Format: string
- Default value: n/a
- Required: true

12.7.7.3.1.2 NetworkPartition This attribute specifies which DDSI2E network partition is to be used for DCPS partition/topic combinations matching the DCPSPartitionTopic attribute within this PartitionMapping element.

- Full path: //OpenSplice/DDSI2EService/Partitioning/PartitionMappings/PartitionMapping[@NetworkPartition]
- Format: string
- Default value: n/a
- Required: true

12.7.8 SSL

The SSL element allows specifying various parameters related to using SSL/TLS for DDSI over TCP.

- Full path: //OpenSplice/DDSI2EService/SSL
- Occurrences min-max: 0-1
- Child elements: CertificateVerification, Ciphers, Enable, EntropyFile, KeyPassphrase, KeystoreFile, SelfSignedCertificates, VerifyClient

12.7.8.1 CertificateVerification

If disabled this allows SSL connections to occur even if an X509 certificate fails verification.

- Full path: //OpenSplice/DDSI2EService/SSL/CertificateVerification
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.8.2 Ciphers

The set of ciphers used by SSL/TLS

- Full path: //OpenSplice/DDSI2EService/SSL/Ciphers
- Format: string
- Default value: ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH
- Occurrences min-max: 0-1

12.7.8.3 Enable

This enables SSL/TLS for TCP.

- Full path: //OpenSplice/DDSI2EService/SSL/Enable
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.8.4 EntropyFile

The SSL/TLS random entropy file name.

- Full path: //OpenSplice/DDSI2EService/SSL/EntropyFile
- Format: string
- Occurrences min-max: 0-1

12.7.8.5 KeyPassphrase

The SSL/TLS key pass phrase for encrypted keys.

- Full path: //OpenSplice/DDSI2EService/SSL/KeyPassphrase
- Format: string
- Default value: secret
- Occurrences min-max: 0-1

12.7.8.6 KeystoreFile

The SSL/TLS key and certificate store file name. The keystore must be in PEM format.

- Full path: //OpenSplice/DDSI2EService/SSL/KeystoreFile
- Format: string
- Default value: keystore
- Occurrences min-max: 0-1

12.7.8.7 SelfSignedCertificates

This enables the use of self signed X509 certificates.

- Full path: //OpenSplice/DDSI2EService/SSL/SelfSignedCertificates
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.8.8 VerifyClient

This enables an SSL server checking the X509 certificate of a connecting client.

- Full path: //OpenSplice/DDSI2EService/SSL/VerifyClient
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.9 Security

The Security element specifies DDSI2E security profiles that can be used to encrypt traffic mapped to DDSI2E network partitions.

- Full path: //OpenSplice/DDSI2EService/Security
- Occurrences min-max: 0-1

12.7.9.1 SecurityProfile

This element defines a DDSI2E security profile.

- Full path: //OpenSplice/DDSI2EService/Security/SecurityProfile
- Occurrences min-max: 0-*
- Required attributes: Name
- Optional attributes: Cipher, CipherKey

Cipher

This attribute specifies the cipher to be used for encrypting traffic over network partitions secured by this security profile. The possible ciphers are:

- *aes128*: AES with a 128-bit key;
- *aes192*: AES with a 192-bit key;
- *aes256*: AES with a 256-bit key;
- *blowfish*: the Blowfish cipher with a 128 bit key;
- *null*: no encryption;

SHA1 is used on conjunction with all ciphers except “null” to ensure data integrity.

- Full path: //OpenSplice/DDSI2EService/Security/SecurityProfile[@Cipher]
- Format: enumeration
- Default value: null
- Valid values: null, blowfish, aes128, aes192, aes256
- Required: false

CipherKey

The CipherKey attribute is used to define the secret key required by the cipher selected using the Cipher attribute. The value can be a URI referencing an external file containing the secret key, or the secret key can be defined in-place as a string value.

The key must be specified as a hexadecimal string with each character representing 4 bits of the key. E.g., 1ABC represents the 16-bit key 0001 1010 1011 1100. The key should not follow a well-known pattern and must exactly match the key length of the selected cipher.

A malformed key will cause the security profile to be marked as invalid, and disable all network partitions secured by the (invalid) security profile to prevent information leaks.

As all DDS 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 its URI. The file should be protected against read and write access from other users on the host.

- Full path: //OpenSplice/DDSI2EService/Security/SecurityProfile[@CipherKey]
- Format: string
- Default value: “”
- Required: false

Name

This attribute specifies the name of this DDSI2E security profile. Two security profiles cannot have the same name.

- Full path: //OpenSplice/DDSI2EService/Security/SecurityProfile[@Name]
- Format: string
- Default value: n/a
- Required: true

12.7.10 Sizing

The Sizing element specifies a variety of configuration settings dealing with expected system sizes, buffer sizes, &c.

- Full path: //OpenSplice/DDSI2EService/Sizing
- Occurrences min-max: 0-1
- Child elements: EndpointsInSystem, EndpointsInSystem, LocalEndpoints, NetworkQueueSize, NetworkQueueSize, ReceiveBufferChunkSize, ReceiveBufferChunkSize

12.7.10.1 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/DDSI2EService/Sizing/EndpointsInSystem
- Format: integer
- Default value: 20000
- Occurrences min-max: 0-1

12.7.10.2 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/DDSI2EService/Sizing/EndpointsInSystem
- Format: integer
- Default value: 20000
- Occurrences min-max: 0-1

12.7.10.3 LocalEndpoints

This element specifies the expected maximum number of endpoints local to one DDSI2E service. 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/DDSI2EService/Sizing/LocalEndpoints
- Format: integer
- Default value: 1000
- Occurrences min-max: 0-1

12.7.10.4 NetworkQueueSize

This element specifies the maximum number of samples in the network queue. Write/dispose operations add samples to this queue, the DDSI2E service drains it. Larger values allow large bursts of writes to occur without forcing synchronization between the application and the DDSI2E 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/DDSI2EService/Sizing/NetworkQueueSize
- Format: integer
- Default value: 1000
- Occurrences min-max: 0-1

12.7.10.5 NetworkQueueSize

This element specifies the maximum number of samples in the network queue. Write/dispose operations add samples to this queue, the DDSI2E service drains it. Larger values allow large bursts of writes to occur without forcing synchronization between the application and the DDSI2E 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/DDSI2EService/Sizing/NetworkQueueSize
- Format: integer
- Default value: 1000
- Occurrences min-max: 0-1

12.7.10.6 ReceiveBufferChunkSize

This element specifies the 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). Each allocation is shrunk immediately after processing a message, or freed straightaway.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Sizing/ReceiveBufferChunkSize
- Format: string
- Default value: 128 KiB
- Occurrences min-max: 0-1

12.7.10.7 ReceiveBufferChunkSize

This element specifies the 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). Each allocation is shrunk immediately after processing a message, or freed straightaway.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Sizing/ReceiveBufferChunkSize
- Format: string
- Default value: 128 KiB
- Occurrences min-max: 0-1

12.7.10.8 Watermarks

#if LITE { LEAF (“ReceiveBufferSize”), 1, “128 KiB”, ABSOFF (rbuf_size), 0, uf_memsize, 0, pf_memsize, “This element sets the size of a single receive buffer. Many receive buffers may be needed. Their size must be greater than ReceiveBufferChunkSize by a modest amount.

- Full path: //OpenSplice/DDSI2EService/Sizing/Watermarks
- Occurrences min-max: 0-1
- Child elements: WhcAdaptive, WhcHigh, WhcHighInit, WhcLow

WhcAdaptive

This element controls whether DDSI2E will adapt the high-water mark to current traffic conditions, based on retransmit requests and transmit pressure.

- Full path: //OpenSplice/DDSI2EService/Sizing/Watermarks/WhcAdaptive
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

WhcHigh

This element sets the maximum allowed high-water mark for the DDSI2E WHCs, expressed in bytes. A writer is suspended when the WHC reaches this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Sizing/Watermarks/WhcHigh
- Format: string
- Default value: 100 kB
- Occurrences min-max: 0-1

WhcHighInit

This element sets the initial level of the high-water mark for the DDSI2E WHCs, expressed in bytes.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Sizing/Watermarks/WhcHighInit
- Format: string
- Default value: 30 kB
- Occurrences min-max: 0-1

WhcLow

This element sets the low-water mark for the DDSI2E WHCs, expressed in bytes. A suspended writer resumes transmitting when its DDSI2E WHC shrinks to this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Sizing/Watermarks/WhcLow
- Format: string
- Default value: 1 kB
- Occurrences min-max: 0-1

12.7.11 TCP

The TCP element allows specifying various parameters related to running DDSI over TCP.

- Full path: //OpenSplice/DDSI2EService/TCP
- Occurrences min-max: 0-1
- Child elements: Enable, NoDelay, Port, ReadTimeout, WriteTimeout

12.7.11.1 Enable

This element enables the optional TCP transport.

- Full path: //OpenSplice/DDSI2EService/TCP/Enable
- Format: boolean

- Default value: false
- Occurrences min-max: 0-1

12.7.11.2 NoDelay

This element enables the TCP_NODELAY socket option, preventing multiple DDSI messages being sent in the same TCP request. Setting this option typically optimises latency over throughput.

- Full path: //OpenSplice/DDSI2EService/TCP/NoDelay
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.7.11.3 Port

This element specifies the TCP port number on which DDSI2E accepts connections. If the port is set it is used in entity locators, published with DDSI discovery. Dynamically allocated if zero. Disabled if -1 or not configured. If disabled other DDSI services will not be able to establish connections with the service, the service can only communicate by establishing connections to other services.

- Full path: //OpenSplice/DDSI2EService/TCP/Port
- Format: integer
- Default value: -1
- Valid values: -1 / 65535
- Occurrences min-max: 0-1

12.7.11.4 ReadTimeout

This element specifies the timeout for blocking TCP read operations. If this timeout expires then the connection is closed.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/TCP/ReadTimeout
- Format: string
- Default value: 2 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.11.5 WriteTimeout

This element specifies the timeout for blocking TCP write operations. If this timeout expires then the connection is closed.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2EService/TCP/WriteTimeout
- Format: string
- Default value: 2 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.7.12 ThreadPool

The ThreadPool element allows specifying various parameters related to using a thread pool to send DDSI messages to multiple unicast addresses (TCP or UDP).

- Full path: //OpenSplice/DDSI2EService/ThreadPool
- Occurrences min-max: 0-1
- Child elements: Enable, ThreadMax, Threads

12.7.12.1 Enable

This element enables the optional thread pool.

- Full path: //OpenSplice/DDSI2EService/ThreadPool/Enable
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.12.2 ThreadMax

This elements configures the maximum number of threads in the thread pool.

- Full path: //OpenSplice/DDSI2EService/ThreadPool/ThreadMax
- Format: integer
- Default value: 8
- Occurrences min-max: 0-1

12.7.12.3 Threads

This elements configures the initial number of threads in the thread pool.

- Full path: //OpenSplice/DDSI2EService/ThreadPool/Threads
- Format: integer
- Default value: 4
- Occurrences min-max: 0-1

12.7.13 Threads

This element is used to set thread properties.

- Full path: //OpenSplice/DDSI2EService/Threads
- Occurrences min-max: 0-1

12.7.13.1 Thread

This element is used to set thread properties.

- Full path: //OpenSplice/DDSI2EService/Threads/Thread
- Occurrences min-max: 0-1000
- Child elements: StackSize

- Required attributes: Name

Name

The Name of the thread for which properties are being set. The following threads exist:

- *gc*: garbage collector thread involved in deleting entities;
- *recv*: receive thread, taking data from the network and running the protocol state machine;
- *dq.builtins*: delivery thread for DDSI-builtin data, primarily for discovery;
- *lease*: DDSI liveliness monitoring;
- *tev*: general timed-event handling, retransmits and discovery;
- *xmit.CHAN*: transmit thread for channel CHAN;
- *dq.CHAN*: delivery thread for channel CHAN;
- *tev.CHAN*: timed-even thread for channel CHAN.
- Full path: //OpenSplice/DDSI2EService/Threads/Thread[@Name]
- Format: string
- Default value: n/a
- Required: true

Scheduling

This element configures the scheduling properties of the thread.

- Full path: //OpenSplice/DDSI2EService/Threads/Thread/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

12.7.13.1.2.1 Class This element specifies the thread scheduling class (*realtime*, *timeshare* or *default*). The user may need special privileges from the underlying operating system to be able to assign some of the privileged scheduling classes.

- Full path: //OpenSplice/DDSI2EService/Threads/Thread/Scheduling/Class
- Format: enumeration
- Default value: default
- Valid values: realtime, timeshare, default
- Occurrences min-max: 0-1

12.7.13.1.2.2 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/DDSI2EService/Threads/Thread/Scheduling/Priority
- Format: string
- Default value: default
- Occurrences min-max: 0-1

StackSize

This element configures the stack size for this thread. The default value *default* leaves the stack size at the operating system default.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2EService/Threads/Thread/StackSize
- Format: string
- Default value: default
- Occurrences min-max: 0-1

12.7.14 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/DDSI2EService/Tracing
- Occurrences min-max: 0-1
- Child elements: AppendToFile, EnableCategory, OutputFile, PacketCaptureFile, Timestamps, Verbosity

12.7.14.1 AppendToFile

This option specifies whether the output is to be appended to an existing log file. The default is to create a new log file each time, which is generally the best option if a detailed log is generated.

- Full path: //OpenSplice/DDSI2EService/Tracing/AppendToFile
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.7.14.2 EnableCategory

This element enables individual logging categories. These are enabled in addition to those enabled by Tracing/Verbosity. Recognised categories are:

- *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
- *whc*: tracing of writer history cache changes

- *tcp*: tracing of TCP-specific activity
- *topic*: tracing of topic definitions
- >i>plist*: tracing of discovery parameter list interpretation

In addition, there is the keyword *trace* that enables all but *radmin*, *topic*, *plist* and *whc* . 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. Currently, the most useful is *trace*.

- Full path: //OpenSplice/DDSI2EService/Tracing/EnableCategory
- Format: string
- Occurrences min-max: 0-1

12.7.14.3 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/DDSI2EService/Tracing/OutputFile
- Format: string
- Default value: ddsi2.log
- Occurrences min-max: 0-1

12.7.14.4 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 ficitious, 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. Currently IPv4 only.

- Full path: //OpenSplice/DDSI2EService/Tracing/PacketCaptureFile
- Format: string
- Occurrences min-max: 0-1

12.7.14.5 Timestamps

This option has no effect.

- Full path: //OpenSplice/DDSI2EService/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

absolute

This option has no effect

- Full path: //OpenSplice/DDSI2EService/Tracing/Timestamps[@absolute]
- Format: boolean

- Default value: true
- Required: false

12.7.14.6 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. Recognised verbosity levels and the categories they map to are:

- *none*: no DDSI2E log
- *severe*: error and fatal
- *warning*: *severe* + warning
- *info*: *warning* + info
- *config*: *info* + config
- *fine*: *config* + discovery
- *finer*: *fine* + traffic and timing
- *finest*: *finer* + trace

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. Currently, the most useful verbosity levels are *config*, *fine* and *finest*.

- Full path: //OpenSplice/DDSI2EService/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, finer, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.7.15 Watchdog

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DDSI2EService/Watchdog
- Occurrences min-max: 0-1

12.7.15.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DDSI2EService/Watchdog/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

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/DDSI2EService/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: default
- Valid values: realtime, timeshare, default
- Occurrences min-max: 0-1

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/DDSI2EService/Watchdog/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1
- Optional attributes: priority_kind

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

- Full path: //OpenSplice/DDSI2EService/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: relative
- Valid values: relative, absolute
- Required: false

12.8 DDSI2Service

The root element of a DDSI2 networking service configuration.

- Full path: //OpenSplice/DDSI2Service
- Occurrences min-max: 0-*
- Required attributes: name

12.8.1 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/Domain/Service[@name] in the Domain Service configuration.

- Full path: //OpenSplice/DDSI2Service[@name]

- Format: string
- Default value: ddsi2
- Required: true

12.8.2 Compatibility

The Compatibility elements allows specifying various settings related to compatability with standards and with other DDSI implementations.

- Full path: //OpenSplice/DDSI2Service/Compatibility
- Occurrences min-max: 0-1
- Child elements: AckNackNumbitsEmptySet, ArrivalOfDataAssertsPpAndEpLiveliness, AssumeRtiHasPmdEndpoints, ExplicitlyPublishQosSetToDefault, ManySocketsMode, RespondToRtiInitZeroAckWithInvalidHeartbeat, StandardsConformance

12.8.2.1 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 reject by the strict and pedantic conformance modes, but several other implementation require this setting for smooth interoperation.

If set to 1, all acknowledgements sent by DDSI2 adhere 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 an AckNackNumbitsEmptySet=0 to prevent the transmitting of invalid messages.

- Full path: //OpenSplice/DDSI2Service/Compatibility/AckNackNumbitsEmptySet
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.8.2.2 ArrivalOfDataAssertsPpAndEpLiveliness

When set to true, arrival of a message from a peer asserts liveliness of that peer. When set to false, only SPDP and explicit lease renewals have this effect.

- Full path: //OpenSplice/DDSI2Service/Compatibility/ArrivalOfDataAssertsPpAndEpLiveliness
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.2.3 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
- Occurrences min-max: 0-1

12.8.2.4 ExplicitlyPublishQosSetToDefault

This element 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 significant 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
- Occurrences min-max: 0-1

12.8.2.5 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 will ensure this to be the case. This is not the defeault.

Disabling it slightly improves performance and reduces network traffic somewhat. It also causes the set of port numbers needed by DDSI2 to become predictable, which may be useful for firewall and NAT configuration.

- Full path: //OpenSplice/DDSI2Service/Compatibility/ManySocketsMode
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.2.6 RespondToRtiInitZeroAckWithInvalidHeartbeat

This element 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
- Occurrences min-max: 0-1

12.8.2.7 StandardsConformance

This element sets 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 conform 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*: a slightly less strict view of the standard than does pedantic: it follows the established behaviour where the standard is obviously in error.

- *lax*: attempt 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

12.8.3 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: AdvertiseBuiltinTopicWriters, DSGracePeriod, DefaultMulticastAddress, DomainId, GenerateBuiltinTopics, LocalDiscoveryPartition, MaxAutoParticipantIndex, ParticipantIndex, SPDPInterval, SPDPMulticastAddress

12.8.3.1 AdvertiseBuiltinTopicWriters

This element controls whether or not DDSI2 advertises writers for the built-in topics from its discovery for backwards compatibility with older Vortex OpenSplice versions.

- Full path: //OpenSplice/DDSI2Service/Discovery/AdvertiseBuiltinTopicWriters
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.3.2 DSGracePeriod

This setting controls for how long endpoints discovered via a Cloud discovery service will survive after the discovery service disappeared, allowing reconnect without loss of data when the discovery service restarts (or another instance takes over).

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Discovery/DSGracePeriod
- Format: string
- Default value: 30 s
- Occurrences min-max: 0-1

12.8.3.3 DefaultMulticastAddress

This element specifies the default multicast address for all traffic other than participant discovery packets. It defaults to Discovery/SPDPMulticastAddress.

- Full path: //OpenSplice/DDSI2Service/Discovery/DefaultMulticastAddress

- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.8.3.4 DomainId

This element allows overriding of the DDS Domain Id that is used for DDSI2.

- Full path: //OpenSplice/DDSI2Service/Discovery/DomainId
- Format: string
- Default value: default
- Occurrences min-max: 0-1

12.8.3.5 GenerateBuiltinTopics

This element controls whether or not DDSI2 generates built-in topics from its discovery. When disabled, it relies on the durability service.

- Full path: //OpenSplice/DDSI2Service/Discovery/GenerateBuiltinTopics
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.3.6 LocalDiscoveryPartition

This element controls which partition is monitored by DDSI2 for built-in topics describing entities the it mirrors in DDSI.

- Full path: //OpenSplice/DDSI2Service/Discovery/LocalDiscoveryPartition
- Format: string
- Default value: __BUILT-IN PARTITION__
- Occurrences min-max: 0-1

12.8.3.7 MaxAutoParticipantIndex

This element specifies the maximum DDSI participant index selected by this instance of the DDSI2 service if the Discovery/ParticipantIndex is “auto”.

- Full path: //OpenSplice/DDSI2Service/Discovery/MaxAutoParticipantIndex
- Format: integer
- Default value: 9
- Occurrences min-max: 0-1

12.8.3.8 ParticipantIndex

This element specifies the DDSI participant index 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*: which will attempt to automatically determine an available participant index (see also Discovery/MaxAutoParticipantIndex), 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.

The default is *auto*. The participant index is part of the port number calculation and if predictable port numbers are needed and fixing the participant index has no adverse effects, it is recommended that the second be option be used.

- Full path: //OpenSplice/DDSI2Service/Discovery/ParticipantIndex
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.8.3.9 Peers

This element statically configures addresses for discovery.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers
- Occurrences min-max: 0-1

Group

This element statically configures a fault tolerant group of addresses for discovery. Each member of the group is tried in sequence until one succeeds.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers/Group
- Occurrences min-max: 0-*

12.8.3.9.1.1 Peer This element statically configures an addresses for discovery.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers/Group/Peer
- Occurrences min-max: 0-*
- Required attributes: Address

12.8.3.9.1.2 Address This element specifies an IP address to which discovery packets must be sent, in addition to the default multicast address (see also General/AllowMulticast). Both a hostnames and a numerical IP address is accepted; the hostname or IP address may be suffixed with :PORT to explicitly set the port to which it must be sent. Multiple Peers may be specified.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers/Group/Peer[@Address]
- Format: string
- Default value: n/a
- Required: true

Peer

This element statically configures an addresses for discovery.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers/Peer
- Occurrences min-max: 0-*
- Required attributes: Address

12.8.3.9.2.1 Address This element specifies an IP address to which discovery packets must be sent, in addition to the default multicast address (see also General/AllowMulticast). Both a hostnames and a numerical IP address is accepted; the hostname or IP address may be suffixed with :PORT to explicitly set the port to which it must be sent. Multiple Peers may be specified.

- Full path: //OpenSplice/DDSI2Service/Discovery/Peers/Peer[@Address]
- Format: string
- Default value: n/a
- Required: true

12.8.3.10 Ports

The Ports element allows specifying various parameters related to the port numbers used for discovery. These all have default values specified by the DDSI 2.1 (and 2.2) specification and rarely need to be changed.

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports
- Occurrences min-max: 0-1
- Child elements: Base, DomainGain, MulticastDataOffset, MulticastMetaOffset, ParticipantGain, UnicastDataOffset, UnicastMetaOffset

Base

This element specifies the base port number (refer to the DDSI 2.1 or 2.2 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

DomainGain

This element specifies the domain gain, relating domain ids to sets of port numbers (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant DG).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/DomainGain
- Format: integer
- Default value: 250
- Occurrences min-max: 0-1

MulticastDataOffset

This element specifies the port number for multicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d2).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/MulticastDataOffset
- Format: integer
- Default value: 1
- Occurrences min-max: 0-1

MulticastMetaOffset

This element specifies the port number for multicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d0).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/MulticastMetaOffset
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

ParticipantGain

This element specifies the participant gain, relating p0, articipant index to sets of port numbers (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant PG).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/ParticipantGain
- Format: integer
- Default value: 2
- Occurrences min-max: 0-1

UnicastDataOffset

This element specifies the port number for unicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d3).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/UnicastDataOffset
- Format: integer
- Default value: 11
- Occurrences min-max: 0-1

UnicastMetaOffset

This element specifies the port number for unicast meta traffic (refer to the DDSI 2.1 or 2.2 specification, section 9.6.1, constant d1).

- Full path: //OpenSplice/DDSI2Service/Discovery/Ports/UnicastMetaOffset
- Format: integer
- Default value: 10
- Occurrences min-max: 0-1

12.8.3.11 SPDPInterval

This element specifies the interval between spontaneous transmissions of participant discovery packets.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Discovery/SPDPInterval
- Format: string
- Default value: 30 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.3.12 SPDPMulticastAddress

This element specifies the multicast address that is used as the destination for the participant discovery packets. In IPv4 mode the default is the (standardised) 239.255.0.1, in IPv6 mode it becomes ff02::ffff:239.255.0.1, which is a non-standardised link-local multicast address.

- Full path: //OpenSplice/DDSI2Service/Discovery/SPDPMulticastAddress
- Format: string
- Default value: 239.255.0.1
- Occurrences min-max: 0-1

12.8.4 General

The General element specifies overall DDSI2 service settings.

- Full path: //OpenSplice/DDSI2Service/General
- Occurrences min-max: 0-1
- Child elements: AllowMulticast, CoexistWithNativeNetworking, DontRoute, EnableMulticastLoopback, ExternalNetworkAddress, ExternalNetworkMask, FragmentSize, MaxMessageSize, MulticastRecvNetworkInterfaceAddresses, MulticastTimeToLive, NetworkInterfaceAddress, StartupModeCoversTransient, StartupModeDuration, UseIPv6

12.8.4.1 AllowMulticast

This element controls whether DDSI2 uses multicasts for data traffic.

It is a comma-separated list of some of the following keywords: “spdp”, “asm”, “ssm”, or either of “false” or “true”.

- *spdp*: enables the use of ASM (any-source multicast) for participant discovery
- *asm*: enables the use of ASM for all traffic (including SPDP)
- *ssm*: enables the use of SSM (source-specific multicast) for all non-SPDP traffic (if supported)

When set to “false” all multicasting is disabled. The default, “true” enables full use of multicasts. Listening for multicasts can be controlled by General/MulticastRecvNetworkInterfaceAddresses.

- Full path: //OpenSplice/DDSI2Service/General/AllowMulticast
- Format: string
- Default value: true
- Occurrences min-max: 0-1

12.8.4.2 CoexistWithNativeNetworking

This element specifies whether the DDSI2 service operates in conjunction with the Vortex OpenSplice RT Networking service. When "false", the DDSI2 service will take care of all communications, including those between Vortex OpenSplice nodes; when "true", the DDSI2 service only communicates with DDS implementations other than Vortex OpenSplice. In this case the RT Networking service should be configured as well.

- Full path: //OpenSplice/DDSI2Service/General/CoexistWithNativeNetworking
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.4.3 DontRoute

This element allows setting the SO_DONTROUTE option for outgoing packets, to bypass the local routing tables. This is generally useful only when the routing tables cannot be trusted, which is highly unusual.

- Full path: //OpenSplice/DDSI2Service/General/DontRoute
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.4.4 EnableMulticastLoopback

This element specifies whether DDSI2 allows IP multicast packets to be visible to all DDSI participants in the same node, including itself. It must be "true" for intra-node multicast communications, but if a node runs only a single DDSI2 service and does not host any other DDSI-capable programs, it should be set to "false" for improved performance.

- Full path: //OpenSplice/DDSI2Service/General/EnableMulticastLoopback
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.4.5 ExternalNetworkAddress

This element allows explicitly overruling the network address DDSI2 advertises in the discovery protocol, which by default is the address of the preferred network interface (General/NetworkInterfaceAddress), to allow DDSI2 to communicate across a Network Address Translation (NAT) device.

- Full path: //OpenSplice/DDSI2Service/General/ExternalNetworkAddress
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.8.4.6 ExternalNetworkMask

This element specifies the network mask of the external network address. This element is relevant only when an external network address (General/ExternalNetworkAddress) is explicitly configured. In this case locators received via the discovery protocol that are within the same external subnet (as defined by this mask) will be

translated to an internal address by replacing the network portion of the external address with the corresponding portion of the preferred network interface address. This option is IPv4-only.

- Full path: //OpenSplice/DDSI2Service/General/ExternalNetworkMask
- Format: string
- Default value: 0.0.0.0
- Occurrences min-max: 0-1

12.8.4.7 FragmentSize

This element specifies the size of DDSI sample fragments generated by DDSI2. Samples larger than FragmentSize are fragmented into fragments of FragmentSize bytes each, except the last one, which may be smaller. The DDSI spec 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.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/General/FragmentSize
- Format: string
- Default value: 1280 B
- Occurrences min-max: 0-1

12.8.4.8 MaxMessageSize

This element specifies the maximum size of the UDP payload that DDSI2 will generate. DDSI2 will try to maintain this limit within the bounds of the DDSI specification, which means that in some cases (especially for very low values of MaxMessageSize) larger payloads may sporadically be observed (currently up to 1192 B).

On some networks it may be necessary to set this item to keep the packetsize below the MTU to prevent IP fragmentation. In those cases, it is generally advisable to also consider reducing Internal/FragmentSize.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/General/MaxMessageSize
- Format: string
- Default value: 4096 B
- Occurrences min-max: 0-1

12.8.4.9 MulticastRecvNetworkInterfaceAddresses

This element specifies on which network interfaces DDSI2 listens to multicasts. The following options are available:

- *all*: listen for multicasts on all multicast-capable interfaces; or
- *any*: listen for multicasts on the operating system default interface; or
- *preferred*: listen for multicasts on the preferred interface (General/NetworkInterfaceAddress); or
- *none*: does not listen for multicasts on any interface; or
- a comma-separated list of network addresses: configures DDSI2 to listen for multicasts on all of the listed addresses.

If DDSI2 is in IPv6 mode and the address of the preferred network interface is a link-local address, "all" is treated as a synonym for "preferred" and a comma-separated list is treated as "preferred" if it contains the preferred interface and as "none" if not.

- Full path: //OpenSplice/DDSI2Service/General/MulticastRecvNetworkInterfaceAddresses
- Format: string
- Default value: preferred
- Occurrences min-max: 0-1

12.8.4.10 MulticastTimeToLive

This element specifies the time-to-live setting for outgoing multicast packets.

- Full path: //OpenSplice/DDSI2Service/General/MulticastTimeToLive
- Format: integer
- Default value: 32
- Valid values: 0 / 255
- Occurrences min-max: 0-1

12.8.4.11 NetworkInterfaceAddress

This element specifies the preferred network interface for use by DDSI2. The preferred network interface determines the IP address that DDSI2 advertises in the discovery protocol (but see also General/ExternalNetworkAddress), and is also the only interface over which multicasts are transmitted. The interface can be identified by its IP address, network interface name or network portion of the address. If the value "auto" is entered here, DDSI2 will select what it considers the most suitable interface.

- Full path: //OpenSplice/DDSI2Service/General/NetworkInterfaceAddress
- Format: string
- Default value: auto
- Occurrences min-max: 0-1

12.8.4.12 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. Configurations without defined merge policies best leave this enabled.

- Full path: //OpenSplice/DDSI2Service/General/StartupModeCoversTransient
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.4.13 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, allowing existing readers on remote nodes to obtain 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, DDSI2 keeps track of the existence of remote readers whether or not matching writers exist locally, avoiding this discovery delay and ensuring this is merely a node startup issue.

Setting General/StartupModeDuration to 0s will disable it.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/General/StartupModeDuration
- Format: string
- Default value: 2 s
- Valid values: 0 / 60000
- Occurrences min-max: 0-1

12.8.4.14 UseIPv6

This element can be used to DDSI2 use IPv6 instead of IPv4. This is currently an either/or switch.

- Full path: //OpenSplice/DDSI2Service/General/UseIPv6
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5 Internal

The Internal elements deal with a variety of settings that evolving and that are not necessarily fully supported. For the vast majority of the Internal settings, the functionality per-se is supported, but the right to change the way the options control the functionality is reserved. This includes renaming or moving options.

- Full path: //OpenSplice/DDSI2Service/Internal
- Occurrences min-max: 0-1
- Child elements: AccelerateRexmitBlockSize, AggressiveKeepLastWhc, AssumeMulticastCapable, AutoReschedNackDelay, BuiltinEndpointSet, ConservativeBuiltinReaderStartup, DDSI2DirectMaxThreads, DefragReliableMaxSamples, DefragUnreliableMaxSamples, DeliveryQueueMaxSamples, ForwardAllMessages, ForwardRemoteData, GenerateKeyhash, HeartbeatInterval, LateAckMode, LeaseDuration, LegacyFragmentation, LogStackTraces, MaxParticipants, MaxQueuedRexmitBytes, MaxQueuedRexmitMessages, MaxSampleSize, MeasureHbToAckLatency, MinimumSocketReceiveBufferSize, MinimumSocketSendBufferSize, MirrorRemoteEntities, MonitorPort, NackDelay, PreEmptiveAckDelay, PrimaryReorderMaxSamples, PrioritizeRetransmit, RediscoveryBlacklistDuration, ResponsivenessTimeout, RetransmitMerging, RetransmitMergingPeriod, RetryOnRejectBestEffort, RetryOnRejectDuration, SPDPResponseMaxDelay, ScheduleTimeRounding, SecondaryReorderMaxSamples, SquashParticipants, SuppressSPDPMulticast, SynchronousDeliveryLatencyBound, SynchronousDeliveryPriorityThreshold, UnicastResponseToSPDPMessages, UseMulticastIfMreqn, WriterLingerDuration

12.8.5.1 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

12.8.5.2 AggressiveKeepLastWhc

Internal This element controls whether to drop a reliable sample from a DDSI2 WHC before all readers have acknowledged it as soon as a later sample becomes available. It only affects DCPS data writers with a history QoS setting of KEEP_LAST with depth 1. The default setting, *false*, mimics the behaviour of the Vortex OpenSplice RT networking and is necessary to make the behaviour of wait_for_acknowledgements() consistent across the networking services.

- Full path: //OpenSplice/DDSI2Service/Internal/AggressiveKeepLastWhc
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.3 AssumeMulticastCapable

Internal This element controls which network interfaces are assumed to be capable of multicasting even when the interface flags returned by the operating system state it is not (this provides a workaround for some platforms). It is a comma-separated lists of patterns (with ? and * wildcards) against which the interface names are matched.

- Full path: //OpenSplice/DDSI2Service/Internal/AssumeMulticastCapable
- Format: string
- Occurrences min-max: 0-1

12.8.5.4 AutoReschedNackDelay

Internal This setting controls the interval with which a reader will continue NACK'ing missing samples in the absence of a response from the writer, as a protection mechanism against writers incorrectly stopping the sending of HEARTBEAT messages.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/AutoReschedNackDelay
- Format: string
- Default value: 1 s
- Occurrences min-max: 0-1

12.8.5.5 BuiltinEndpointSet

Internal This element 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. See also Internal/ConservativeBuiltinReaderStartup.

- Full path: //OpenSplice/DDSI2Service/Internal/BuiltinEndpointSet
- Format: enumeration
- Default value: writers

- Valid values: full, writers, minimal
- Occurrences min-max: 0-1

12.8.5.6 ConservativeBuiltinReaderStartup

Internal This element forces all DDSI2 built-in discovery-related readers to request all historical data, instead of just one for each "topic". There is no indication that any of the current DDSI implementations requires changing of this setting, but it is conceivable that an implementation might track which participants have been informed of the existence of endpoints and which have not been, refusing communication with those that have "can't" know.

Should it be necessary to hide DDSI2's shared discovery behaviour, set this to *true* and Internal/BuiltinEndpointSet to *full*.

- Full path: //OpenSplice/DDSI2Service/Internal/ConservativeBuiltinReaderStartup
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.7 ControlTopic

Internal The ControlTopic element allows configured whether DDSI2 provides a special control interface via a predefined topic or not.

- Full path: //OpenSplice/DDSI2Service/Internal/ControlTopic
- Occurrences min-max: 0-1
- Child elements: Deaf, Mute
- Optional attributes: enable

enable

Internal This attribute controls whether the DDSI2 control topic is defined and acted upon by DDSI2

- Full path: //OpenSplice/DDSI2Service/Internal/ControlTopic[@enable]
- Format: boolean
- Default value: false
- Required: false

Deaf

Internal This element controls whether DDSI2 defaults to deaf mode or to normal mode. This controls both the initial behaviour and what behaviour it auto-reverts to.

- Full path: //OpenSplice/DDSI2Service/Internal/ControlTopic/Deaf
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

Mute

Internal This element controls whether DDSI2 defaults to mute mode or to normal mode. This controls both the initial behaviour and what behaviour it auto-reverts to.

- Full path: //OpenSplice/DDSI2Service/Internal/ControlTopic/Mute
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.8 DDSI2DirectMaxThreads

Internal This element sets the maximum number of extra threads for an experimental, undocumented and unsupported direct mode.

- Full path: //OpenSplice/DDSI2Service/Internal/DDSI2DirectMaxThreads
- Format: integer
- Default value: 1
- Occurrences min-max: 0-1

12.8.5.9 DefragReliableMaxSamples

Internal This element sets the maximum number of samples that can be defragmented simultaneously for a reliable writer. This has to be large enough to handle retransmissions of historical data in addition to new samples.

- Full path: //OpenSplice/DDSI2Service/Internal/DefragReliableMaxSamples
- Format: integer
- Default value: 16
- Occurrences min-max: 0-1

12.8.5.10 DefragUnreliableMaxSamples

Internal This element sets the maximum number of samples that can be defragmented simultaneously for a best-effort writers.

- Full path: //OpenSplice/DDSI2Service/Internal/DefragUnreliableMaxSamples
- Format: integer
- Default value: 4
- Occurrences min-max: 0-1

12.8.5.11 DeliveryQueueMaxSamples

Internal This element controls the Maximum size of a delivery queue, expressed in samples. Once a delivery queue is full, incoming samples destined for that queue are dropped until space becomes available again.

- Full path: //OpenSplice/DDSI2Service/Internal/DeliveryQueueMaxSamples
- Format: integer
- Default value: 256
- Occurrences min-max: 0-1

12.8.5.12 ForwardAllMessages

Internal Forward all messages from a writer, rather than trying to forward each sample only once. The default of trying to forward each sample only once filters out duplicates for writers in multiple partitions under nearly all circumstances, but may still publish the odd duplicate. Note: the current implementation also can lose in contrived test cases, that publish more than 2**32 samples using a single data writer in conjunction with carefully controlled management of the writer history via cooperating local readers.

- Full path: //OpenSplice/DDSI2Service/Internal/ForwardAllMessages
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.13 ForwardRemoteData

Internal This element controls whether DDSI2 forwards data received from other network services in the domain.

- Full path: //OpenSplice/DDSI2Service/Internal/ForwardRemoteData
- Format: enumeration
- Default value: default
- Valid values: false, true, default
- Occurrences min-max: 0-1

12.8.5.14 GenerateKeyhash

Internal When true, include keyhashes in outgoing data for topics with keys.

- Full path: //OpenSplice/DDSI2Service/Internal/GenerateKeyhash
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.5.15 HeartbeatInterval

Internal This element sets the base interval for the asynchronous, periodic writer heartbeats when unacknowledged data is present in its writer history cache. The actual interval is dynamically adjusted, the attributes of this element allow further configuration.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/HeartbeatInterval
- Format: string
- Default value: 100 ms
- Occurrences min-max: 0-1
- Optional attributes: max, min, minsched

max

Internal This attribute sets the maximum interval for periodic heartbeats.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/HeartbeatInterval[@max]
- Format: string
- Default value: 8 s
- Required: false

min

Internal This attribute sets the minimum interval that must have passed since the most recent heartbeat from a writer, before another asynchronous (not directly related to writing) will be sent.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/HeartbeatInterval[@min]
- Format: string
- Default value: 5 ms
- Required: false

minsched

Internal This attribute sets the minimum interval for periodic heartbeats. Other events may still cause heartbeats to go out.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/HeartbeatInterval[@minsched]
- Format: string
- Default value: 20 ms
- Required: false

12.8.5.16 LateAckMode

Internal Ack a sample only when it has been delivered, instead of when committed to delivering it.

- Full path: //OpenSplice/DDSI2Service/Internal/LateAckMode
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.17 LeaseDuration

Internal This setting controls the default participant lease duration. The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/LeaseDuration
- Format: string
- Default value: 0 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.5.18 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
- Occurrences min-max: 0-1

12.8.5.19 LogStackTraces

Internal This element controls whether or not to write stack traces to the DDSI2 trace when a thread fails to make progress (on select platforms only).

- Full path: //OpenSplice/DDSI2Service/Internal/LogStackTraces
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.5.20 MaxParticipants

Internal This elements configures the maximum number of DCPS domain participants this DDSI2 instance is willing to service. 0 is unlimited.

- Full path: //OpenSplice/DDSI2Service/Internal/MaxParticipants
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.8.5.21 MaxQueuedRexmitBytes

Internal This setting limits the maximum number of bytes queued for retransmission. The default value of 0 is unlimited unless an AuxiliaryBandwidthLimit has been set, in which case it becomes NackDelay * AuxiliaryBandwidthLimit. It must be large enough to contain the largest sample that may need to be retransmitted.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/MaxQueuedRexmitBytes

- Format: string
- Default value: 50 kB
- Occurrences min-max: 0-1

12.8.5.22 MaxQueuedRexmitMessages

Internal This settings limits the maximum number of samples queued for retransmission.

- Full path: //OpenSplice/DDSI2Service/Internal/MaxQueuedRexmitMessages
- Format: integer
- Default value: 200
- Occurrences min-max: 0-1

12.8.5.23 MaxSampleSize

Internal This setting controls the maximum (CDR) serialised size of samples that DDSI2 will forward in either direction. Samples larger than this are discarded with a warning.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/MaxSampleSize
- Format: string
- Default value: 2147483647 B
- Occurrences min-max: 0-1

12.8.5.24 MeasureHbToAckLatency

Internal This element enables heartbeat-to-ack latency among DDSI2 services by prepending timestamps to Heartbeat and AckNack messages and calculating round trip times. This is non-standard behaviour. The measured latencies are quite noisy and are currently not used anywhere.

- Full path: //OpenSplice/DDSI2Service/Internal/MeasureHbToAckLatency
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.25 MinimumSocketReceiveBufferSize

Internal This setting controls the minimum size of socket receive buffers. The operating system provides some size receive buffer upon creation of the socket, this option can be used to increase the size of the buffer beyond that initially provided by the operating system. If the buffer size cannot be increased to the specified size, an error is reported.

The default setting is the word "default", which means DDSI2 will attempt to increase the buffer size to 1MB, but will silently accept a smaller buffer should that attempt fail.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/MinimumSocketReceiveBufferSize
- Format: string

- Default value: default
- Occurrences min-max: 0-1

12.8.5.26 MinimumSocketSendBufferSize

Internal This setting controls the minimum size of socket send buffers. This setting can only increase the size of the send buffer, if the operating system by default creates a larger buffer, it is left unchanged.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/MinimumSocketSendBufferSize
- Format: string
- Default value: 64 KiB
- Occurrences min-max: 0-1

12.8.5.27 MirrorRemoteEntities

Internal This element controls whether DDSI2 mirrors all entities in the domain in DDSI or only local ones. Default is to discover remote ones iff General/LocalDiscoveryPartition is not the built-in partition.

- Full path: //OpenSplice/DDSI2Service/Internal/MirrorRemoteEntities
- Format: enumeration
- Default value: default
- Valid values: false, true, default
- Occurrences min-max: 0-1

12.8.5.28 MonitorPort

Internal This element allows configuring a service that dumps a text description of part the internal state to TCP clients. By default (-1), this is disabled; specifying 0 means a kernel-allocated port is used; a positive number is used as the TCP port number.

- Full path: //OpenSplice/DDSI2Service/Internal/MonitorPort
- Format: integer
- Default value: -1
- Occurrences min-max: 0-1

12.8.5.29 NackDelay

Internal This setting controls the delay between receipt of a HEARTBEAT indicating missing samples and a NACK (ignored when the HEARTBEAT requires an answer). However, no NACK is sent if a NACK had been scheduled already for a response earlier than the delay requests: then that NACK will incorporate the latest information.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/NackDelay
- Format: string
- Default value: 10 ms
- Valid values: 0 / 1hr

- Occurrences min-max: 0-1

12.8.5.30 PreEmptiveAckDelay

Internal This setting controls the delay between the discovering a remote writer and sending a pre-emptive AckNack to discover the range of data available.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/PreEmptiveAckDelay
- Format: string
- Default value: 10 ms
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.5.31 PrimaryReorderMaxSamples

Internal This element sets the maximum size in samples of a primary re-order administration. Each proxy writer has one primary re-order administration to buffer the packet flow in case some packets arrive out of order. Old samples are forwarded to secondary re-order administrations associated with readers in need of historical data.

- Full path: //OpenSplice/DDSI2Service/Internal/PrimaryReorderMaxSamples
- Format: integer
- Default value: 64
- Occurrences min-max: 0-1

12.8.5.32 PrioritizeRetransmit

Internal This element controls whether retransmits are prioritized over new data, speeding up recovery.

- Full path: //OpenSplice/DDSI2Service/Internal/PrioritizeRetransmit
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.5.33 RediscoveryBlacklistDuration

Internal This element controls for how long a remote participant that was previously deleted will remain on a blacklist to prevent rediscovery, giving the software on a node time to perform any cleanup actions it needs to do. To some extent this delay is required internally by DDSI2, but in the default configuration with the 'enforce' attribute set to false, DDSI2 will reallow rediscovery as soon as it has cleared its internal administration. Setting it to too small a value may result in the entry being pruned from the blacklist before DDSI2 is ready, it is therefore recommended to set it to at least several seconds.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/RediscoveryBlacklistDuration
- Format: string
- Default value: 10s
- Occurrences min-max: 0-1

- Optional attributes: enforce

enforce

Internal This attribute controls whether the configured time during which recently deleted participants will not be rediscovered (i.e., "black listed") is enforced and following complete removal of the participant in DDSI2, or whether it can be rediscovered earlier provided all traces of that participant have been removed already.

- Full path: //OpenSplice/DDSI2Service/Internal/RediscoveryBlacklistDuration[@enforce]
- Format: boolean
- Default value: false
- Required: false

12.8.5.34 ResponsivenessTimeout

Internal This element controls for how long an unresponsive reader can block the transmit thread by failing to acknowledge data when a writer's DDSI2 write cache is full. If after this time the writer's cache has not shrunk to below the low-water mark, the reader is considered unresponsive and degraded to unreliable. It will be restored to reliable service once it resumes acknowledging samples.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/ResponsivenessTimeout
- Format: string
- Default value: inf
- Occurrences min-max: 0-1

12.8.5.35 RetransmitMerging

Internal This elements controls the addressing and timing of retransmits. Possible values are:

- *never*: retransmit only to the NACK-ing reader;
- *adaptive*: attempt to combine retransmits needed 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*. See also Internal/RetransmitMergingPeriod.

- Full path: //OpenSplice/DDSI2Service/Internal/RetransmitMerging
- Format: enumeration
- Default value: adaptive
- Valid values: never, adaptive, always
- Occurrences min-max: 0-1

12.8.5.36 RetransmitMergingPeriod

Internal This setting determines the size of the time window in which a NACK of some sample is ignored because a retransmit of that sample has been multicasted too recently. This setting has no effect on unicasted retransmits.

See also Internal/RetransmitMerging.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/RetransmitMergingPeriod
- Format: string
- Default value: 5 ms
- Valid values: 0 / 1s
- Occurrences min-max: 0-1

12.8.5.37 RetryOnRejectBestEffort

Internal Whether or not to locally retry pushing a received best-effort sample into the reader caches when resource limits are reached.

- Full path: //OpenSplice/DDSI2Service/Internal/RetryOnRejectBestEffort
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.38 RetryOnRejectDuration

Internal How long to keep locally retrying pushing a received sample into the reader caches when resource limits are reached. Default is dependent on Internal/LateAckMode: if the latter is false, it is 80% of Internal/ResponsivenessTimeout, otherwise it is 0.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/RetryOnRejectDuration
- Format: string
- Default value: default
- Occurrences min-max: 0-1

12.8.5.39 SPDPResponseMaxDelay

Internal Maximum pseudo-random delay in milliseconds between discovering a remote participant and responding to it.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/SPDPResponseMaxDelay
- Format: string
- Default value: 0 ms
- Valid values: 0 / 1s
- Occurrences min-max: 0-1

12.8.5.40 ScheduleTimeRounding

Internal This setting allows 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 and causes no rounding at all, i.e. are scheduled exactly, whereas a value of 10ms would mean that events are rounded up to the nearest 10 milliseconds.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/ScheduleTimeRounding

- Format: string
- Default value: 0 ms
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.5.41 SecondaryReorderMaxSamples

Internal This element sets the maximum size in samples of a secondary re-order administration. The secondary re-order administration is per reader in need of historical data.

- Full path: //OpenSplice/DDSI2Service/Internal/SecondaryReorderMaxSamples
- Format: integer
- Default value: 16
- Occurrences min-max: 0-1

12.8.5.42 SquashParticipants

Internal This element controls whether DDSI2 advertises all the domain participants it serves in DDSI (when set to *false*), or rather only one domain participant (the one corresponding to the DDSI2 process; when set to *true*). In the latter case DDSI2 becomes the virtual owner of all readers and writers of all domain participants, dramatically reducing discovery traffic (a similar effect can be obtained by setting Internal/BuiltinEndpointSet to "minimal" but with less loss of information).

- Full path: //OpenSplice/DDSI2Service/Internal/SquashParticipants
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.43 SuppressSPDPMulticast

Internal The element controls whether the mandatory multicasting of the participant discovery packets occurs. Completely disabling multicasting requires this element be set to *true*, and generally requires explicitly listing peers to ping for unicast discovery.

See also General/AllowMulticast.

- Full path: //OpenSplice/DDSI2Service/Internal/SuppressSPDPMulticast
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.5.44 SynchronousDeliveryLatencyBound

Internal This element controls whether samples sent by a writer with QoS settings transport_priority >= SynchronousDeliveryPriorityThreshold and a latency_budget at most this element's value will be delivered synchronously from the "recv" thread, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of aggregate bandwidth.

Valid values are finite durations with an explicit unit or the keyword 'inf' for infinity. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/SynchronousDeliveryLatencyBound

- Format: string
- Default value: inf
- Occurrences min-max: 0-1

12.8.5.45 SynchronousDeliveryPriorityThreshold

Internal This element controls whether samples sent by a writer with QoS settings latency_budget <= SynchronousDeliveryLatencyBound and transport_priority greater than or equal to this element's value will be delivered synchronously from the "recv" thread, all others will be delivered asynchronously through delivery queues. This reduces latency at the expense of aggregate bandwidth.

- Full path: //OpenSplice/DDSI2Service/Internal/SynchronousDeliveryPriorityThreshold
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.8.5.46 Test

Internal Testing options.

- Full path: //OpenSplice/DDSI2Service/Internal/Test
- Occurrences min-max: 0-1
- Child elements: XmitLossiness

XmitLossiness

Internal This element controls the fraction of outgoing packets to drop, specified as samples per thousand.

- Full path: //OpenSplice/DDSI2Service/Internal/Test/XmitLossiness
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1

12.8.5.47 UnicastResponseToSPDPMessages

Internal This element controls whether the response to a newly discovered participant is sent as a unicasted SPDP packet, instead of rescheduling the periodic multicasted one. There is no known benefit to setting this to *false*.

- Full path: //OpenSplice/DDSI2Service/Internal/UnicastResponseToSPDPMessages
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.5.48 UseMulticastIfMreqn

Internal Do not use.

- Full path: //OpenSplice/DDSI2Service/Internal/UseMulticastIfMreqn
- Format: integer

- Default value: 0
- Occurrences min-max: 0-1

12.8.5.49 Watermarks

Internal Watermarks for flow-control.

- Full path: //OpenSplice/DDSI2Service/Internal/Watermarks
- Occurrences min-max: 0-1
- Child elements: WhcAdaptive, WhcHigh, WhcHighInit, WhcLow

WhcAdaptive

Internal This element controls whether DDSI2 will adapt the high-water mark to current traffic conditions, based on retransmit requests and transmit pressure.

- Full path: //OpenSplice/DDSI2Service/Internal/Watermarks/WhcAdaptive
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

WhcHigh

Internal This element sets the maximum allowed high-water mark for the DDSI2 WHCs, expressed in bytes. A writer is suspended when the WHC reaches this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/Watermarks/WhcHigh
- Format: string
- Default value: 100 kB
- Occurrences min-max: 0-1

WhcHighInit

Internal This element sets the initial level of the high-water mark for the DDSI2 WHCs, expressed in bytes.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/Watermarks/WhcHighInit
- Format: string
- Default value: 30 kB
- Occurrences min-max: 0-1

WhcLow

Internal This element sets the low-water mark for the DDSI2 WHCs, expressed in bytes. A suspended writer resumes transmitting when its DDSI2 WHC shrinks to this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Internal/Watermarks/WhcLow
- Format: string
- Default value: 1 kB
- Occurrences min-max: 0-1

12.8.5.50 WriterLingerDuration

Internal This setting controls the maximum duration for which actual deletion of a reliable writer with unacknowledged data in its history will be postponed to provide proper reliable transmission. The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/Internal/WriterLingerDuration
- Format: string
- Default value: 1 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.6 SSL

The SSL element allows specifying various parameters related to using SSL/TLS for DDSI over TCP.

- Full path: //OpenSplice/DDSI2Service/SSL
- Occurrences min-max: 0-1
- Child elements: CertificateVerification, Ciphers, Enable, EntropyFile, KeyPassphrase, KeystoreFile, SelfSignedCertificates, VerifyClient

12.8.6.1 CertificateVerification

If disabled this allows SSL connections to occur even if an X509 certificate fails verification.

- Full path: //OpenSplice/DDSI2Service/SSL/CertificateVerification
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.6.2 Ciphers

The set of ciphers used by SSL/TLS

- Full path: //OpenSplice/DDSI2Service/SSL/Ciphers
- Format: string
- Default value: ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH
- Occurrences min-max: 0-1

12.8.6.3 Enable

This enables SSL/TLS for TCP.

- Full path: //OpenSplice/DDSI2Service/SSL/Enable
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.6.4 EntropyFile

The SSL/TLS random entropy file name.

- Full path: //OpenSplice/DDSI2Service/SSL/EntropyFile
- Format: string
- Occurrences min-max: 0-1

12.8.6.5 KeyPassphrase

The SSL/TLS key pass phrase for encrypted keys.

- Full path: //OpenSplice/DDSI2Service/SSL/KeyPassphrase
- Format: string
- Default value: secret
- Occurrences min-max: 0-1

12.8.6.6 KeystoreFile

The SSL/TLS key and certificate store file name. The keystore must be in PEM format.

- Full path: //OpenSplice/DDSI2Service/SSL/KeystoreFile
- Format: string
- Default value: keystore
- Occurrences min-max: 0-1

12.8.6.7 SelfSignedCertificates

This enables the use of self signed X509 certificates.

- Full path: //OpenSplice/DDSI2Service/SSL/SelfSignedCertificates
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.6.8 VerifyClient

This enables an SSL server checking the X509 certificate of a connecting client.

- Full path: //OpenSplice/DDSI2Service/SSL/VerifyClient
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.7 Sizing

The Sizing element specifies a variety of configuration settings dealing with expected system sizes, buffer sizes, &c.

- Full path: //OpenSplice/DDSI2Service/Sizing
- Occurrences min-max: 0-1
- Child elements: EndpointsInSystem, LocalEndpoints, NetworkQueueSize, ReceiveBufferChunkSize

12.8.7.1 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

12.8.7.2 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; signficantly 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

12.8.7.3 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

12.8.7.4 ReceiveBufferChunkSize

This element specifies the 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). Each allocation is shrunk immediately after processing a message, or freed straightaway.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Sizing/ReceiveBufferChunkSize
- Format: string
- Default value: 128 KiB
- Occurrences min-max: 0-1

12.8.7.5 Watermarks

{ LEAF ("ReceiveBufferSize"), 1, "1 MiB", ABSOFF (rbuf_size), 0, uf_memsize, 0, pf_memsize, "This element sets the size of a single receive buffer. Many receive buffers may be needed. Their size must be greater than ReceiveBufferChunkSize by a modest amount.

- Full path: //OpenSplice/DDSI2Service/Sizing/Watermarks
- Occurrences min-max: 0-1
- Child elements: WhcAdaptive, WhcHigh, WhcHighInit, WhcLow

WhcAdaptive

This element controls whether DDSI2 will adapt the high-water mark to current traffic conditions, based on retransmit requests and transmit pressure.

- Full path: //OpenSplice/DDSI2Service/Sizing/Watermarks/WhcAdaptive
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

WhcHigh

This element sets the maximum allowed high-water mark for the DDSI2 WHCs, expressed in bytes. A writer is suspended when the WHC reaches this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Sizing/Watermarks/WhcHigh
- Format: string
- Default value: 100 kB
- Occurrences min-max: 0-1

WhcHighInit

This element sets the initial level of the high-water mark for the DDSI2 WHCs, expressed in bytes.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Sizing/Watermarks/WhcHighInit
- Format: string
- Default value: 30 kB
- Occurrences min-max: 0-1

WhcLow

This element sets the low-water mark for the DDSI2 WHCs, expressed in bytes. A suspended writer resumes transmitting when its DDSI2 WHC shrinks to this size.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Sizing/Watermarks/WhcLow
- Format: string
- Default value: 1 kB
- Occurrences min-max: 0-1

12.8.8 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: Enable, NoDelay, Port, ReadTimeout, WriteTimeout

12.8.8.1 Enable

This element enables the optional TCP transport.

- Full path: //OpenSplice/DDSI2Service/TCP/Enable
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.8.2 NoDelay

This element enables the TCP_NODELAY socket option, preventing multiple DDSI messages being sent in the same TCP request. Setting this option typically optimises latency over throughput.

- Full path: //OpenSplice/DDSI2Service/TCP/NoDelay
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1

12.8.8.3 Port

This element specifies the TCP port number on which DDSI2 accepts connections. If the port is set it is used in entity locators, published with DDSI discovery. Dynamically allocated if zero. Disabled if -1 or not configured. If disabled other DDSI services will not be able to establish connections with the service, the service can only communicate by establishing connections to other services.

- Full path: //OpenSplice/DDSI2Service/TCP/Port
- Format: integer
- Default value: -1
- Valid values: -1 / 65535
- Occurrences min-max: 0-1

12.8.8.4 ReadTimeout

This element specifies the timeout for blocking TCP read operations. If this timeout expires then the connection is closed.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/TCP/ReadTimeout
- Format: string
- Default value: 2 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.8.5 WriteTimeout

This element specifies the timeout for blocking TCP write operations. If this timeout expires then the connection is closed.

The unit must be specified explicitly. Recognised units: ns, us, ms, s, min, hr, day.

- Full path: //OpenSplice/DDSI2Service/TCP/WriteTimeout
- Format: string
- Default value: 2 s
- Valid values: 0 / 1hr
- Occurrences min-max: 0-1

12.8.9 ThreadPool

The ThreadPool element allows specifying various parameters related to using a thread pool to send DDSI messages to multiple unicast addresses (TCP or UDP).

- Full path: //OpenSplice/DDSI2Service/ThreadPool
- Occurrences min-max: 0-1
- Child elements: Enable, ThreadMax, Threads

12.8.9.1 Enable

This element enables the optional thread pool.

- Full path: //OpenSplice/DDSI2Service/ThreadPool/Enable
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.9.2 ThreadMax

This elements configures the maximum number of threads in the thread pool.

- Full path: //OpenSplice/DDSI2Service/ThreadPool/ThreadMax
- Format: integer
- Default value: 8
- Occurrences min-max: 0-1

12.8.9.3 Threads

This elements configures the initial number of threads in the thread pool.

- Full path: //OpenSplice/DDSI2Service/ThreadPool/Threads
- Format: integer
- Default value: 4
- Occurrences min-max: 0-1

12.8.10 Threads

This element is used to set thread properties.

- Full path: //OpenSplice/DDSI2Service/Threads
- Occurrences min-max: 0-1

12.8.10.1 Thread

This element is used to set thread properties.

- Full path: //OpenSplice/DDSI2Service/Threads/Thread
- Occurrences min-max: 0-1000
- Child elements: StackSize
- Required attributes: Name

Name

The Name of the thread for which properties are being set. The following threads exist:

- *gc*: garbage collector thread involved in deleting entities;
- *recv*: receive thread, taking data from the network and running the protocol state machine;
- *dq.builtins*: delivery thread for DDSI-builtin data, primarily for discovery;

- *lease*: DDSI liveliness monitoring;
- *tev*: general timed-event handling, retransmits and discovery;
- *xmit.CHAN*: transmit thread for channel CHAN;
- *dq.CHAN*: delivery thread for channel CHAN;
- *tev.CHAN*: timed-even thread for channel CHAN.
- Full path: //OpenSplice/DDSI2Service/Threads/Thread[@Name]
- Format: string
- Default value: n/a
- Required: true

Scheduling

This element configures the scheduling properties of the thread.

- Full path: //OpenSplice/DDSI2Service/Threads/Thread/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

12.8.10.1.2.1 Class This element specifies the thread scheduling class (*realtime*, *timeshare* or *default*). The user may need special 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

12.8.10.1.2.2 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

StackSize

This element configures the stack size for this thread. The default value *default* leaves the stack size at the operating system default.

The unit must be specified explicitly. Recognised units: B (bytes), kB & KiB (2^{10} bytes), MB & MiB (2^{20} bytes), GB & GiB (2^{30} bytes).

- Full path: //OpenSplice/DDSI2Service/Threads/Thread/StackSize
- Format: string
- Default value: default

- Occurrences min-max: 0-1

12.8.11 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: AppendToFile, EnableCategory, OutputFile, PacketCaptureFile, Timestamps, Verbosity

12.8.11.1 AppendToFile

This option specifies whether the output is to be appended to an existing log file. The default is to create a new log file each time, which is generally the best option if a detailed log is generated.

- Full path: //OpenSplice/DDSI2Service/Tracing/AppendToFile
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.8.11.2 EnableCategory

This element enables individual logging categories. These are enabled in addition to those enabled by Tracing/Verbosity. Recognised categories are:

- *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
- *whc*: tracing of writer history cache changes
- *tcp*: tracing of TCP-specific activity
- *topic*: tracing of topic definitions
- >i>plist*: tracing of discovery parameter list interpretation

In addition, there is the keyword *trace* that enables all but *radmin*, *topic*, *plist* and *whc* . 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. Currently, the most useful is *trace*.

- Full path: //OpenSplice/DDSI2Service/Tracing/EnableCategory
- Format: string

- Occurrences min-max: 0-1

12.8.11.3 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

12.8.11.4 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 ficitious, 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. Currently IPv4 only.

- Full path: //OpenSplice/DDSI2Service/Tracing/PacketCaptureFile
- Format: string
- Occurrences min-max: 0-1

12.8.11.5 Timestamps

This option has no effect.

- Full path: //OpenSplice/DDSI2Service/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

absolute

This option has no effect

- Full path: //OpenSplice/DDSI2Service/Tracing/Timestamps[@absolute]
- Format: boolean
- Default value: true
- Required: false

12.8.11.6 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. Recognised verbosity levels and the categories they map to are:

- *none*: no DDSI2 log

- *severe*: error and fatal
- *warning*: *severe* + warning
- *info*: *warning* + info
- *config*: *info* + config
- *fine*: *config* + discovery
- *finer*: *fine* + traffic and timing
- *finest*: *finer* + trace

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. Currently, the most useful verbosity levels are *config*, *fine* and *finest*.

- Full path: //OpenSplice/DDSI2Service/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, finer, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.8.12 Watchdog

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DDSI2Service/Watchdog
- Occurrences min-max: 0-1

12.8.12.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DDSI2Service/Watchdog/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

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/Scheduling/Class
- Format: enumeration
- Default value: default
- Valid values: realtime, timeshare, default
- Occurrences min-max: 0-1

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/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1
- Optional attributes: priority_kind

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

- Full path: //OpenSplice/DDSI2Service/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: relative
- Valid values: relative, absolute
- Required: false

12.9 TunerService

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-*
- Required attributes: name

12.9.1 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 *OpenSplice/Domain/Service[@name]* in the configuration of the Domain Service.

- Full path: //OpenSplice/TunerService[@name]
- Format: string
- Default value: cmsoap
- Required: true

12.9.2 Watchdog

This element controls the characteristics of the Watchdog thread.

- Full path: //OpenSplice/TunerService/Watchdog
- Occurrences min-max: 0-1

12.9.2.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/TunerService/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.9.2.1.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.9.3 Server

This element determines the serverside behaviour of the Tuner service.

- Full path: //OpenSplice/TunerService/Server
- Occurrences min-max: 0-1

- Child elements: PortNr, Backlog, Verbosity

12.9.3.1 PortNr

This element determines 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. Valid portnumbers are 1 till 65535. When using the single process option set this value to Auto.

- Full path: //OpenSplice/TunerService/Server/PortNr
- Format: string
- Default value: 8000
- Occurrences min-max: 0-1

12.9.3.2 Backlog

This element specifies 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: integer
- Default value: 5
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.9.3.3 Verbosity

This element specifies the verbosity level of the logging of the service.

- Full path: //OpenSplice/TunerService/Server/Verbosity
- Format: enumeration
- Default value: 0
- Valid values: 0, 1, 2, 3, 4, 5
- Occurrences min-max: 0-1

12.9.4 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: MaxClients, MaxThreadsPerClient, LeasePeriod

12.9.4.1 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: integer

- Default value: 10
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.9.4.2 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: integer
- Default value: 10
- Valid values: 1 / -
- Occurrences min-max: 0-1

12.9.4.3 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
- Dimension: seconds
- Default value: 15.0
- Valid values: 10.0 / -
- Occurrences min-max: 0-1

12.9.4.4 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: 0-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1

- Optional attributes: priority_kind

12.9.4.4.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.9.5 GarbageCollector

This element specifies the behaviour of the garbage collection thread of the service.

- Full path: //OpenSplice/TunerService/GarbageCollector
- Occurrences min-max: 0-1

12.9.5.1 Scheduling

This element specifies the scheduling policies used to control the garbagage collection thread of the Tuner Service.

- Full path: //OpenSplice/TunerService/GarbageCollector/Scheduling
- Occurrences min-max: 0-1
- Child elements: Priority, Class

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/GarbageCollector/Scheduling/Priority
- Format: integer
- Default value: 0

- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.9.5.1.1.1 priority_kind

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

- Full path: //OpenSplice/TunerService/GarbageCollector/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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/GarbageCollector/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.9.6 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

12.9.6.1 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: 0-1
- Child elements: Priority, Class

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: integer

- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.9.6.1.1.1 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
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.10 DbmsConnectService

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-*
- Required attributes: name

12.10.1 name

This attribute identifies the configuration for the DBMS connect service by name. Multiple DBMS connect 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 *OpenSplice/Domain/Service[@name]* in the configuration of the DomainService.

- Full path: //OpenSplice/DbmsConnectService[@name]
- Format: string
- Default value: dbmsconnect
- Required: true

12.10.2 Watchdog

This element controls the characteristics of the Watchdog thread

- Full path: //OpenSplice/DbmsConnectService/Watchdog
- Occurrences min-max: 0-1

12.10.2.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/DbmsConnectService/Watchdog/Scheduling
- Occurrences min-max: 1-1
- Child elements: Priority, Class

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: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.10.2.1.1.1 priority_kind

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

- Full path: //OpenSplice/DbmsConnectService/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.10.3 DdsToDbms

This element specifies the configuration properties concerning DDS to DBMS bridging.

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms
- Occurrences min-max: 0-1
- Optional attributes: replication_mode

12.10.3.1 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 DBMS-connect service ignores samples that were published by itself. (Currently that 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
- Default value: FALSE
- Required: false

12.10.3.2 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-*
- Required attributes: dsn, usr, pwd
- Optional attributes: name, odbc, partition, topic, update_frequency, schema, catalog, replication_mode

name

The name of the namespace. If not specified, the namespace will be named "(nameless)".

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@name]
- Format: string
- Default value: (nameless)
- Required: false

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
- Default value: “”
- Required: false

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 one single character.

This expression is used to specify the DDS partition(s) from which DDS samples must be ‘bridged’ to the DBMS domain.

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@partition]
- Format: string
- Default value: *
- Required: false

topic

This attribute specifies an expression that represents one or more DDS topic names. It is allowed to use wildcards in the expression: a ‘*’ represents any sequence of characters and a ‘?’ represents one 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
- Default value: *
- Required: false

update_frequency

This attribute specifies the frequency (in Hz) at which the service will update the DBMS domain with DDS data. By default, it is 0.0Hz which means it is done event based (every time new DDS data arrives).

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@update_frequency]
- Dimension: Hz
- Default value: 0.0

- Valid values: 0.0 / -
- Required: false

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
- Default value: n/a
- Required: true

usr

Represents the user name that is used when connecting to the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@usr]
- Format: string
- Default value: n/a
- Required: true

pwd

Represents the password that is used when connecting to the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace[@pwd]
- Format: string
- Default value: n/a
- Required: true

schema

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
- Default value: “”
- Required: false

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
- Default value: “”
- Required: false

replication_mode

This attribute specifies the replication mode for the current *NameSpace* element. If not specified, the value will be inherited from the *replication_mode* of the parent *DdsToDbms* element, which if not explicitly specified 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 DBMS-connect service ignores samples that were published by itself. (Currently that 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
- Default value: FALSE
- Required: false

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-*
- Required attributes: topic
- Optional attributes: table, query, filter

12.10.3.2.12.1 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
- Default value: n/a
- Required: true

12.10.3.2.12.2 table

This attribute specifies an alternative name for the table that must be associated with the topic. By default the table name is equal to the topic name.

- Full path: //OpenSplice/DbmsConnectService/DdsToDbms/NameSpace/Mapping[@table]
- Format: string
- Default value: “”
- Required: false

12.10.3.2.12.3 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
- Default value: “”
- Required: false

12.10.3.2.12.4 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
- Default value: “”
- Required: false

12.10.4 DbmsToDds

Holds the configuration of the service concerning DBMS to DDS bridging

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds
- Occurrences min-max: 0-1
- Optional attributes: publish_initial_data, event_table_policy, trigger_policy, replication_user

12.10.4.1 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
- Default value: true
- Required: false

12.10.4.2 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
- Default value: FULL
- Valid values: FULL, LAZY, NONE
- Required: false

12.10.4.3 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 exists.
- **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
- Default value: FULL
- Valid values: FULL, LAZY, NONE
- Required: false

12.10.4.4 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
- Default value: “”
- Required: false

12.10.4.5 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-*
- Required attributes: dsn, usr, pwd
- Optional attributes: name, odbc, partition, table, update_frequency, publish_initial_data, force_key_equality, event_table_policy, trigger_policy, schema, catalog, replication_user

name

The name of the namespace. If not specified, the namespace will be named “(nameless)”.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@name]
- Format: string
- Default value: (nameless)
- Required: false

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
- Default value: “”
- Required: false

partition

This attribute specifies an expression represents one or more DDS partitions. It is allowed to use wildcards in the expression: a ‘*’ represents any sequence of characters and a ‘?’ represents one 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
- Default value: *
- Required: false

table

This attribute specifies an expression that represents one or more DBMS table names. It is allowed to use wildcards in the expression: a ‘*’ represents any sequence of characters and a ‘?’ represents one 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
- Default value: *
- Required: false

update_frequency

This attribute specifies the frequency (in Hz) at which the service will update the DDS domain with DBMS data. By default, it 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_frequency]
- Dimension: Hz
- Default value: 2.0
- Valid values: 0.0 / -
- Required: false

dsn

Represents the 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

- Default value: n/a
- Required: true

usr

Represents the user name that is used when connecting to the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@usr]
- Format: string
- Default value: n/a
- Required: true

pwd

Represents the password that is used when connecting to the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@pwd]
- Format: string
- Default value: n/a
- Required: true

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 will be inherited from the *publish_initial_data* of the parent *DbmsToDds* element, which 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
- Default value: true
- Required: false

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
- Default value: true
- Required: false

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 will be inherited from the *event_table_policy* of the parent *DbmsToDds* element, which if not explicitly specified 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
- Default value: FULL
- Valid values: FULL, LAZY, NONE
- Required: false

trigger_policy

This attribute specifies the default trigger policy for all *Mapping* elements in the current *NameSpace* element. If not specified, the value will be inherited from the *trigger_policy* of the parent *DbmsToDds* element, which if not explicitly specified 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**: *(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 exists.
- **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
- Default value: FULL
- Valid values: FULL, LAZY, NONE
- Required: false

schema

Represents the schema that is used when communicating with the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@schema]
- Format: string
- Default value: “”
- Required: false

catalog

Represents the catalog that is used when communicating with the DBMS.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace[@catalog]
- Format: string
- Default value: “”
- Required: false

replication_user

This attribute specifies the default replication user for all *Mapping* elements in the current *NameSpace* element. If not specified, the value will be inherited from the *replication_user* of the parent *DbmsToDds* element, which by default has no separate replication user specified.

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
- Default value: “”
- Required: false

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-*
- Required attributes: table

- Optional attributes: topic, query, publish_initial_data, force_key_equality, event_table_policy, trigger_policy

12.10.4.5.16.1 table

This attribute specifies the name of the table where the Mapping applies to.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@table]
- Format: string
- Default value: n/a
- Required: true

12.10.4.5.16.2 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
- Default value: “”
- Required: false

12.10.4.5.16.3 query

Optional DBMS 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
- Default value: “”
- Required: false

12.10.4.5.16.4 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 will be inherited from the *publish_initial_data* of the parent *NameSpace* element, which if not explicitly specified inherits from the *publish_initial_data* of the parent *DbmsToDds* element, which 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
- Default value: true
- Required: false

12.10.4.5.16.5 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 will be inherited from the *force_key_equality* of the parent *NameSpace* element, which if not explicitly specified defaults to true.

- Full path: //OpenSplice/DbmsConnectService/DbmsToDds/NameSpace/Mapping[@force_key_equality]

- Format: boolean
- Default value: true
- Required: false

12.10.4.5.16.6 event_table_policy

This attribute specifies the event table policy in the current *Mapping* element. If not specified, the value will be inherited from the *event_table_policy* of the parent *NameSpace* element, which if not explicitly specified inherits from the *event_table_policy* of the parent *DbmsToDds* element, which 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
- Default value: FULL
- Valid values: FULL, LAZY, NONE
- Required: false

12.10.4.5.16.7 trigger_policy

This attribute specifies the trigger policy for the current *Mapping* element. If not specified, the value will be inherited from the *trigger_policy* of the parent *DbmsToDds* element, which if not explicitly specified inherits from the *trigger_policy* of the parent *DbmsToDds* element, which 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**: *(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 exists.
- **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
- Default value: FULL

- Valid values: FULL, LAZY, NONE
- Required: false

12.10.5 Tracing

This element controls the amount and type of information that is written into the tracing log file by the DbmsConnect Service. This is useful to track the DbmsConnect Service during application development. In the runtime system it should be turned off.

- Full path: //OpenSplice/DbmsConnectService/Tracing
- Occurrences min-max: 0-1
- Child elements: OutputFile, Timestamps, Verbosity

12.10.5.1 OutputFile

This element specifies where the logging is printed to. Note that "stdout" and "stderr" are considered legal values that represent "standard out" and "standard error" respectively. The default value is an empty string, indicating that all tracing is disabled.

- Full path: //OpenSplice/DbmsConnectService/Tracing/OutputFile
- Format: string
- Dimension: file name
- Default value: dbmsconnect.log
- Occurrences min-max: 0-1

12.10.5.2 Timestamps

This element specifies whether the logging must contain timestamps.

- Full path: //OpenSplice/DbmsConnectService/Tracing/Timestamps
- Format: boolean
- Default value: true
- Occurrences min-max: 0-1
- Optional attributes: absolute

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
- Default value: true
- Required: false

12.10.5.3 Verbosity

This element specifies the verbosity level of the logging.

- Full path: //OpenSplice/DbmsConnectService/Tracing/Verbosity
- Format: enumeration
- Default value: INFO
- Valid values: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
- Occurrences min-max: 0-1

12.11 RnRService

The Record and Replay Service allows 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-*
- Required attributes: name

12.11.1 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 under the *OpenSplice/Domain/Service[@name]* in the configuration of the Domain Service.

- Full path: //OpenSplice/RnRService[@name]
- Format: string
- Default value: rnr
- Required: true

12.11.2 Watchdog

This element controls the characteristics of the Watchdog thread.

- Full path: //OpenSplice/RnRService/Watchdog
- Occurrences min-max: 0-1

12.11.2.1 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: Priority, Class

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
- Format: integer
- Default value: 0
- Occurrences min-max: 1-1
- Optional attributes: priority_kind

12.11.2.1.1.1 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: enumeration
- Default value: Relative
- Valid values: Relative, Absolute
- Required: false

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/RnRService/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: Default
- Valid values: Timeshare, Realtime, Default
- Occurrences min-max: 1-1

12.11.3 Storage

This element specifies a storage to use for recording and/or replaying data. Currently the supported storage backends are XML and CDR. 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 Vortex OpenSplice configuration file, so the description given here also applies to commands.

- Full path: //OpenSplice/RnRService/Storage
- Occurrences min-max: 0-*
- Required attributes: name

12.11.3.1 name

The name used to identify the storage in Record and Replay commands.

- Full path: //OpenSplice/RnRService/Storage[@name]
- Format: string
- Default value: default
- Required: true

12.11.3.2 rr_storageAttrXML

Attributes describing an XML storage.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrXML
- Occurrences min-max: 0-1
- Child elements: filename, MaxFileSize

filename

The filename template used for files that comprise a storage. The filename may contain a relative or absolute path. If a path is omitted, the storage files are created in the current working directory.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrXML/filename
- Format: string
- Default value: rnr-storage.dat
- Occurrences min-max: 1-1

MaxFileSize

The maximum size per storage file. When approaching the maximum size while recording, a new storage file is automatically created with a sequence number appended to the filename. The active file is also switched transparently while replaying from a storage that contains multiple data files. 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. This element is optional, when omitted or when 0 is configured, the file size is not monitored by the service and limited only by filesystem and/or platform-specific limits.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrXML/MaxFileSize
- Dimension: bytes
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.11.3.3 rr_storageAttrCDR

Attributes describing an CDR storage.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrCDR
- Occurrences min-max: 0-1
- Child elements: filename, MaxFileSize

filename

The filename template used for files that comprise a storage. The filename may contain a relative or absolute path. If a path is omitted, the storage files are created in the current working directory.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrCDR/filename
- Format: string
- Default value: rnr-storage.dat
- Occurrences min-max: 1-1

MaxFileSize

The maximum size per storage file. When approaching the maximum size while recording, a new storage file is automatically created with a sequence number appended to the filename. The active file is also switched transparently while replaying from a storage that contains multiple data files. 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. This element is optional, when omitted or when 0 is configured, the file size is not monitored by the service and limited only by filesystem and/or platform-specific limits.

- Full path: //OpenSplice/RnRService/Storage/rr_storageAttrCDR/MaxFileSize
- Dimension: bytes
- Default value: 0
- Valid values: 0 / -
- Occurrences min-max: 0-1

12.11.3.4 Statistics

Maintain and optionally publish statistics for this storage.

- Full path: //OpenSplice/RnRService/Storage/Statistics
- Occurrences min-max: 0-1
- Optional attributes: enabled, publish_interval, reset

enabled

This attribute specifies if statistics should be maintained for this storage.

- Full path: //OpenSplice/RnRService/Storage/Statistics[@enabled]
- Format: boolean
- Default value: true
- Occurrences min-max: 1-1
- Required: false

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. This means the statistics are published when the storage is closed. Note that a value of 0 means statistics are never published.

- Full path: //OpenSplice/RnRService/Storage/Statistics[@publish_interval]
- Format: integer
- Dimension: seconds
- Default value: 30
- Occurrences min-max: 1-1
- Required: false

reset

This attribute allows 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 Vortex OpenSplice configuration file always start out with empty statistics.

- Full path: //OpenSplice/RnRService/Storage/Statistics[@reset]
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1
- Required: false

12.11.4 Tracing

This element enables debug output of the R&R service to a logfile.

- Full path: //OpenSplice/RnRService/Tracing
- Occurrences min-max: 0-1
- Child elements: OutputFile, AppendToFile, Verbosity, EnableCategory

12.11.4.1 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".

- Full path: //OpenSplice/RnRService/Tracing/OutputFile
- Format: string
- Default value: rnr.log
- Occurrences min-max: 1-1

12.11.4.2 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
- Default value: false
- Occurrences min-max: 0-1

12.11.4.3 Verbosity

This element 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
- Default value: INFO
- Valid values: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, NONE
- Occurrences min-max: 0-1

12.11.4.4 EnableCategory

This option allows to enable specific logging 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: string
- Occurrences min-max: 0-1

12.12 Agent

The root element of a Control and Monitoring Agent configuration.

- Full path: //OpenSplice/Agent
- Occurrences min-max: 0-1
- Required attributes: name

12.12.1 name

This attribute identifies the configuration for the Control and Monitoring Agent. Multiple service configurations can be specified in one single XML file. The actual applicable configuration is determined by the value of the name attribute, which must match the string specified in the element OpenSplice/Domain/Service[@name] in the Domain Service configuration.

- Full path: //OpenSplice/Agent[@name]
- Format: string
- Default value: cmagent

- Required: true

12.12.2 Tracing

The Tracing element controls the amount and type of information that is written into the tracing log by the Control and Monitoring Agent service. This is useful to track the service during application development.

- Full path: //OpenSplice/Agent/Tracing
- Occurrences min-max: 0-1
- Child elements: EnableCategory, Verbosity, OutputFile, AppendToFile

12.12.2.1 EnableCategory

This element enables individual logging categories. These are enabled in addition to those enabled by Tracing/Verbosity. Recognised categories are:- *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 In addition, there is the keyword *trace* that enables all categories

- Full path: //OpenSplice/Agent/Tracing/EnableCategory
- Format: string
- Occurrences min-max: 0-1

12.12.2.2 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. Recognised verbosity levels and the categories they map to are:- *none*: no Control and Monitoring Agent log- *severe*: error and fatal- *warning*: *severe* + warning- *info*: *warning* + general information messages- *config*: *info* + config- *fine*: equivalent to *config*- *finest*: *fine* + trace While *none* prevents any message from being written to a Control and Monitoring Agent log file, warnings and errors are still logged in the ospl-info.log and ospl-error.log files.

- Full path: //OpenSplice/Agent/Tracing/Verbosity
- Format: enumeration
- Default value: none
- Valid values: finest, finer, fine, config, info, warning, severe, none
- Occurrences min-max: 0-1

12.12.2.3 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/Agent/Tracing/OutputFile
- Format: string
- Dimension: file path
- Default value: cmagent.log
- Occurrences min-max: 0-1

12.12.2.4 AppendToFile

This option specifies whether the output is to be appended to an existing log file. The default is to create a new log file each time, which is generally the best option if a detailed log is generated.

- Full path: //OpenSplice/Agent/Tracing/AppendToFile
- Format: boolean
- Default value: false
- Occurrences min-max: 0-1

12.12.3 Watchdog

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/Agent/Watchdog
- Occurrences min-max: 0-1

12.12.3.1 Scheduling

This element specifies the type of OS scheduling class will be used by the thread that announces its liveliness periodically.

- Full path: //OpenSplice/Agent/Watchdog/Scheduling
- Occurrences min-max: 0-1
- Child elements: Class, Priority

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/Agent/Watchdog/Scheduling/Class
- Format: enumeration
- Default value: default
- Valid values: realtime, timeshare, default
- Occurrences min-max: 0-1

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/Agent/Watchdog/Scheduling/Priority
- Format: integer
- Default value: 0
- Occurrences min-max: 0-1
- Required attributes: priority_kind

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

- Full path: //OpenSplice/Agent/Watchdog/Scheduling/Priority[@priority_kind]
- Format: enumeration
- Default value: relative
- Valid values: relative, absolute
- Required: true

13

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.

13.1 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 shut down without any extra configuration because the default discovery mechanism will keep track of the networking topology.

13.2 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.

13.3 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.

13.3.1 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.

13.3.2 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.

13.3.3 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.

13.3.4 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.

14

Logrotate

*The OpenSplice middleware can produce several trace and log files depending on the applied configuration settings. Log files and in particular trace files can become very large over time and run into resource limitations. It is advised to use the logrotate function to manage resource consumption of log and trace files. The logrotate function is standard available on linux distributions and for windows an opensource version is available on sourceforge (LogRotateWin).**

14.1 Description

logrotate is designed to ease administration of systems that generate large numbers of log files. It allows automatic rotation, compression, removal, and mailing of log files. Each log file may be handled daily, weekly, monthly, or when it grows too large. Normally, logrotate is run as a daily cron job. It will not modify a log multiple times in one day unless the criterion for that log is based on the log's size and logrotate is being run multiple times each day, or unless the -f or –forceoption is used. For example, logrotate can also be run by a supervisory process at any time to process log files specified in the config_file. For a more detailed description of logrotate see the linux man pages.

14.2 Configuration file

logrotate reads everything about the log files it should be handling from the series of configuration files specified on the command line. Each configuration file can set global options (local definitions override global ones, and later definitions override earlier ones) and specify log files to rotate.

14.3 Example configuration

For OpenSplice the following example config_file arguments are advised for logrotate:

```
# sample logrotate configuration file

# The copytruncate option specifies that logfiles are first copied and
# then truncate. This option is required for OpenSplice instead of using
# the create option to avoid closing the file descriptors used by
# OpenSplice.
#
copytruncate

# The compress option is used to compress the logfile copies.
compress

# The following options specify that the log files in the current directory
# will be rotates when the file size exceeds 100k and that at most the 5
# most recent rotated files are maintained.
"./\*.log" {
```

```
    rotate 5
    size 100k
}
```

15

Contacts & Notices

15.1 Contacts

ADLINK Technology Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA
Tel: +1 781 569 5819

ADLINK Technology Limited
The Edge
5th Avenue
Team Valley
Gateshead
NE11 0XA
UK
Tel: +44 (0)191 497 9900

ADLINK Technology SARL
28 rue Jean Rostand
91400 Orsay
France
Tel: +33 (1) 69 015354

Web: http://ist.adlinktech.com/

Contact: http://ist.adlinktech.com

E-mail: ist_info@adlinktech.com

LinkedIn: https://www.linkedin.com/company/79111/

Twitter: https://twitter.com/ADLINKTech_usa

Facebook: https://www.facebook.com/ADLINKTECH

15.2 Notices