Date: March 2023

DDS Extensions for Time Sensitive Networking (DDS-TSN)

Version 1.0 - beta 1

OMG Document Number:	ptc/2023-03-03
Normative Reference:	https://www.omg.org/spec/DDS-TSN/1.0

This OMG document replaces the submission document (mars/2022-12-03). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome and should be directed to issues@omg.org by June 30, 2023.

You may view the pending issues for this specification from the OMG revision issues web page https://issues.omg.org/issues/lists.

The FTF Recommendation and Report for this specification will be published in December 2023. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

Table of Contents

Table of Figures

Table of Tables

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language®); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA

Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

1 Scope

Data Distribution Service (DDS) is a family of standards from the Object Management Group (OMG) that provide connectivity, interoperability and portability for Industrial Internet, cyber physical, and mission-critical applications.

Time Sensitive Networking (TSN) is a collection of standards developed by the TSN Task Group of the IEEE 802.1 Work Group. Their purpose is to enable deterministic, highly reliable communication on standard Ethernet. With its support for different types of Quality of Service (QoS), a single TSN network infrastructure can be used to communicate mission critical data with real-time delivery requirements side-by-side with non-critical data.

There are several reasons why DDS and TSN are a good fit. Most fundamentally, both technologies provide one-to-many communications that support different levels of QoS for different streams of data[1]. Consequently, some of the basic DDS concepts have similar counterparts in TSN. For example, DDS revolves around a strongly typed data-centric publish-subscribe model where DataWriters are responsible for updating particular types of data and matching DataReaders observe those updates. This granularity of interest is called a Topic and typical DDS systems consist of one to hundreds of them. The combination of a DataWriter with its Topic name can be seen as an identifier of the source of a DDS data stream. All matching DataReaders are sinks to that same Stream. Similarly, TSN has Talkers that update one or more streams delivering data to the connected Listeners. TSN Streams are identified by their VLAN Tag and destination MAC address.

DDS also has quite an extensive set of QoS policies. Their purpose is to instruct the middleware about, among other things, the importance and urgency of the information in the different DDS data streams. In line with the data-centric philosophy, QoS policies are applied per data stream. Some of those QoS policies have close similarities to the mechanisms that define TSN traffic classes (e.g., the Deadline, LatencyBudget and TransportPriority QoS policies). However, such DDS QoS policies may not be met without a deterministic network infrastructure.

To provide DDS with a deterministic network infrastructure that can guarantee time-critical behavior, this specification defines a set of mechanisms that allow and simplify the deployment of DDS applications over a TSN-enabled network infrastructure. Mapping DDS streams to underlying TSN Streams, system designers can rely on a deterministic data distribution behavior from end-to-end. That is, from the producing application all the way down to the network stack, over the network, and back up to the consuming application.

This specification covers two fundamental aspects of the integration of DDS and TSN:

- Clause 7 provides a comprehensive configuration model for DDS-TSN applications. It extends the standard configuration syntax defined in [DDS-XML] and [DDS-JSON] to specify deployment and TSN-specific settings to configure TSN-enabled equipment to prioritize and schedule time-sensitive DDS traffic.
- Clause 8 defines a set of mechanisms to successfully deploy DDS applications over TSN. That includes rules and considerations to configure DDS applications that need to comply with a TSN configuration, such as the most appropriate QoS policies, data modeling considerations, etc.

Moreover, this specification includes two Annexes that provide additional definitions and examples:

- Annex A (Normative) defines a Platform-Specific Model (PSM) for the DDSI-RTPS wire protocol that allows RTPS Messages (i.e., the messages that encapsulate DDS traffic) to be sent directly over Ethernet frames. This alternative to the DDSI-RTPS UDP/IP PSM is suitable for some scenarios where the IP stack may introduce unnecessary delays.
- Annex B (Informative) includes two examples that show how to design, configure, and deploy two DDS systems over TSN. Each example provides instructions to deploy a set of applications using one of the two DDSI-RTPS PSMs this specification addresses: UDP/IP and Ethernet.

[1] DDS also provides support for many-to-many communications for different streams of data.

2 Conformance

This document contains no independent conformance points.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[802.1AS] IEEE, 802.1AS-2011: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks, 2011

[802.1CB] IEEE, 802.1CB-2017: Frame Replication and Elimination for Reliability, 2017

[802.1Q] IEEE, 802.1Q-2018: Bridges and Bridged Networks, 2018

[802.1Qbv] IEEE, 802.1Qbv-2015: Enhancements for Scheduled Traffic, 2015

[802.1Qcc] IEEE, 802.1Qcc-2018: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, 2018

[802.1Qcr] IEEE, 802.1Qcr-2020: Amendment 34: Asynchronous Traffic Shaping, 2020

[DDS] OMG, Data Distribution Service, Version 1.4, https://www.omg.org/spec/DDS/1.4

[DDS-JSON] OMG, DDS Consolidated JSON Syntax, Version 1.0, https://www.omg.org/spec/DDS-JSON/1.0

[DDS-SECURITY] OMG, DDS Security, Version 1.1, https://www.omg.org/spec/DDS-SECURITY/1.1

[DDS-XML] OMG, DDS Consolidated XML Syntax, Version 1.0, https://www.omg.org/spec/DDS-XML/1.0

[DDS-XTYPES] OMG, Extensible and Dynamic Topic Types for DDS, Version 1.3, https://www.omg.org/spec/DDS-XTypes/1.3

[DDSI-RTPS] OMG, Real-Time Publish-Subscribe Protocol DDS Interoperability Wire Protocol, Version 2.5, https://www.omg.org/spec/DDSI-RTPS/2.5

[IDL] OMG, Interface Definition Language, Version 4.2, https://www.omg.org/spec/IDL/4.2

[RFC2460] IETF, RFC 2460, Internet Protocol, Version 6 (IPv6) Specification, 1998

[RFC2474] IETF, RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, 1998

[RFC3290] IETF, RFC 3290, An Informal Management Model for Diffserv Routers, 2002

[RFC768] IEEE, RFC 768, User Datagram Protocol, 1980

[RFC791] IETF, RFC 791, Internet Protocol—DARPA Internet Program Protocol Specification, 1981

[RFC7950] IETF, RFC 7950, The YANG 1.1 Data Modeling Language, 2016

[RFC8939] IETF, RFC 8939, Deterministic Networking (DetNet) Data Plane: IP, 2020

[RFC9023] IETF, RFC 9023, Deterministic Networking (DetNet) Data Plane: IP over IEEE 802.1 Time-Sensitive Networking (TSN), 2021

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Data Distribution Service

Data Distribution Service (DDS) is a family of standards from the Object Management Group (OMG) that provide connectivity, interoperability and portability for Industrial Internet, cyber-physical, and mission-critical applications. The DDS connectivity standards cover Publish-Subscribe (DDS), Service Invocation (DDS-RPC), Interoperability (DDSI-RTPS), Information Modeling (DDS-XTYPES), Security (DDS-SECURITY), as well as programming APIs for C, C++, Java and other languages.

Platform-Independent Model

Platform-Independent Model (PIM) is an abstract definition of a facility, often expressed with the aid of formal or semi-formal modeling languages such as OMG UML, which does not depend on any particular implementation technology.

Platform-Specific Model

Platform-Specific Model (PSM) is a concrete definition of a facility—typically based on a corresponding PIM—in which all implementation-specific dependencies have been resolved.

Time Sensitive Networking

Time Sensitive Networking (TSN) is a collection of standards developed by the TSN Task Group of the IEEE 802.1 Working Group. Their purpose is to enable deterministic, highly reliable communication on standard Ethernet. With its support for different types of Quality of Service (QoS), a single TSN network infrastructure can be used to communicate mission critical data with real-time delivery requirements side-by-side with non-critical data.

5 Symbols

The acronyms used in this specification are shown in Table 5.1.

Table 5.1: Acronyms

Acronym	Meaning
CNC	Centralized Network Configuration
CUC	Centralized User Configuration
DDS	Data Distribution Service
DSCP	Differentiated Services Code Point
IDL	Interface Definition Language

Acronym	Meaning
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
MAC	Media Access Control
MTU	Maximum Transmission Unit
OMG	Object Management Group
PIM	Platform-Independent Model
PSM	Platform-Specific Model
QoS	Quality of Service
RTPS	Real Time Publish Subscribe
SEDP	Simple Endpoint Discovery Protocol
SPDP	Simple Participant Discovery Protocol
TCP	Transmission Control Protocol
TSN	Time Sensitive Networking
UDP	User Datagram Protocol
UML	Unified Modeling Language
UNI	User/Network Configuration Interface
VLAN	Virtual Local Area Network
XML	Extensible Markup Language
XTYPES	Extensible and Dynamic Topic Types
YANG	Yet Another Next Generation

6 Additional Information

6.1 Changes to Adopted OMG Specifications

This specification does not change any adopted OMG specification.

6.2 Acknowledgments

The following individuals and companies submitted content that was incorporated into this specification:

- Real-Time Innovations, Inc.
- Twin Oaks Computing, Inc.
- ZettaScale Technology Ltd.

Submitting contributors:

- (lead) Fernando Garcia-Aranda—Real-Time Innovations, Inc.
- Reinier Torenbeek—Real-Time Innovations, Inc.
- Clark Tucker—Twin Oaks Computing, Inc.
- Erik Hendriks—ZettaScale Technology Ltd.

7 DDS-TSN System Deployment

7.1 Overview

TSN is a data-link layer technology that relies on the mechanism of Virtual Local Area Networking (VLAN) defined in the IEEE 802.1Q standard [802.1Q]. By adding the optional VLAN Tag to a standard 802.3 Ethernet packet, network equipment, such as switches, can be instructed to follow VLAN procedures. By means of a set of amendments to [802.1Q], TSN adds support for deterministic communications over Ethernet. For example, [802.1Qbv] (now part of [802.1Q]) introduced the concept of a network-wide schedule that allows allocating dedicated time slots for the guaranteed transmission of specific traffic classes—leaving the rest of time slots for the transmission of Ethernet frames of lower priority traffic classes (on a best effort basis). The aforementioned schedule requires tight synchronization of all devices, as described in (a revision to) [802.1AS].

In TSN, time-critical communications are carried from Talkers on the sending side to Listeners on the receiving side. Such information Streams are either one-to-one or one-to-many and may have their own timing and bandwidth requirements.

TSN supports different configuration models, ranging from fully distributed to fully centralized. Such models allow users to allocate resources for the transmission of Streams of data between one Talker application and one or more Listener applications, located in different end stations.

Common to all configuration models is the concept of User/Network Configuration Interface (UNI), which is the mechanism end stations use to provide data requirements from Talkers and Listeners to the network. The IEEE 802.1Qcc standard [802.1Qcc] defines the TSN UNI in a “schema, encoding, or protocol” independent manner, and applies it in the different configuration model as follows:

- In the Fully Distributed model ([802.1Qcc], subclause 46.1.3.1), Talkers and Listeners provide their requirements directly to their closest Bridge using the TSN UNI. Such information is passed through all the Bridges between end stations, which configure themselves to accommodate resources for the requested data requirements.
- The Centralized Network/Distributed User Model ([802.1Qcc], subclause 46.1.3.2) introduces a Centralized Network Configuration (CNC) entity that has a complete view of the network topology and communicates with every Bridge involved in the communication using a network management protocol. End stations communicate their requirements to edge Bridges (i.e., Bridges connected to an end station) using the TSN UNI. Edge Bridges act as proxies, propagating Talker and Listener requirements using the TSN UNI as well.
- The Fully Decentralized Model ([802.1Qcc], subclause 46.1.3.3) introduces a Centralized User Configuration (CUC) entity to collect the requirements of end stations using an end station configuration protocol. The CUC uses the TSN UNI to exchange requirements with the CNC, which like in the case of the Centralized Network/Distributed User Model is responsible for configuring all Bridges using a network management protocol. The end station configuration protocol is out of the scope of [802.1Qcc].

[802.1Q] defines managed objects for the configuration of Bridge features, such as TSN features. However, it does not specify a single data modeling language or network management protocol to configure Bridge managed objects. The IETF provides YANG [RFC7950] to model Bridge managed objects, as well as network management protocols that support YANG, such as NETCONF and RESTCONF. The TSN Task Group provides YANG modules for TSN Bridge features; for example, [802.1Qcc] provides YANG data module definitions for TSN UNI ([802.1Qcc], subclause 46.3).

To accommodate as many use cases as possible, this clause defines a platform-independent DDS-TSN Configuration Model that can be mapped to different implementations of the TSN UNI, as well as to all three configuration models described in [802.1Qcc].

Implementers of this specification may also use the DDS-TSN Configuration Model to build applications or toolchains capable of processing configuration files describing a complete DDS-TSN deployment, generating code, other configuration files, or API calls to configure and deploy a statically-defined TSN system.

7.2 Configuration Model (PIM)

The configuration of a DDS system capable of leveraging a TSN-enabled network needs to address multiple aspects:

- Modeling DDS Applications, including the definition of DDS entities, data types, and QoS policies associated with them (see subclause 7.2.1).
- Modeling the Nodes where DDS Applications may be deployed (subclause 7.2.2).
- Modeling specific deployment scenarios, matching DDS Applications with Deployment Nodes (see subclause 7.2.2), and defining Talker and Listener requirements to allocate resources for Streams of data encapsulating time-sensitive DDS samples (see subclause 7.2.3).

Parts of the configuration model, such as the concept of QoS Libraries or DomainParticipant Libraries, were first introduced in the [DDS-XML] and [DDS-JSON] specifications. These mechanisms can be leveraged by tools capable of deploying a preconfigured DDS system or capable of taking a snapshot of a running DDS system. This subclause extends the existing concepts by providing a complete Platform-Independent Model (PIM) to configure DDS Applications, their deployment topology, and the TSN configuration parameters required by a TSN-enabled system.

7.2.1 DDS Application Configuration

Figure 7.1 defines the entities required to model a standalone DDS system composed of different DDS applications.

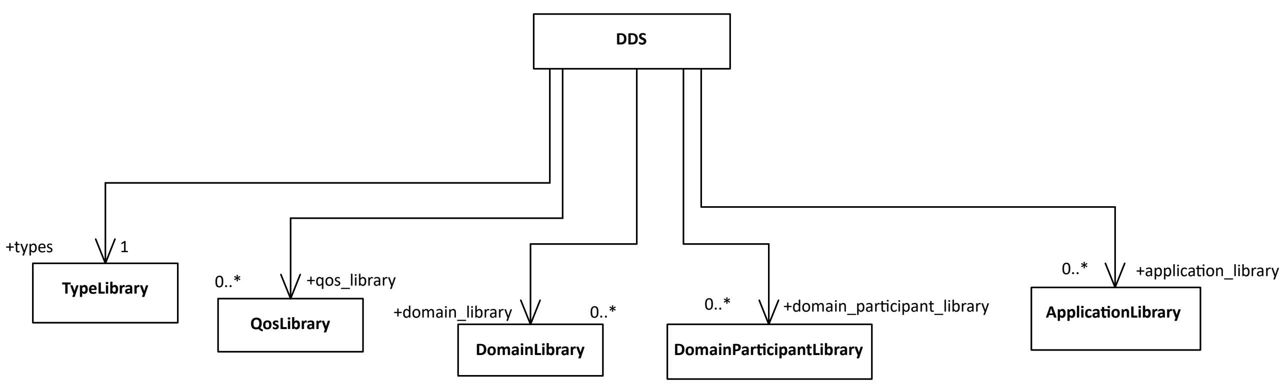

Figure 7.1: DDS Application Configuration Model

In this model, a DDS system is comprised of:

- A Type Library, which contains the definition of all the types available in the DDS system.
- A QoS Library, which organizes QoS profiles associated with different DDS entities.
- A Domain Library, which organizes the entities that model the resources exchanged in a Domain.
- A DomainParticipant Library, which organizes DomainParticipants and their contained DDS entities.
- An Application Library, which organizes DDS Applications that are running or can be deployed on a DDS system.

7.2.1.1 Type Libraries

The set of types that can be configured and represented in a type library (which comprise all the types in the DDS Type System) are defined in subclause 7.2.2 of [DDS-XTYPES].

7.2.1.2 QoS Libraries

Table 7.1 and Table 7.2 provide a formal definition of the classes that model a Qos Library.

Table 7.1: QoSLibrary Definition

Class	`QosLibrary`		
Note	QoS Libraries are named collections of Qos Profiles.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	`String8`	1	Qos Library name.
`qos_profile`	`QosProfile`	0..*	Qos Profiles associated with the Qos Library.

Table 7.2: QoSProfile Definition

Class	`QoSProfile`		
Note	Qos Profiles group a related set of entity QoS Policies.		
Attribute	**Type**	**Multiplicity**	**Note**
`domain_participant_qos`	`DomainParticipantQos`	0..*	Configures the QoS Policies that are applicable to a DomainParticipant. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the `DomainParticipantQos` type. The model allows the definition of different `domain_participant_qos` settings. In that case, the `domain_participant_qos` shall provide a name to identify the specific `DomainParticipantQos` configuration.
`publisher_qos`	`PublisherQos`	0..*	Configures the QoS Policies that are applicable to a Publisher. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the `PublisherQos` type. The model allows the definition of different `publisher_qos` settings. In that case, the `publisher_qos` shall provide a name to identify the specific `PublisherQos` configuration.
`subscriber_qos`	`SubscriberQos`	0..*	Configures the QoS Policies that are applicable to a Subscriber. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the `SubscriberQos` type. The model allows the definition of different `subscriber_qos` settings. In that case, the `subscriber_qos` shall provide a name to identify the specific `SubscriberQos` configuration.

topic_qos	**TopicQos**	0..*	Configures the QoS Policies that are applicable to a Topic. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the **TopicQos** type. The model allows the definition of different **topic_qos** settings. In that case, the **topic_qos** shall provide a name to identify the specific **TopicQos** configuration.
datawriter_qos	**DataWriterQos**	0..*	Configures the QoS Policies that are applicable to a DataWriter. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the **DataWriterQos** type. The model allows the definition of different **datawriter_qos** settings. In that case, the **datawriter_qos** shall provide a name to identify the **DataWriterQos** configuration.
datareader_qos	**DataReaderQos**	0..*	Configures the QoS Policies that are applicable to a DataReader. See [DDS] (subclauses 2.2.5 and 2.3.3) for a formal definition of the **DataReaderQos** type. The model allows the definition of different **datareader_qos** settings. In that case, the **datareader_qos** shall provide a name to identify the **DataReaderQos** configuration.

7.2.1.3 Domain Libraries

Table 7.3 and Table 7.4 provide a formal definition of the classes that model a Domain Library.

Table 7.3: DomainLibrary Definition

Class	**DomainLibrary**		
Note	Domain Libraries are named collections of Domains.		
Attribute	**Type**	**Multiplicity**	**Note**
name	**String8**	1	Domain Library name.
domain	**Domain**	0..*	Domains associated with the Domain Library.

Table 7.4: Domain Definition

Class	**Domain**
Note	A Domain defines the resources that are available within a particular DDS Domain. Such resources include the types that are registered within that Domain, and the Topics that are published and subscribed to. DomainParticipants may refer to a Domain in their type definition. This capability enables them to instantiate the entities that are defined within a Domain, inheriting Topic and Registered Type definitions within the Domain declaration.

Attribute	Type	Multiplicity	Note
name	**String8**	1	Domain Library name.
registered_type	**RegisteredType**	0..*	Defines the types to be registered within the Domain. These types become available to the DomainParticipants that are instantiated according to this Domain definition.
Topic	**Topic**	0..*	Configures the DDS Topics to be instantiated within the DDS Domain. See [DDS] (subclauses 2.2.2.3.1) for a formal definition of the **Topic** type. The configuration model shall represent only the attributes of the Topic type that apply (i.e., **topic_name** and **type_name**). Therefore, the operations provided by the **Topic** type are not part of the configuration model.

Table 7.5 provides a formal definition of the **RegisteredType** class.

Table 7.5: RegisteredType Definition

Class	**RegisteredType**		
Note	A registered type effectively registers a type in the TypeLibrary within a DDS Domain.		
Attribute	**Type**	**Multiplicity**	**Note**
name	**String8**	1	Name under which the original type will be registered within the Domain.
type_ref	**String8**	1	Fully-qualified name (within the context of the TypeLibrary) of the type to be registered.

7.2.1.4 DomainParticipant Libraries

Table 7.6 and Table 7.7 provide a formal definition of the classes that model a DomainParticipant Library.

Table 7.6: DomainParticipantLibrary Definition

Class	**DomainParticipantLibrary**		
Note	DomainParticiant Libraries are named collections of DomainParticipants.		
Attribute	**Type**	**Multiplicity**	**Note**
name	**String8**	1	DomainParticipantLibray name.
domain_participant	**DomainParticipant**	0..*	DomainParticipants that are part of the DomainParticipantLibrary.

Table 7.7: DomainParticipant Definition

Class	`DomainParticipant`		
Note	The **`DomainParticipant`** class describes a DDS DomainParticipant. Its definition is based on that of the DDS DomainParticipant type in [DDS], which specifies a Domain ID and is composed of DDS Publishers, Subscribers, and Topics. The class definition also provides the necessary mechanisms to inherit DomainParticipant configurations from other DomainParticipants through the **`base_name`** attribute, and to inherit the properties defined in a Domain through the **`domain_ref`** attribute.		
Base	**`DDS::DomainParticipant`** (as defined in [DDS], subclause 2.2.2.2.1).		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	**`String8`**	1	DomainParticipant name.
`base_name`	**`String8`**	0..1	Fully-qualified name of the DomainParticipant from which the current DomainParticipant shall inherit its definition.
`domain_ref`	**`String8`**	0..1	Fully-qualified name of the Domain from which the current DomainParticipant shall inherit its definition.

7.2.1.5 Application Libraries

Table 7.8 and Table 7.9 provide a formal definition of the classes that model an Application Library.

Table 7.8: ApplicationLibrary Definition

Class	**`ApplicationLibrary`**		
Note	Application Libraries are named collections of DDS Applications.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	**`String8`**	1	ApplicationLibrary name.
`application`	**`Application`**	0..*	DDS Applications that are part of the ApplicationLibrary.

Table 7.9: Application Definition

Class	**`Application`**		
Note	Applications are collections of DomainParticipants (and their contained entities) in the context of a process. Applications may be associated with a Deployment Node as specified in subclause 7.2.2.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	**`String8`**	1	Application name.

domain_participant	**DomainParticipant**	0..*	DomainParticipants that are part of the Application (see Table 7.7 for the definition of **DomainParticipant**).

7.2.2 Deployment Configuration

The network topology description captures the deployment properties of a DDS system. The Deployment Configuration includes the definition of a set of Deployment Nodes where DDS Applications can run, explicit deployment configurations modeling instantiations of DDS Applications in Deployment Nodes, as well as any requirements these applications may pose on an underlying TSN infrastructure to accommodate resources for time-sensitive information exchange.

Figure 7.2 extends the DDS Application Configuration Model in subclause 7.2.1 to provide a NodeLibrary of Nodes where Applications can run, and a DeploymentLibrary of Deployment configurations.

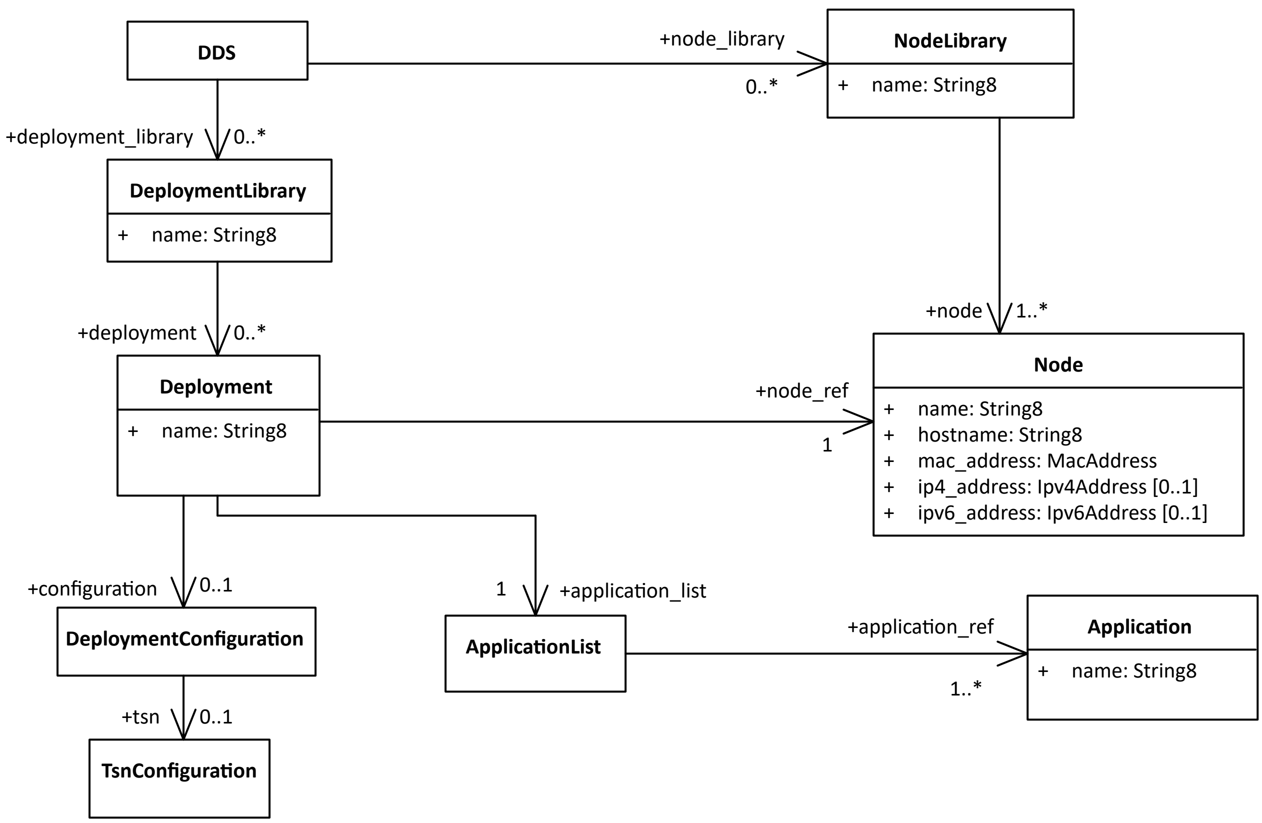

Figure 7.2: Deployment Configuration Model

7.2.2.1 Node Libraries

Table 7.10 and Table 7.11 provide a formal definition of the **NodeLibrary** and **Node** classes.

Table 7.10: NodeLibrary Definition

Class	**NodeLibrary**

Note	NodeLibraries are named collections of deployment Nodes.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	`String8`	1	NodeLibrary name.
`node`	`Node`	0..*	Nodes that are part of the NodeLibrary.

Table 7.11: Node Definition

Class	`Node`		
Note	Nodes define hosts where DDS applications may run. They provide the necessary information to access such nodes, including their Hostname, MAC address, IPv4, and IPv6 address.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	`String8`	1	Name identifying the deployment Node within the NodeLibrary.
`hostname`	`String8`	1	Hostname of the Node.
`mac_address`	`MacAddress`	1	MAC address associated with the network interface to be used for DDS traffic.
`ipv4_address`	`Ipv4Address`	0..1	IPv4 address associated with the network interface to be used for DDS traffic. This field is only required when operating over IPv4 using the DDSI-RTPS UDP/IP PSM (see subclause 8.3).
`ipv6_address`	`Ipv6Address`	0..1	IPv6 address associated with the network interface to be used for DDS traffic. This field is only required when operating over IPv6 using the DDSI-RTPS UDP/IP PSM (see subclause 8.3).

7.2.2.2 DeploymentLibraries

Table 7.12 and Table 7.13 provide a formal definition of the **`DeploymentLibrary`** and **`Deployment`** classes.

Table 7.12: DeploymentLibrary Definition

Class	`DeploymentLibrary`		
Note	DeploymentLibraries are named collections of Deployments.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	`String8`	1	DeploymentLibrary name.

deployment	**Deployment**	0..*	Deployments that are part of the DeploymentLibrary.

Table 7.13: Deployment Definition

Class	**Deployment**		
Note	Deployments determine the Node where a list of DDS Applications will run. They also provide deployment-specific configuration.		
Attribute	**Type**	**Multiplicity**	**Note**
name	**String8**	1	Deployment name.
node_ref	**Node**	1	Reference to the Node associated with the specific deployment definition.
application_list	**Application**	1..*	List of Applications that run in the deployment node.
configuration	**DeploymentConfiguration**	0..1	Configurations specific to the deployment of DDS applications in the deployment Node.

Table 7.14 Provides a formal definition of the **DeploymentConfiguration** class.

Table 7.14: DeploymentConfiguration Definition

Class	**DeploymentConfiguration**		
Note	Configuration specific to the deployment of DDS applications in the deployment Node.		
Attribute	**Type**	**Multiplicity**	**Note**
tsn	**TsnConfiguration**	0..1	TSN-specific deployment configuration, including the requirements from TSN Talkers and Listeners, and their association with DDS DataWriters and DataReaders. For a formal definition of **TsnConfiguration**, see subclause 7.2.3.

7.2.3 TSN Configuration

On top of the necessary deployment information, a comprehensive user configuration model for DDS shall include means to provide information relevant to TSN-enabled systems.

Figure 7.3 describes a TSN Configuration that defines TSN Talkers and Listeners, and associates them with time-sensitive DDS DataWriters and DataReaders. Such configuration provides the underlying TSN-enabled network with information about time-sensitive DDS traffic, including its size and periodicity, along with potential network requirements.

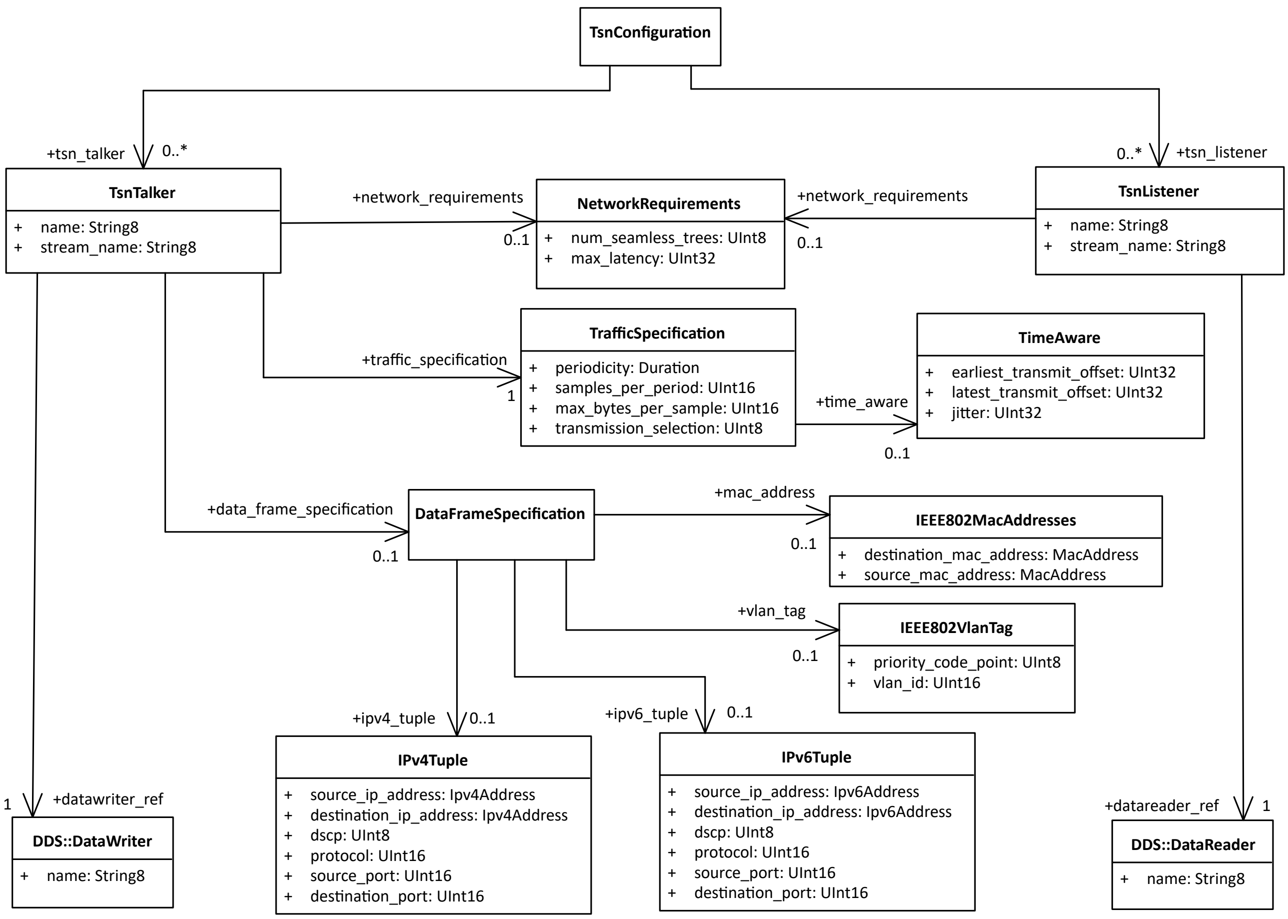

Figure 7.3: TSN Configuration Model

7.2.3.1 TSN Talker

Table 7.15 provides a formal definition of the **`TsnTalker`** class.

Table 7.15: TsnTalker Definition

Class	**`TsnTalker`**

Note	TSN Talkers define the characteristics of the traffic sent by time-sensitive DataWriters, as well as their requirements for the underlying TSN infrastructure. The associated DataWriter is responsible for sending the traffic according to the specified configuration.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	`String8`	1	Name identifying the TSN Talker within the TSN configuration.
`stream_name`	`String8`	1	Name identifying the TSN Stream the Talker is associated with.
`traffic_specification`	`TrafficSpecification`	1	Specifies the way the Talker transmits the information, including information on periodicity, message size, etc.
`network_requirements`	`NetworkRequirements`	0..1	Poses user to network requirements for the underlying TSN infrastructure (e.g., maximum latency and redundant trees). This group is optional.
`data_frame_specification`	`DataFrameSpecification`	0..1	Indicates the mechanism that the network uses to identify the Talker's Stream data. For example, MAC addresses and VLAN Tags, an IPv4 tuple, or an IPv6 tuple. This group is optional, as it may not be required by the TSN configuration model.
`datawriter_ref`	`String8`	1	Name of the DataWriter associated with the TSN Talker.

7.2.3.1.1 TrafficSpecification

Table 7.16 provides a formal definition of the **`TrafficSpecification`** class.

Table 7.16: TrafficSpecification Definition

Class	`TrafficSpecification`		
Note	Specifies the way the TSN Talker transmits information. That is, how often it sends data, how big data samples are, what is the shaper used for the traffic class, etc.		
Attribute	**Type**	**Multiplicity**	**Note**
`periodicity`	`Duration`	1	Cyclic transmission interval of the TSN Talker.
`samples_per_period`	`UInt16`	1	Number of samples the TSN Talker will transmit in every period.
`max_bytes_per_sample`	`UInt16`	1	Maximum size of the DDS sample the TSN Talker shall transmit per period.

transmission_selection	**UInt8**	1	Algorithm the Talker uses to transmit the traffic class. That is, the shaper for the traffic class. Table 8-6 of [802.1Q] and extensions provide the appropriate transmission selection algorithm identifiers. For example: • Strict Priority: 0 • Credit-based Shaper: 1 • Enhanced Transmission Selection (ETS): 2 • Asynchronous Traffic Shaping (ATS) Transmission Selection (defined in [802.1Qcr]): 3
time_aware	**TimeAware**	0..1	Indicates whether the Talker is synchronized to a time in the network. If it is, the configuration includes the offsets at which the Talker may transmit its frames, and the maximum difference in time between those offsets. (See [802.1Qcc], subclause 46.2.3.5 b, and Table 7.17 below.)

Table 7.17 provides a formal definition of the **TimeAware** class.

Table 7.17: TimeAware Definition

Class	**TimeAware**		
Note	Indicates the offsets at which a Talker may transmit its frame within an interval (earliest and latest), as well as the maximum difference in time between those offsets and the synchronized time (jitter). (See [802.1Qcc], subclause 46.2.3.5 b.)		
Attribute	**Type**	**Multiplicity**	**Note**
earliest_transmit_offset	**UInt32**	1	Earliest offset within the interval at which the Talker is capable of starting to transmit. The earliest transmit offset shall be specified in nanoseconds (See [802.1Qcc], subclause 46.2.3.5.5.)
latest_transmit_offset	**UInt32**	1	Latest offset within the interval at which the Talker is capable of starting to transmit. The latest transmit offset shall be specified in nanoseconds. (See [802.1Qcc], subclause 46.2.3.5.6.)
jitter	**UInt32**	1	Maximum difference in time between transmit offsets and the synchronized network time. Jitter shall be specified in nanoseconds (See [802.1Qcc], subclause 46.2.3.5.7.)

7.2.3.1.2 NetworkRequirements

Table 7.18 provides a formal definition of the **NetworkRequirements** class.

Table 7.18: NetworkRequirements Definition

Class	`NetworkRequirements`		
Note	Poses user to network requirements for the underlying TSN infrastructure.		
Attribute	**Type**	**Multiplicity**	**Note**
`num_seamless_trees`	`UInt8`	1	Number of trees that the network will deliver to provide seamless redundancy for the Stream. Zero indicates that no seamless redundancy is required. NOTE—For more information on the behavior of `num_seamless_trees`, see [802.1Qcc] subclause 46.2.3.6.1.
`max_latency`	`UInt32`	1	As defined in [802.1Qcc], it "specifies latency from Talker to Listeners(s) for a single frame of the Stream." It shall be specified in nanoseconds. The requirement shall be satisfied by all Listeners associated with the TSN Stream. NOTE—For more information on the behavior and special values for `max_latency`, see [802.1Qcc] subclause 46.2.3.6.2.

7.2.3.1.3 DataFrameSpecification

Table 7.19 provides a formal definition of the `DataFrameSpecification` class.

Table 7.19: DataFrameSpecification Definition

Class	`DataFrameSpecification`		
Note	Indicates the mechanism that the network uses to identify the Talker's Stream data. For example, MAC addresses and VLAN Tags, an IPv4 tuple, or an IPv6 tuple. • When operating directly over Ethernet, the `DataFrameSpecification` declaration shall define both `mac_address` and `vlan_tag`. • When operating over UDP/IP or TCP/IP, the `DataFrameSpecification` declaration shall include either `ipv4_tuple` (if using IPv4) or `ipv6_tuple` (if using IPv6).		
Attribute	**Type**	**Multiplicity**	**Note**
`mac_addresses`	`IEEE802MacAddresses`	0..1	IEEE 802 MAC addresses for Stream Identification.
`vlan_tag`	`IEEE802VlanTag`	0..1	VLAN Tag (see [802.1Q], Clause 9) for Stream Identification, excluding the Drop Eligible Indicator (DEI) field.
`ipv4_tuple`	`IPv4Tuple`	0..1	Specifies fields in the IPv4 and UDP headers to identify an IPv4 Stream.

ipv6_tuple	**IPv6Tuple**	0..1	Specifies fields in the IPv6 and UDP headers to identify an IPv6 Stream.

Table 7.20 provides a formal definition of the **IEEE802MacAddresses** class.

Table 7.20: IEEE802MacAddresses Definition

Class	**IEEE802MacAddresses**		
Note	IEEE 802 MAC addresses for Stream Identification.		
Attribute	**Type**	**Multiplicity**	**Note**
destination_mac_address	**MacAddress**	1	Destination MAC address. An address with ones in all bits indicates the destination MAC address shall be ignored for Stream Identification.
source_mac_address	**MacAddress**	1	Source MAC address. An address with ones in all bits indicates the source MAC address shall be ignored for Stream Identification.

Table 7.21 provides a formal definition of the **IEEE802VlanTag** class.

Table 7.21: IEEE802VlanTag Definition

Class	**DataFrameSpecification**		
Note	VLAN Tag (see [802.1Q], Clause 9) for Stream Identification, excluding the DEI field.		
Attribute	**Type**	**Multiplicity**	**Note**
priority_code_point	**UInt8**	1	Priority Code Point (PCP) field of the VLAN Tag identifying a traffic class in a Bridge. Value range: [0, 7].
vlan_id	**UInt16**	1	VLAN ID (VID) field of the VLAN Tag. If unknown, **vlan_id** shall be set to 0. Value range: [0, 4095].

Table 7.22 provides a formal definition of the **IPv4Tuple** class.

Table 7.22: IPv4Tuple Definition

Class	**IPv4Tuple**		
Note	Specifies fields in the IPv4 and transport protocol headers to identify an IPv4 Stream.		
Attribute	**Type**	**Multiplicity**	**Note**

source_ip_address	**IPv4Address**	1	Source IPv4 address [RFC791]. A 0.0.0.0 address indicates the source IP address shall be ignored for Stream Identification.
destination_ip_address	**IPv4Address**	1	Destination IPv4 address [RFC791].
dscp	**UInt8**	1	Differentiated Services Code Point (DSCP) field, as defined in [RFC2474]. A **dscp** field of value 64 indicates that the field shall be ignored for Stream Identification.
protocol	**UInt16**	1	The **protocol** field indicates the encapsulated next level protocol. For example, 17 indicates UDP (as specified in [RFC768]).
source_port	**UInt16**	1	Source port of the protocol (e.g., UDP).
destination_port	**UInt16**	1	Destination port of the protocol (e.g., UDP).

Table 7.23 provides a formal definition of the **IPv6Tuple** class.

Table 7.23: IPv6Tuple Definition

Class	**IPv6Tuple**		
Note	Specifies fields in the IPv6 and transport protocol headers to identify an IPv6 Stream.		
Attribute	**Type**	**Multiplicity**	**Note**
source_ip_address	**IPv6Address**	1	Source IPv6 address [RFC2460]. A 0:0:0:0:0:0:0:0 address (the unspecified address), indicates the source IP address shall be ignored for Stream Identification.
destination_ip_address	**IPv6Address**	1	Destination IPv6 address [RFC2460].
dscp	**UInt8**	1	Differentiated Services Code Point (DSCP) field, as defined in [RFC2474]. A **dscp** field of value 64 indicates that the field shall be ignored for Stream Identification.
protocol	**UInt16**	1	The **protocol** field indicates the encapsulated next level protocol. For example, 17 indicates UDP (as specified in [RFC768]).
source_port	**UInt16**	1	Source port of the protocol (e.g., UDP).
destination_port	**UInt16**	1	Destination port of the protocol (e.g., UDP).

7.2.3.2 TSN Listener

Table 7.24 provides a formal definition of the **`TsnListener`** class.

Table 7.24: TsnListener Definition

Class	`TsnListener`		
Note	TSN Listeners define the requirements from time-sensitive DataReaders for the underlying TSN infrastructure.		
Attribute	**Type**	**Multiplicity**	**Note**
`name`	**`String8`**	1	Name identifying the Listener within the **`TsnConfiguration`**.
`stream_name`	**`String8`**	1	Name identifying the TSN Stream the Listener is associated with.
`network_requirements`	**`NetworkRequirements`**	0..1	Poses user to network requirements for the underlying TSN infrastructure (e.g., maximum latency and redundant trees).

7.2.3.2.1 NetworkRequirements

Table 7.25 provides a formal definition of the **`NetworkRequirements`** class.

Table 7.25: NetworkRequirements Definition

Class	`NetworkRequirements`		
Note	Poses user to network requirements for the underlying TSN infrastructure.		
Attribute	**Type**	**Multiplicity**	**Note**
`num_seamless_trees`	**`UInt8`**	1	Number of trees that the network will deliver to provide seamless redundancy for the Stream. Value 0 indicates that no seamless redundancy is required. NOTE—For more information on the behavior of **`num_seamless_trees`**, see [802.1Qcc] subclause 46.2.3.6.1.
`max_latency`	**`String8`**	1	As defined in [802.1Qcc], it "specifies latency from Talker to Listeners(s) for a single frame of the Stream." It shall be specified in nanoseconds. (See [802.1Qcc] subclause 46.2.3.6.2.) The requirement shall be satisfied only by the present Listener. NOTE—For more information on the behavior and special values for **`max_latency`**, see [802.1Qcc] subclause 46.2.3.6.2.

7.3 Configuration Representation (PSM)

This subclause specifies a collection of Platform-Specific Models (PSM) representing the configuration model in different formats.

7.3.1 XML PSM

The syntax to represent the configuration of a DDS system capable of leveraging a TSN-enabled network in XML format is described in the following normative XML schema files:

- *dds-tsn_definitions_nonamespace.xsd* (Normative), contains the XSD type definition for all the types required to represent the configuration model. It defines no **`targetNamespace`**, so that type definitions can be reused by other schemas following the XML Chameleon Schema Definition pattern as described in [DDS-XML].
- *dds-tsn_definitions.xsd* (Normative), defines the XML configuration model and sets **`targetNamespace`** to https://www.omg.org/spec/DDS-TSN.

The syntax defined in these XML files is based on the standard DDS System Building Block Set defined in [DDS-XML] to represent a complete DDS system. In particular, it uses its definitions to model Type Libraries, Qos Libraries, Domain Libraries, DomainParticipant Libraries, and Application Libraries. The schema files add syntax to describe Nodes and Deployment configurations, including the appropriate TSN requirements.

The following non-normative files contain an example that applies the XML schema to represent a DDS system to be deployed on a TSN-enabled network:

- *dds-tsn_configuration_example_system_design.xml* (Informative)
- *dds-tsn_configuration_example_deployment_design.xml* (Informative)

7.3.2 JSON PSM

The syntax to represent the configuration of a DDS system capable of leveraging a TSN-enabled network in JSON format is described in the following normative JSON schema file:

- *dds-tsn_definitions.schema.json* (Normative), which contains JSON type definitions for all the types required to represent the configuration model.

The syntax defined in this JSON schema file is based on the standard DDS System Building Block Set defined in [DDS-JSON] to represent a complete DDS system. In particular, it uses its definitions to model Type Libraries, Qos Libraries, Domain Libraries, Domain Participant Libraries, and Application Libraries. The schema file adds syntax to describe Nodes and Deployment configurations, including the appropriate TSN requirements.

The following non-normative files contain an example that applies the JSON schema to represent a DDS system to be deployed on a TSN-enabled network:

- *dds-tsn_configuration_example_system_design.json* (Informative)
- *dds-tsn_configuration_example_deployment_design.json* (Informative)

7.3.3 YANG PSM

This subclause maps the configuration model to YANG data module definitions for TSN user/network configuration. In particular, it maps the model to the data module definitions to represent Talker, Listener, and Status Groups that are specified in subclause 46.3 of [802.1Qcc]. These definitions can be transformed into other platform-specific formats for existing UNI protocols, such as XML or JSON for RESTCONF.

As mentioned above, the DDS-TSN configuration model provides a Deployment configuration that allows the representation of TSN requirements for a specific deployment scenario. Such requirements define the TSN Talkers and Listeners that are responsible for exchanging time-sensitive DDS data, and refer to the DataWriters and DataReaders that are responsible for the actual information exchange. An application compliant with the mapping rules defined in this subclause should be capable of parsing a DDS-TSN configuration document, identifying the TSN Talkers and Listeners involved in the DDS-TSN system (see subclauses 7.2.2 and 7.2.3), and creating the appropriate requests with Talker and Listener Groups.

NOTE—As specified in subclause 46.2.2 of [802.1Qcc], the configuration of a TSN system based on the fully centralized model can be viewed conceptually as a request/response exchange between a CUC and a CNC, where the CUC transmits a protocol message that contains a Talker or Listener Group (a request), and the CNC sends a protocol message that contains a Status group (a response). Those interactions are modeled according to the specific UNI protocol CUCs and CNCs use to communicate with each other.

7.3.3.1 Talker and Listener Groups

To construct request messages containing Talker and Listener Groups using the appropriate YANG data module definitions, implementers shall identify the DataWriters and DataReaders in the deployment configuration that are associated with a **`TsnTalker`** and a **`TsnListener`** configuration, respectively. The specified configuration, along with the configuration of the **`Application`**s, **`Deployment`**s, and **`Node`**s associated with those DataWriters and DataReaders will be the input for the subsequent transformation of Deployment configurations to platform-independent YANG Talker and Listener Group definitions.

Every TSN Stream provided by a **`TsnTalker`** shall be identified by its Stream ID, represented in YANG with a **`stream-id-type`** string, according to the following transformation rules.

- **`stream-id-type`** (**`typedef`**):
  - The MAC address field—represented by the first six octets of the **`stream-id-type`** string—shall be the MAC address of the **`Node`** associated (via **`node_ref`**) with the **`Deployment`** configuration that instantiates the **`TsnTalker`**.
  - The Unique ID field—represented by the last two octets of the **`stream-id-type`** string—shall be a 16-bit unsigned integer that uniquely identifies the DataWriter within the Deployment node.

Every **`TsnTalker`** in the TSN Deployment configuration shall be mapped to an equivalent Talker Group, represented as a YANG **`group-talker grouping`** according to the following transformation rules:

- **`group-talker`** (**`grouping`**):
  - **`stream-rank`** (**`container`**) shall have a **`rank`** of 1.
  - **`end-station-interfaces`** (**`list`**) shall contain the MAC address of the **`Node`** associated (via **`node_ref`**) with the **`Deployment`** configuration that instantiates the **`TsnTalker`**.
  - **`data-frame-specification`** (**`list`**) is optional. If the **`TsnTalker`**'s **`data_frame_specification`** attribute is not present, the Talker shall not provide **`data-frame-specification`**. If the **`TsnTalker`**'s **`data_frame_specification`** attribute is present, **`data-frame-specification`** shall be configured as follows:
    - If the DataWriter is configured to send data using the DDSI-RTPS Ethernet PSM, **`data-frame-specification`** shall set:
      - **`group-ieee802-mac-addresses`** (**`grouping`**) with the equivalent fields in the **`mac_addresses`** attribute of the **`TsnTalker`**'s **`data_frame_specification`**. That is:
        - **`destination-mac-address`** (**`leaf`**) with the value of **`mac_addresses.destination_mac_address`**.
        - **`source-mac-address`** (**`leaf`**) with the value of **`mac_addresses.source_mac_address`**.

        - **group-ieee802-vlan-tag** (**grouping**) with the equivalent fields in the **vlan_tag** attribute of the **TsnTalker**'s **data_frame_specification**:
            - **priority-code-point** (**leaf**) with the value **vlan_tag.priority_code_point**.
            - **vlan-id** (**leaf**) with the value of **vlan_tag.vlan_id**.
    - If the DataWriter is configured to send data using the DDSI-RTPS UDP/IP PSM over IPv4, **data-frame-specification** shall set:
        - **group-ipv4-tuple** (**grouping**) with the equivalent fields in the **ipv4_tuple** attribute of the **TsnTalker**'s **data_frame_specification**. That is,
            - **source-ip-address** (**leaf**) with the value of **ipv4_tuple.source_ip_address**.
            - **destination-ip-address** (**leaf**) with the value of **ipv4_tuple.destination_ip_address**.
            - **dscp** (**leaf**) with the value of **ipv4_tuple.dscp**.
            - **protocol** (**leaf**) with the value of **ipv4_tuple.protocol**.
            - **source-port** (**leaf**) with the value of **ipv4_tuple.source_port**.
            - **destination-port** (**leaf**) with the value of **ipv4_tuple.destination_port**.
        - **group-ipv6-tuple** (**grouping**) with the equivalent fields in the **ipv6_tuple** attribute of the **TsnTalker**'s **data_frame_specification**. That is:
            - **source-ip-address** (**leaf**) with the value of **ipv6_tuple.source_ip_address**.
            - **destination-ip-address** (**leaf**) with the value of **ipv6_tuple.destination_ip_address**.
            - **dscp** (**leaf**) with the value of **ipv6_tuple.dscp**.
            - **protocol** (**leaf**) with the value of **ipv6_tuple.protocol**.
            - **source-port** (**leaf**) with the value of **ipv6_tuple.source_port**.
            - **destination-port** (**leaf**) with the value of **ipv6_tuple.destination_port**.
- **traffic-specification** (**container**) shall be configured with the equivalent fields in the **TsnTalker**'s **traffic_specification** attribute. The rules to configure equivalent fields are the following:
    - **interval** (**container**) shall be set to the value of **traffic_specification.periodicity**, expressed in terms of a **numerator** and a **denominator** (fractions of a second).
    - **max-frames-per-interval** (**leaf**) shall be set to the value of **traffic_specification.samples_per_period**.
    - **max-frame-size** (**leaf**) shall be set to the value of **traffic_specification.max_bytes_per_sample**.
    - **transmission-selection** (**leaf**) shall be set to the value of **traffic_specification.transmission_selection**.
    - **time-aware** (container) shall be configured as follows:
        - If **traffic_specification.time_aware** field is unspecified in the **TsnTalker**, the following leaf members shall be set to zero: **earliest-transmit-offset**, **latest-transmit-offset**, and **jitter**.
        - Otherwise, **earliest-transmit-offset**, **latest-transmit-offset**, and **jitter** shall be set to the value of the corresponding field within **traffic_specification.time_aware**. That is, **earliest_transmit_offset**, **latest_transmit_offset**, and **jitter**, respectively.

    - **user-to-network-requirements** (**container**) is optional. If the **TsnTalker**'s **network_requirements** attribute is not present, the Talker shall not provide **user-to-network-requirements**. If the **TsnTalker**'s **network_requirements** attribute is present, **user-to-network-requirements** shall be set with equivalent fields in the **TsnTalker**'s **network_requirements**:
        - **num-seamless-trees** (**leaf**) shall be set to the value of **network_requirements.num_seamless_trees**.
        - **max-latency** (**leaf**) shall be set to the value of **network_requirements.max_latency**.
    - **interface-capabilities** (**container**) shall be configured as follows:
        - **vlan-tag-capable** (**leaf**) shall be set according to the capabilities of the underlying infrastructure, as specified in subclause 46.3.1 of [802.1Qcc].
        - **cb-stream-iden-type-list** (**leaf-list**) and **cb-sequence-type-list** (**leaf-list**) shall be configured according to the [802.1CB] support of the underlying infrastructure, as specified in subclause 46.3.1 of [802.1Qcc].

Every **TsnListener** in the TSN Deployment configuration shall be mapped to an equivalent Listener Group, represented as a YANG **group-listener grouping**, according to the following transformation rules:

- **group-listener** (**grouping**):
    - **end-station-interfaces** (**list**) shall contain the MAC address of the **Node** associated (via **node_ref**) with the **Deployment** configuration that instantiates the **TsnListener**.
    - **user-to-network-requirements** (**container**) is optional. If the **TsnListener**'s **network_requirements** attribute is not present, the Listener shall not provide **user-to-network-requirements**. If the **TsnListener**'s **network_requirements** attribute is present, **user-to-network-requirements** shall be configured as follows:
        - **num-seamless-trees** (**leaf**) shall be set to the value of **network_requirements.num_seamless_trees**.
        - **max-latency** (**leaf**) shall be set to the value of **network_requirements.max_latency**.
    - **interface-capabilities** (**container**) shall be configured as follows:
        - **vlan-tag-capable** (**leaf**) shall be set according to the capabilities of the underlying infrastructure, as specified in subclause 46.3.1 of [802.1Qcc].
        - **cb-stream-iden-type-list** (**leaf-list**) and **cb-sequence-type-list** (**leaf-list**) shall be configured according to the [802.1CB] support of the underlying infrastructure, as specified in subclause 46.3.1 of [802.1Qcc].

7.3.3.2 Reception of Status Group

Upon the reception of a request message containing one or more Talker and Listener Groups, a CNC will respond with a protocol message including a **group-status-stream grouping**:

- **group-status-stream** (**grouping**):
    - **status-info** (**container**) with the status for every stream configuration in the network, including the status of Talkers and Listeners, and a failure code indicating if the Stream encountered a failure.
    - **failed-interfaces** (**list**) with the MAC address and interface name of any interface that may have failed within the physical topology.

The response message may also include a **group-status-talker-listener grouping**, which provides the status for a specific Talker or Listener:

- **`group-status-talker-listener`** (**`grouping`**):
  - **`accumulated-latency`** (**`leaf`**) with the worst-case latency in nanoseconds that a Stream frame will encounter along its path. When delivered to a Talker, **`accumulated-latency`** provides the worst-case latency for all Listeners. In contrast, when delivered to a Listener, **`accumulated-latency`** provides the worst-case latency for that specific Listener.
  - **`interface-configuration`** (**`container`**) provides the appropriate configuration for the interfaces specified for the Talker or Listener in the **`end-station-interfaces group`**. Therefore, the list of interface configuration values will include zero or more configurations for **`ieee802-mac-addresses`**, **`ieee802-vlan-tag`**, **`ieee802-vlan-tag`**, or **`ipv6-tuple`**. The returned configuration is specific to each Talker and Listener of the associated Stream.

8 DDSI-RTPS Wire Protocol over TSN

8.1 Overview

The DDS Interoperability Real-Time Publish-Subscribe wire protocol (DDSI-RTPS) is responsible for delivering DDS user and discovery data from publishing to subscribing applications. DDSI-RTPS follows a model-driven design, where a PIM defines the structure and behavior of the RTPS Messages that construct the wire protocol, such that they can be mapped to different transport protocols or lower layer protocols in specific PSMs. DDSI-RTPS poses little requirements on the underlying technology. Indeed, it is designed to work on top of transport protocols that are neither connection oriented nor reliable, such as UDP (see UDP/IP PSM in [DDSI-RTPS]).

In the context of time-sensitive streams of data, information exchanged between Talkers and Listeners may be sent according to a schedule that requires the definition of a period and a maximum message size (see subclause 7.2.3.1). The global schedule is guaranteed by the underlying network infrastructure and the configuration of the TSN system. These characteristics determine the type of DDS information that Talkers and Listeners can exchange using time-critical Streams, as well as the type of reliability that DDS applications can expect from the network infrastructure.

The purpose of this clause is to define the rules, mechanisms, and behavior of DDS systems configured to operate over TSN. In this sense, it defines: the subset of RTPS Messages and Submessages that Endpoints may exchange to achieve a deterministic behavior, considerations and requirements for discovery and user traffic, a set of QoS Policies that DDS Endpoints may configure to operate in a deterministic manner, and other requirements and considerations related to DDS Security and data modeling.

8.2 DDSI-RTPS PIM over TSN

8.2.1 Message Module

The message module is the part of the DDSI-RTPS PIM that defines the types and structure of an RTPS Messages. All RTPS Messages consist of a Header followed by a series of Submessages. The number of Submessages encapsulated in an RTPS Message is limited by the maximum message size supported by the underlying transport mechanism.

As described in subclause 8.3.7 of [DDSI-RTPS], RTPS Submessages are divided in two groups: Entity Submessages and Interpreter Submessages. Entity Submessages target an RTPS Entity. In contrast, Interpreter Submessages modify the RTPS Receiver state and provide a context to process Entity Messages.

To provide a deterministic behavior and a deterministic message size, DataWriters associated with an **`TsnTalker`** in the Deployment configuration (see subclause 7.2.3.1) may need to limit their RTPS Message exchange to RTPS Messages that include the following RTPS Submessages:

- InfoTimestamp
- Data
- DataFrag

The size of the corresponding RTPS Header, plus the size of InfoTimestamp Submessages and subsequent Data or DataFrag Submessages that follow it in the RTPS Message need to be accounted for in the configuration of the **`max_bytes_per_sample`** field in Table 7.16.

RTPS Submessages responsible for achieving reliability or in-order delivery, such as Gap, AckNack, NackFrag; as well as the rest of Interpreter Submessages, may be sent as part of “Best Effort” Streams. However, given the guarantees of the underlying TSN system, this sort of traffic may be unnecessary for TSN-enabled DataReaders and DataWriters. Also, retransmissions and other types of aperiodic traffic may fail to meet the schedule and configuration of the TSN Streams associated with the delivery of time-critical data.

8.2.2 Discovery Module

The Discovery Module defines the discovery protocols that allow DomainParticipants to discover other DomainParticipants and their corresponding Endpoints. As a result of this process, Endpoints that have discovered matching counterparts can establish communication.

8.2.2.1 Performing Discovery over TSN

The standard Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP) (which are defined in subclauses 8.5.3 and 8.5.4 of [DDSI-RTPS]) are not considered time-critical. While DomainParticiants send periodic announcements, the process by which DomainParticiants exchange information about their Endpoints may not be easily scheduled. Therefore, discovery may need to be performed either:

- Over non-critical channels using non-critical streams, ensuring that it is performed before the TSN scenario is properly configured (and must adhere to a schedule);
- or preconfigured statically, as specified in subclause 8.5.6 of [DDSI-RTPS].

8.2.2.2 Restricting Discovery for Time-Sensitive Applications

Unless otherwise specified, SPDP and SEDP may match TSN-enabled DDS applications with regular DDS applications that do not meet the specified TSN requirements. In other words, DataWriters acting as Talkers of a TSN Stream may discover (and send data to) compatible DataReaders that are not Listeners of that TSN Stream, and DataReaders acting as Listeners of a TSN Stream may discover (and receive data from) compatible DataWriters that are not acting as the Talker of that TSN Stream. The reason is that the matching rules for DataReaders and DataWriter are based on Topic names, associated types, and QoS policies—they are unaware of the underlying TSN requirements. As a consequence, a DataReader expecting data at a certain rate or within certain latency boundaries may end up receiving data from a matching DataWriter that does not adhere to that predefined configuration.

Implementers of this specification may apply different techniques to restrict discovery of applications that do not provide time-sensitive requirements. For instance, the DataWriter and DataReaders associated with the Talker and Listeners of a TSN Stream, respectively, could use the PARTITION QoS ([DDS], subclause 2.2.3.13) to prevent them from matching compatible DataReaders and DataWriters that do not belong to that TSN Stream. For that purpose, implementers may use the **`stream_name`** in the **`TsnTalker`** and **`TsnListener`** configuration class (see subclause 7.2.3) as the partition name. With such configuration, the SEDP will not match DataReaders and DataWriters that are not reading and writing in that specific partition, guaranteeing that the DataWriter acting as the Talker of a TSN Stream will only communicate with DataReaders acting as Listeners of that TSN Stream.

8.2.3 QoS Policies

To guarantee low latency and provide a deterministic behavior, users of this specification shall take into account the exchange of meta traffic, such as acknowledgments and potential repairs, exchanged between DataReaders and DataWriters that may prevent a publishing application from adhering to a predefined schedule, or to a predetermined message size. That behavior can be guaranteed using the appropriate QoS Policy settings.

For instance, DataReaders and DataWriters associated with a TSN Stream that have time-critical requirements disallowing message repairs and support for retransmission of information to late-joining applications, may be configured according to the following QoS Policies:

- RELIABILITY QoS: BEST_EFFORT
- DURABILITY QoS: VOLATILE
- HISTORY QoS: KEEP_LAST with depth of 1

Such configuration disables the need for sending repairs and acknowledgments and guarantees that data will only be delivered to DataReaders that have already been discovered, with a predictable message size.

8.2.4 Other Considerations

8.2.4.1 Data Modeling Considerations

As mentioned above, DDS applications using certain TSN capabilities may need to adhere to a predefined schedule, as well as to predictable message sizes and publication rates. In that sense, there are two data modeling capabilities that users of this specification need to account for: the use of unbounded types and the number of instances per Topic.

8.2.4.1.1 Use of Unbounded Types

The DDS Type System defined in [DDS-XTYPES] supports types, such as **`Sequence`**, **`String8`**, **`String16`**, and **`Map`**, for which the bound parameter may be omitted. Such collections are considered unbounded and their size might be variable throughout the lifetime of the applications that exchange them.

Users of this specification dealing with TSN deployments with requirements for a predictable message size will therefore need to ensure that **`Sequence`**s, **`String`**s, and **`Map`**s are either bounded or that their upper bound does not increase beyond the predetermined message size.

8.2.4.1.2 Number of Instances per Topic

Likewise, implementers of time-sensitive DDS applications need to consider the number of instances per Topic and the rate at which they are updated. Adhering to a predefined schedule requires taking into account the messages sent every period. In the case of keyed Topics, that implies accounting for the number, size, and update life cycle of all the instances that are handled by a DataWriter associated with a TSN Talker.

In certain use cases, where instances of a Topic have different life cycles, implementers may need to use a separate DataWriter for each specific instance, associating each instance-specific DataWriter to a separate TSN Stream.

8.2.4.2 DDS Security Considerations

The DDS Security specification [DDS-SECURITY] defines a security model and a service plugin interface architecture compliant with DDS and its DDSI-RTPS wire protocol. To accomplish that, it extends the data types used by DDS discovery and defines new built-in discovery Topics to enable authentication and access control of DDS applications, and implements mechanisms to secure DDS messages on the wire.

Applications using DDS Security systems often require authentication of the DomainParticipants that are discovered within a DDS Domain. They may also restrict access to certain Topics, and validate and enforce the permissions of discovered applications (e.g., if a discovered DomainParticipant can create DataReaders or DataWriters to read or write certain Topics). The information exchange between the built-in Endpoints of the DomainParticipants to perform such operations have the same behavior and requirements as regular built-in Endpoints for DDS discovery. Therefore, the same considerations for discovery traffic, specified in subclause 8.2.2.1, apply to the traffic exchange of the built-in Endpoints defined in [DDS-SECURITY].

Moreover, DDS Security introduces mechanisms to secure RTPS Messages on the wire. That implies adding secure Submessage elements, which increases the size of protected messages. Therefore, implementers of this specification need to take into account the increase in message size introduced by DDS Security to protect (sign or encrypt) user data, setting the **`max_bytes_per_sample`** field in the **`TsnTalker`** definition (see Table 7.16) to an appropriate value.

8.3 DDSI-RTPS UDP/IP PSM over TSN

As specified in [802.1Qcc]: “the goal of TSN configuration is to allow Talkers and Listeners to use their existing transport layer and application layer protocols for data, rather than requiring a TSN-specific frame format.” That implies supporting the use of “well-established frame formats, such as TCP, UDP and IEEE 802.1 (MAC addresses and VLAN identifier)” over TSN. For that purpose, [802.1Qcc] introduces the concept of Stream Transformation, which: “provides features to enable the transformation of the stream's identification at the user/network boundary, either within an end station or at the nearest Bridge."

In the case of the DDSI-RTPS UDP/IP PSM, the implementation of the Stream Identification function needs to pinpoint RTPS Messages containing data samples associated with TSN Streams. In other words, data samples that a DataWriter associated with a TSN Talker sends to one or more DataReaders associated with TSN Listeners.

8.3.1 Stream Identification of UDP Datagrams Encapsulating RTPS Messages

To provide the Stream Identification function with sufficient information, this specification recommends the use of the 6-tuple, as defined in [RFC3290] (also adopted in RFCs related to DetNet, such as [RFC9023] and [RFC8939]), to uniquely identify the RTPS Messages from information available in the following six fields from the IP header and UDP headers: destination address, source address, IP protocol, source port, destination port, and differentiated services code point (DSCP).

Implementers of this specification may compute the TSN Streams that are part of the DDS-TSN configuration and determine a combination of fields from the 6-tuple that uniquely identifies the UDP datagrams encapsulating RTPS Messages associated with a TSN Stream. For example, the users may:

- Configure the DataWriter associated with a TSN Stream to use a specific source port and source address to uniquely identify all the RTPS Messages it sends to all matching DataReaders.
- Configure all DataReaders associated with a TSN Stream to listen on a specific destination multicast address and port to uniquely identify all the RTPS Messages that are sent to those DataReaders.
- Configure a combination of source addresses and ports, or destination addresses and ports, with a DSCP value to uniquely identify RTPS Messages.

The specific mechanism to configure the source port or source IP address of a DataWriter, the destination port or destination address of a DataReader, and the value of DSCP in the IP header is out of the scope of this specification. The mechanism to configure the end station or nearest Bridge to perform the Stream Identification function is also out of the scope of this specification. It may be derived from the **`DataFrameSpecification`** setting specified in subclause 7.2.3.1.3, or computed by an external entity, such as a CNC or a system integrator. In either case, the resulting 6-tuple shall provide a combination of fields that enables unique identification of UDP datagrams encapsulating the RTPS Messages that belong to a TSN Stream, differentiating them from other UDP datagrams.

8.3.2 Stream Transformation of UDP Datagrams Encapsulating RTPS Messages

UDP datagrams matching the identification criteria for a TSN Stream shall be treated specially when encapsulated over Ethernet frames. In other words, the Stream Transformation function either at the end station or at the nearest Bridge shall use the VLAN Tag and group destination MAC address that have been preestablished to identify the specific TSN Stream. For example, implementers may apply the IEEE 802.1CB function for Active Destination MAC and VLAN Stream Identification (see [802.1CB], subclause 6.6), which assigns the VLAN Tag and group destination MAC address, in combination with the IP Stream Identification function (see [802.1CB], subclause 6.7). As indicated in [802.1Qcc], these functions may be implemented both in software and hardware.

8.4 DDSI-RTPS Ethernet PSM over TSN

When operating directly over Ethernet, implementers may also need to apply a Stream Transformation function, either at the end station or at the nearest Bridge, capable of setting the VLAN Tag and group destination address assigned to a TSN Stream. This is due to the fact that DDS applications may discover real multicast addresses and may be unaware of the VLAN Tags and group destination addresses the network uses to identify a TSN Stream. In such cases, implementers of this specification may use the IEEE 802.1CB function for Active Destination MAC and VLAN Stream Identification function (see [802.1CB]) to translate the VLAN Tag and group destination address pair to the appropriate value for Stream Identification.

Annex A: DDSI-RTPS Ethernet PSM

(normative)

A.1 Introduction

This Platform Specific Model (PSM) maps the DDSI-RTPS Wire Protocol PIM defined in [DDSI-RTPS] to Ethernet. The goal for this PSM is to provide a mapping with minimal overhead directly on top of Ethernet, without the IP and UDP headers that are part of the existing UDP/IP PSM.

A.2 Notational Conventions

This PSM uses the same notational contentions defined in subclause 9.2 of [DDSI-RTPS] for the UDP/IP PSM. In particular:

- It defines all data types under the **`RTPS`** namespace.
- It uses OMG IDL [IDL] for definition of types.
- It uses CDR for wire representation.

A.3 Mapping of the RTPS Types

The mapping of RTPS types is the same as the mapping defined in subclause 9.3 of [DDSI-RTPS] for the UDP/IP PSM, except those noted in Table A.1.

Table A.1: PSM mapping of the value types that appear on the wire

Type	Description of the PSM Mapping

`Locator_t`	Mapping of **`Locator_t`** is the same as the mapping defined by the UDP/IP PSM in subclause 9.3.2 of [DDSI-RTPS]. That is:
	`struct Locator_t {`
	`long locatorKind;`
	`unsigned long port;`
	`octet[16] address;`
	`};`
	This PSM adds the **`LOCATOR_KIND_ETHERNET`** to the list of values reserved by the DDSI-RTPS protocol in subclause 8.2.1.2 of [DDSI-RTPS]. **`LOCATOR_KIND_ETHERNET`** shall be defined as:
	`const long LOCATOR_KIND_ETHERNET = 0x02000000;`
	If the **`Locator_t`** kind is **`LOCATOR_KIND_ETHERNET`**, the **`port`** encodes the Ethernet VLAN ID (VID), the Ethernet Priority Code Point (PCP), and the RTPS logical port. In this case, the leading 12 bits contain the VLAN ID, followed by 4 bits containing the Priority Code Point (4 bits). The last 2 bytes contain the RTPS logical port.
	If the **`Locator_t`** kind is **`LOCATOR_KIND_ETHERNET`**, the **`address`** contains the corresponding host MAC address. In this case, the leading 10 octets of the address shall be zero. The last 6 octets are used to store the MAC address. The mapping between the colon-notation "AA:BB:CC:DD:EE:FF" of a MAC address and its representation in the **`address`** field of a **`Locator_t`** is:
	`address = {0,0,0,0,0,0,0,0,0,0,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF}`

A.4 Mapping of the RTPS Messages

A.4.1 Overall Structure

The RTPS PIM defines the overall structure of a Message, which is composed of a Header and a set of Submessages.

This PSM follows the structure defined by the RTPS UDP/IP PSM in subclause 9.4.1 of [DDSI-RTPS], which aligns each Submessage on a 32-bit boundary with respect to the start of the Message.

```
Locator_t:
0...2...........8...............16.............24...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header                                                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Submessage                                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Submessage                                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A Message has a well-known length. This length is not sent explicitly by the DDSI-RTPS protocol but is part of the underlying mechanism with which Messages are sent. In the case of Ethernet, the length of the Message is the length of the Ethernet frame's payload (i.e., the Ethernet MTU).

A.4.2 Mapping of PIM SubmessageElements

This PSM preserves the IDL type and on-the-wire representation defined by the RTPS UDP/IP PSM in subclause 9.4.2 of [DDSI-RTPS] for all SubmessageElements except for LocatorList, which shall be represented as defined below.

A.4.2.1 LocatorList

The PSM mapping for the `LocatorList` SubmessageElement is the same as that defined in subclause 9.4.2.10 of [DDSI-RTPS].

Each `Locator_t` with `kind = LOCATOR_KIND_ETHERNET` has the following wire representation:

```
0...2...........8...............16.............24...............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              locatorKind = LOCATOR_KIND_ETHERNET                |
+-----------------------+-------+---------------+-----------------+
|      VID (12 bits)    |  PCP  |      Logical Port (2 bytes)     |
+---------------+-------+-------+---------------+-----------------+
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|                                 |
+-------------------------------+      MAC Address (6 bytes)      +
|                                                                 |
+---------------+---------------+---------------+-----------------+
```

A.4.3 Additional SubmessageElements

This specification does not introduce any additional SubmessageElements to those defined by the RTPS PIM in subclause 8.3.5 of [DDSI-RTPS].

A.4.4 Mapping of RTPS Header

The mapping of the RTPS Header shall follow the mapping defined by the RTPS UDP/IP PSM in subclause 9.4.4 of [DDSI-RTPS].

A.4.5 Mapping of RTPS Submessages

The mapping of the RTPS Header shall follow the mapping defined by the RTPS UDP/IP PSM in subclause 9.4.5 of [DDSI-RTPS] for the RTPS Submessages defined in the RTPS PIM.

Platform-specific Submessages that apply only to the UDP/IP PSM, such as **`InfoReplyIp4`**, shall not be mapped.

A.5 RTPS Message Encapsulation

When RTPS operates over Ethernet, a Message is the contents (payload) of exactly one Ethernet frame.

NOTE—Ethernet frames containing an RTPS Message should be set with an appropriate EtherType indicating the protocol encapsulated in their payload. Future versions of this specification may mandate the use of an EtherType registered with the IEEE Registration Authority to identify Ethernet frames encapsulating RTPS Messages.

A.6 Mapping of the RTPS Protocol

A.6.1 Default Locators

A.6.1.1 Discovery traffic

Discovery traffic is the traffic generated by the Participant and Endpoint Discovery Protocols. For the Simple Discovery Protocols (SPDP and SEDP), discovery traffic is the traffic exchanged between the built-in Endpoints.

The SPDP built-in Endpoints are configured using well-known logical ports (see subclause 8.5.3.4 of [DDSI-RTPS]). The Ethernet PSM shall map those well-known ports to the logical port numbers, using the number expressions defined in Table 9.8 of [DDSI-RTPS].

The default logical ports used by the SEDP built-in Endpoints match those used by the SPDP. If a node chooses not to use the default logical ports for the SEDP, it can include the new logical port numbers as part of the information exchanged during the SPDP.

A.6.1.2 User Traffic

User traffic is the traffic exchanged between user-defined Endpoints (i.e., non built-in Endpoints). As such, it pertains to all the traffic not related to discovery. By default, user-defined Endpoints shall use the port number expressions listed in Table 9.9 of [DDSI-RTPS] to derive the corresponding logical port.

User-defined Endpoints may choose to not use the default ports. In this case, remote Endpoints obtain the port number as part of the information exchanged during the SEDP.

A.6.1.3 Default Logical Port Numbers

The default logical port numbers are the same as those defined for the UDP/IP PSM in subclause 9.6.1.3 of [DDSI-RTPS].

A.6.1.4 Default Settings for the Simple Participant Discovery Protocol

A.6.1.4.1 Default multicast address

In order to enable plug-and-play interoperability, the default pre-configured list of locators must include the following multicast locator:

```
DefaultMulticastLocator = {
    LOCATOR_KIND_ETHERNET, // locatorKind
    PB + DG * domainId,    // port
    01:00:5E::EF:FF:00:01  // address
```

```
}
```

All Participants must announce and listen on this multicast address.

```
SPDPbuiltinParticipantWriter.readerLocators CONTAINS DefaultMulticastLocator
SPDPbuiltinParticipantReader.multicastLocatorList CONTAINS DefaultMulticastLocator
```

A.6.1.4.2 Default Announcement rate

The default rate by which SPDP periodic announcements are sent shall be the default rate defined for the UDP/IP PSM in subclause 9.6.1.4.2 of [DDSI-RTPS].

A.6.2 Data Representation for the Built-in Endpoints

The IDL and wire representation of the built-in Endpoints are the same as defined in subclause 9.6.2 of the UDP/IP PSM [DDSI-RTPS].

A.6.3 ParameterId Definitions used to Represent In-line QoS

ParameterIds shall be defined as defined by the RTPS UDP/IP PSM in subclause 9.6.3 of [DDSI-RTPS].

Annex B: DDS-TSN Integration Examples

(informative)

B.1 Overview

This Annex provides two examples that illustrate how to design and deploy a DDS System over TSN. The first example, defined in subclause B.2, uses the DDSI-RTPS UDP/IP PSM for communication. The second example, defined in subclause B.3, uses the DDSI-RTPS Ethernet PSM specified in Annex A.

Both examples comprise five DDS applications that run on five hosts equipped time-sensitive network interfaces:

- **`Application_1`** runs on **`Host_1`**, and publishes a Topic called **`Square`**.
- **`Application_2`** runs on **`Host_2`**, and publishes a Topic called **`Triangle`**.
- **`Application_3`** runs on **`Host_3`** and subscribes to the **`Square`** Topic.
- **`Application_4`** runs on **`Host_4`** and subscribes to the **`Triangle`** Topic.
- **`Application_5`** runs on **`Host_5`** and subscribes to both **`Square`** and **`Triangle`**[2].

The overall configuration of DDS applications, deployment nodes, and TSN endpoints for these examples—which may be used to deploy and configure all the necessary elements (e.g., processing the files to perform remote configuration via a CNC, or with static toolchains)—is available in XML and JSON format. To facilitate the separation of DDS System Design and Deployment Design, the example provides separate design and deployment configuration documents:

- The DDS System Design document defines the DDS applications that are part of the system, along with the DomainParticipants they instantiate, and their contained entities (i.e., Topics, Publishers, Subscribers, DataWriters, and DataReaders), data type declarations, and QoS Libraries.
- The Deployment Design document defines the Hosts where applications may be run, and the Deployment configurations that determine which applications run on which hosts, and deployment-specific requirements, such as TSN configurations and assignments of TSN Talkers to DataWriters and TSN Listeners to DataReaders, respectively.

The following documents combine the configuration for both examples:

- XML format:
  - *dds-tsn_configuration_example_system_design.xml* (Informative)
  - *dds-tsn_configuration_example_deployment_design.xml* (Informative)
- JSON format:
  - *dds-tsn_configuration_example_system_design.json* (Informative)
  - *dds-tsn_configuration_example_deployment_design.json* (Informative)

The scenario assumes the presence of an entity capable of reading these configuration files and deploying **`Application_1`**, **`Application_2`**, **`Application_3`**, **`Application_4`**, and **`Application_5`** on **`Host_1`**, **`Host_2`**, **`Host_3`**, **`Host_4`**, and **`Host_5`**, respectively. Such an entity is also capable of interpreting the TSN requirements expressed in the **`TsnTalker`** and **`TsnListener`** classes of the Deployment configurations, which are associated with each of the time-critical DataWriters and DataWriters, respectively. Lastly, the scenario assumes the entity is capable of communicating with a CNC, a toolchain, or any other entity capable of configuring the underlying network.

2 Using two separate DataReaders.

B.1.1 Deployment Configurations

Each example is composed of five different deployment configurations that determine the DDS applications to be run on each of the five available hosts. As mentioned above, deployment configurations identify the TSN Talkers and Listeners that provide the time requirements from DDS entities, and determine the mapping of DataWriters and DataReaders to TSN Streams. Deployment configurations are organized in DeploymentLibraries. Each example uses a separate DeploymentLibrary, as they contain Deployment configurations specific to the DDSI-RTPS PSM they use (i.e., UDP/IP or Ethernet). These DeploymentLibraries are named **`MyDeploymentLibraryUdp`** and **`MyDeploymentLibraryEthernet`**, respectively. Upon the successful deployment of the DDS time-critical applications, the deployment configurations result in the configuration of two TSN Streams: **`SquareStream`** and **`TriangleStream`**.

B.1.2 Configuration Models

Depending on the configuration model, the TSN configuration information needs to be propagated through each Bridge (fully distributed model) or communicated to a CNC (directly, in the centralized network/distributed user model; or through a CUC, in the fully centralized model). Alternatively, users of this specification may perform the configuration manually, using different kinds of interfaces to configure the network and the hosts where applications run. This example assumes the use of the protocol integration described in subclause 46.2.2 of [802.1Qcc], but these steps can be extrapolated to a manual configuration process.

The TSN user/network configuration must account for three high-level groups of configuration information: Talker, Listener, and Status. The configuration protocol can be seen as a request/response exchange, where an end station or CUC transmits a request message with a Talker or Listener Group, and a Bridge or a CNC responds with a Status Group. Operations on the Talker and Listener Groups allow: (1) joining to a Stream to configure and allocate resources for the Stream to flow from a Talker to one or more Listeners, and (2) leaving a Stream to release resources.

In centralized configuration models, the CNC is responsible for discovering the underlying physical topology, including end stations and Bridges, and reading the capabilities of each Bridge using remote management protocols. Upon the reception of join requests including configurations of Streams with Talker and Listener Groups, the CNC configures the corresponding TSN features for Streams in the path from Talker to Listener. The CNC returns the status of each Stream. In contrast, in the fully distributed model, the Status response is delivered in a message merged with the Listener requests, which are propagated through Bridges to the Talker (see [802.1Qcc], subclause 46.2.2). Manual configurations need to calculate the schedule and propagate the configuration to both Bridges and end stations.

NOTE—The examples follow a custom notation in JSON format to describe Talker, Listener, and Status Groups, which instantiates the YANG data model defined in subclause 46.3 of [802.1Qcc] (adding some of the extra fields required to perform requests, such as **`stream-id`**). Users of this specification may translate these Talker, Listener, and Status Groups to YANG-based protocols or apply them in a manual configuration scheme.

B.2 DDS-TSN Deployment Scenario Using DDSI-RTPS UDP/IP PSM

This example applies the deployment scenario defined in B.1 to a set of DDS applications that use the DDSI-RTPS UDP/IP PSM over TSN.

B.2.1 Stream Configuration

This example is based on five deployment configurations that define where each application runs. Deployment configurations for this example are grouped in a DeploymentLibrary named **`MyDeploymentLibraryUdp`**. As derived from the configuration (looking at the TSN Configuration part for each Deployment, which provides the list of Talkers and Listeners), applications exchange two TSN Streams:

- **`SquareStream`**, which is associated with:

- **SquareWriterTalker_1**—Associated with **SquareWriter_1** (whose fully qualified name is **MyApplicationLibrary::Application_1::DomainParticipant_1::Publisher_1::SquareWriter_1**), which is instantiated by **DomainParticipant_1**, part of **Application_1** running on **Host_1**. (See **MyDeploymentLibraryUdp::Deployment_Host_1** configuration in the XML or JSON system design documents.)
- **SquareReaderListener_3**—Associated with **SquareReader_3** (whose fully qualified name is **MyApplicationLibrary::Application_3::DomainParticipant_3::Subscriber_3::SquareReader_3**), which is instantiated by **DomainParticipant_3**, part of **Application_3** running on **Host_3**. (See **MyDeploymentLibraryUdp::Deployment_Host_3** configuration in the XML or JSON system design documents.)
- **SquareReaderListener_5**—Associated with **SquareReader_5** (whose fully qualified name is **MyApplicationLibrary::Application_5::DomainParticipant_5::Subscriber_5::SquareReader_5**), which is instantiated by **DomainParticipant_5**, part of **Application_5** running on **Host_5**. (See **MyDeploymentLibraryUdp::Deployment_Host_5** configuration in the XML or JSON system design documents.)

- **TriangleStream**, which is associated with:
  - **TriangleWriterTalker_2**—Associated with **TriangleWriter_2** (whose fully qualified name is **MyApplicationLibrary::Application_2::DomainParticipant_2::Publisher_2::TriangleWriter_2**), which is instantiated by **DomainParticipant_2**, part of **Application_2** running on **Host_2**. (See **MyDeploymentLibraryUdp::Deployment_Host_2** configuration in the XML or JSON system design documents.)
  - **TriangleReaderListener_4**—Associated with **TriangleReader_4** (whose fully qualified name is **MyApplicationLibrary::Application_4::DomainParticipant_4::Subscriber_4::TriangleReader_4**), which is instantiated by **DomainParticipant_4**, part of **Application_4** running on **Host_4**. (See **MyDeploymentLibraryUdp::Deployment_Host_4** configuration in the XML or JSON system design documents.)
  - **TriangleReaderListener_5**—Associated with **TriangleReader_5** (its fully qualified name is **MyApplicationLibrary::Application_5::DomainParticipant_5::Subscriber_5::TriangleReader_5**), which is instantiated by **DomainParticipant_5**, part of **Application_5** running on **Host_5**. (See **MyDeploymentLibraryUdp::Deployment_Host_5** configuration in the XML or JSON system design documents.)

With the above information, the requirements for each Stream can be sent through join requests with a Talker and Listener Group for every Talker and Listener. To define each individual Talker and Listener Group, the example follows the mapping rules defined in subclause Error: Reference source not found (which defines the mapping rules using the YANG data modeling syntax).

B.2.1.1 Square Stream Configuration

To configure the system, the first step is to provide the requirements for **SquareStream** sending join requests with the Talker Group for **SquareWriterTalker_1**, and the Listener Groups for **SquareReaderListener_3** and **SquareReaderListener_5**.

The Talker Group for **SquareWriterTalker_1** takes as an input the **MyDeploymentLibraryUdp::Deployment_Host_1** deployment configuration:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "stream-rank": {
        "rank": 1
```

```
    },
    "end-station-interfaces": [
        {
            "mac-address": "AA-AA-AA-AA-AA-AA"
        }
    ],
    "data-frame-specification": [
        {
            "ipv4-tuple": {
                "source-ip-address": "0.0.0.0",
                "destination-ip-address": "239.255.255.1",
                "dscp": 0,
                "protocol": 17,
                "source-port": 0,
                "destination-Port": 7421
            }
        }
    ],
    "traffic-specification": {
        "interval": {
            "numerator": 2,
            "denominator": 1000
        },
        "max-frames-per-interval": 1,
        "max-frame-size": 1000,
        "transmission-selection": 0,
        "time-aware": {
            "earliest-transmit-offset": 0,
            "latest-transmit-offset": 2000000,
            "jitter": 5000
        }
    },
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

NOTE—`DataFrameSpecification` provides information to perform Stream Transformation. As specified in [802.1Qcc], if the end station is responsible for performing Stream Transformation, the Talker group shall not include the `DataFrameSpecification`, because the network will only need to use the destination MAC address and VLAN ID to identify a Stream. In contrast, if the nearest Bridge is responsible for performing Stream Transformation, the appropriate `DataFrameSpecification` must be provided to the CNC, so it can configure IEEE 802.1CB functions in the nearest Bridge to identify the Stream and perform the appropriate transformations. This example includes a `DataFrameSpecification` to show how it is defined, and to provide information for those performing manual configurations, which may require configuring hosts or Bridges to identify Streams based on a combination of the IPv4 6-tuple.

The Listener Group for `SquareReaderListener_3` takes as an input the `MyDeploymentLibraryUdp::Deployment_Host_3` deployment configuration:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "end-station-interfaces": [
        {
```

```
            "mac-address": "CC-CC-CC-CC-CC-CC"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```


The Listener Group for `SquareReaderListener_5` takes as an input the `MyDeploymentLibraryUdp::Deployment_Host_5` deployment configuration:


```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "EE-EE-EE-EE-EE-EE"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```


The schedule for `SquareStream` resulting from a Status response is shown below:


```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "status-info": {
        "talker-status": "ready",
        "listener-status": "ready",
        "failure-code": 0
    },
    "accumulated-latency": 150000,
    "interface-configuration": {
        "interface-list": [
            {
                "ieee802-mac-addresses": {
                    "destination-mac-address": "EE-DD-CC-BB-AA-00",
                    "source-mac-address": "AA-AA-AA-AA-AA-AA"
                },
                "ieee802-vlan-tag": {
                    "priority-code-point": 3,
                    "vlan-id": 4500
                },
                "ipv4-tuple": {
                    "source-ip-address": "0.0.0.0",
                    "destination-ip-address": "239.255.255.1",
                    "dscp": 0,
                    "protocol": 17,
                    "source-port": 0,
                    "destination-port": 7421
```

```
                },
                "time-aware-offset": 25000
            }
        ]
    }
}
```

B.2.1.2 Triangle Stream Configuration

The next step is to provide the requirements for **TriangleStream** sending join requests with the Talker Group for **TriangleWriterTalker_2**, and the Listener Groups for **TriangleReaderListener_4** and **TriangleReaderListener_5**.

The Talker Group for **TriangleWriterTalker_2** takes as an input the **MyDeploymentLibraryUdp::Deployment_Host_2** deployment configuration:


```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "stream-rank": {
        "rank": 1
    },
    "end-station-interfaces": [
        {
            "mac-address": "BB-BB-BB-BB-BB-BB"
        }
    ],
    "data-frame-specification": [
        {
            "ipv4-tuple": {
                "source-ip-address": "0.0.0.0",
                "destination-ip-address": "239.255.255.2",
                "dscp": 0,
                "protocol": 17,
                "source-port": 0,
                "destination-Port": 7422
            }
        }
    ],
    "traffic-specification": {
        "interval": {
            "numerator": 2,
            "denominator": 1000
        },
        "max-frames-per-interval": 1,
        "max-frame-size": 1000,
        "transmission-selection": 0,
        "time-aware": {
            "earliest-transmit-offset": 0,
            "latest-transmit-offset": 2000000,
            "jitter": 5000
        }
    },
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
```

```
}
```

The Listener Group for **`TriangleReaderListener_4`** takes as an input the **`MyDeploymentLibraryUdp::Deployment_Host_4`** deployment configuration:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "DD-DD-DD-DD-DD-DD"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The Listener Group for **`TriangleReaderListener_5`** takes as an input the **`MyDeploymentLibraryUdp::Deployment_Host_5`** deployment configuration:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "EE-EE-EE-EE-EE-EE"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The schedule for **`TriangleStream`** resulting from a Status response is shown below:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "status-info": {
        "talker-status": "ready",
        "listener-status": "ready",
        "failure-code": 0
    },
    "accumulated-latency": 150000,
    "interface-configuration": {
        "interface-list": [
            {
                "ieee802-mac-addresses": {
                    "destination-mac-address": "EE-DD-CC-BB-AA-01",
                    "source-mac-address": "BB-BB-BB-BB-BB-BB"
                },
                "ieee802-vlan-tag": {
                    "priority-code-point": 3,
                    "vlan-id": 4500
```

```
            },
            "ipv4-tuple": {
                "source-ip-address": "0.0.0.0",
                "destination-ip-address": "239.255.255.2",
                "dscp": 0,
                "protocol": 17,
                "source-port": 0,
                "destination-port": 7422
            },
            "time-aware-offset": 25000
        }
    ]
  }
}
```


B.2.2 Host Configuration

After calculating the Stream configuration, each host needs to prepare for the execution of the different applications. In a fully centralized configuration model that would be the responsibility of the CUC. In other models, either applications or a system integrator would be responsible for such configuration.

If the host needs to be configured to perform Stream Transformation, the corresponding IEEE 802.1CB functions for Stream Identification shall be configured to perform the following transformations:

- In **`Host_1`**, replace the DestinationMacAddress and VlanTag fields of Ethernet frames encapsulating IP packets matching the **`interface-configuration.interface-list[0].ipv4-tuple`** in the Status response for **`SquareStream`** with the **`interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address`** and **`interface-configuration.interface-list[0].ieee802-vlan-tag`** specified in that same Status response.
- In **`Host_2`**, replace the DestinationMacAddress and VlanTag fields of Ethernet frames encapsulating IP packets matching the **`interface-configuration.interface-list[0].ipv4-tuple`** in the Status response for **`TriangleStream`** with the **`interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address`** and **`interface-configuration.interface-list[0].ieee802-vlan-tag`** specified in that same Status response.
- In **`Host_3`**, restore the DestinationMacAddress and VlanTag fields of Ethernet frames encapsulating IP packets matching the **`interface-configuration.interface-list[0].ip4-tuple`** in the Status response for **`SquareStream`** to their original value.
- In **`Host_4`**, restore the DestinationMacAddress and VlanTag fields of Ethernet frames encapsulating IP packets matching the **`interface-configuration.interface-list[0].ip4-tuple`** in the Status response for **`TriangleStream`** to their original value.
- In **`Host_5`**, restore the DestinationMacAddress and VlanTag fields of Ethernet frames encapsulating IP packets matching the **`interface-configuration.interface-list[0].ip4-tuple`** in the Status responses for **`SquareStream`** and **`TriangleStream`** to their original value.

If Stream Transformation is performed in the nearest Bridge, then such reconfiguration is unnecessary (the **`interface-configuration`** group will not be part of the Status group).

It is worth noting that, on top of Stream Transformation configuration, the host shall be configured so that each data sample is sent according to the **`time-aware-offset`** indicated in **`interface-configuration.interface-list[0]`** included in the Status response for either Stream.

B.2.3 DDS Application Configuration and Schedule Execution

Once the hosts are configured, everything is ready to begin the publication cycle. It is worth noting that DDS Applications may need to be adjusted to comply with the Stream schedule. This implies:

- Adjusting the QoS Policies to the suggestions in subclause 8.2.3.

- Using a dedicated PARTITION QoS to the **`stream_name`** to restrict DataReaders and DataWriters from discovering counterparts unrelated to the TSN Stream (if such restriction is required).
- Configuring the interfaces through which time-sensitive traffic must be sent. If Stream Transformation needs to be performed, the application shall also configure the fields of the IPv4 tuple specified by the **`interface-configuration`** of the Status response for **`SquareStream`** and **`TriangleStream`** accordingly (e.g., to use specific source and destination ports, and source and destination IP addresses).

As prescribed by the configuration, Talkers execute in intervals of 2ms. That implies, processing information in the application, and invoking the DataWriter's **`write()`** operation every 2ms. The underlying DDS applications copy, serialize the data input, and send it over the appropriate network interface using the DDSI-RTPS UDP/IP PSM implementation. The selected interval must be enough to accommodate for the execution of all the steps listed above within the specified period. The mechanisms to calculate serialization and execution times are out of the scope of this specification.

B.3 DDS-TSN Deployment Scenario Using DDSI-RTPS Ethernet PSM

This example applies the deployment scenario defined in B.1 to a set of DDS applications that use the DDSI-RTPS Ethernet PSM (defined in Annex A) over TSN.

B.3.1 Stream Configuration

This example is based on five deployment configurations that define where each application runs. Deployment configurations for this example are grouped in a DeploymentLibrary named **`MyDeploymentLibraryEthernet`**. As mentioned in subclause B.1.2, looking at the TSN configuration section in each Deployment that contains the list of TSN Talkers and Listeners and their configuration, two TSN Streams can be identified:

- **`SquareStream`**, which is associated with:
    - **`SquareWriterTalker_1`**—Associated with **`SquareWriter_1`** (whose fully qualified name is **`MyApplicationLibrary::Application_1::DomainParticipant_1::Publisher_1::SquareWriter_1`**), which is instantiated by **`DomainParticipant_1`**, part of **`Application_1`** running on **`Host_1`**. (See **`MyDeploymentLibraryEthernet::Deployment_Host_1`** configuration in the XML or JSON system design documents.)
    - **`SquareReaderListener_3`**—Associated with **`SquareReader_3`** (whose fully qualified name is **`MyApplicationLibrary::Application_3::DomainParticipant_3::Subscriber_3::SquareReader_3`**), which is instantiated by **`DomainParticipant_3`**, part of **`Application_3`** running on **`Host_3`**. (See **`MyDeploymentLibraryEthernet::Deployment_Host_3`** configuration in the XML or JSON system design documents.)
    - **`SquareReaderListener_5`**—Associated with **`SquareReader_5`** (whose fully qualified name is **`MyApplicationLibrary::Application_5::DomainParticipant_5::Subscriber_5::SquareReader_5`**), which is instantiated by **`DomainParticipant_5`**, part of **`Application_5`** running on **`Host_5`**. (See **`MyDeploymentLibraryEthernet::Deployment_Host_5`** configuration in the XML or JSON system design documents.)
- **`TriangleStream`**, which is associated with:
    - **`TriangleWriterTalker_2`**—Associated with **`TriangleWriter_2`** (whose fully qualified name is **`MyApplicationLibrary::Application_2::DomainParticipant_2::Publisher_2::TriangleWriter_2`**), which is instantiated by **`DomainParticipant_2`**, part of **`Application_2`** running on **`Host_2`**. (See **`MyDeploymentLibraryEthernet::Deployment_Host_2`** configuration in the XML or JSON system design documents.)

- ◦ **TriangleReaderListener_4**—Associated with **TriangleReader_4** (whose fully qualified name is **MyApplicationLibrary::Application_4::DomainParticipant_4::Subscriber_4::TriangleReader_4**), which is instantiated by **DomainParticipant_4**, part of **Application_4** running on **Host_4**. (See **MyDeploymentLibraryEthernet::Deployment_Host_4** configuration in the XML or JSON system design documents.)
- ◦ **TriangleReaderListener_5**—Associated with **TriangleReader_5** (its fully qualified name is **MyApplicationLibrary::Application_5::DomainParticipant_5::Subscriber_5::TriangleReader_5**), which is instantiated by **DomainParticipant_5**, part of **Application_5** running on **Host_5**. (See **MyDeploymentLibraryEthernet::Deployment_Host_5** configuration in the XML or JSON system design documents.)

With he above information, the requirements for each Stream can be provided sending join requests with a Talker and Listener Group for every Talker and Listener. To define each individual Talker and Listener Group, the example follows the mapping rules defined in subclause Error: Reference source not found (which defines the mapping rules using the YANG data modeling syntax).

B.3.1.1 Square Stream Configuration

To configure the system, the first step is to provide the requirements for **SquareStream** sending join requests with the Talker Group for **SquareWriterTalker_1**, and the Listener Groups for **SquareReaderListener_3** and **SquareReaderListener_5**.

The Talker Group for **SquareWriterTalker_1** takes as an input the **MyDeploymentLibraryEthernet::Deployment_Host_1** configuration:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "stream-rank": {
        "rank": 1
    },
    "end-station-interfaces": [
        {
            "mac-address": "AA-AA-AA-AA-AA-AA"
        }
    ],
    "data-frame-specification": [
        {
            "ieee802-mac-addresses": {
                "destination-mac-address": "FF-FF-FF-FF-FF-FF",
                "source-mac-address": "AA-AA-AA-AA-AA-AA",
            },
            "ieee802-vlan-tag": {
                "priority-code-point": 3,
                "vlan-id": 2500
            }
        }
    ],
    "traffic-specification": {
        "interval": {
            "numerator": 2,
            "denominator": 1000
        },
        "max-frames-per-interval": 1,
        "max-frame-size": 1000,
        "transmission-selection": 0,
        "time-aware": {
            "earliest-transmit-offset": 0,
            "latest-transmit-offset": 2000000,
```

```
            "jitter": 5000
        }
    },
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

NOTE—**DataFrameSpecification** provides information to perform Stream Transformation. As specified in [802.1Qcc], if the end station is responsible for performing Stream Transformation, the Talker group shall not include the **DataFrameSpecification**, because the network will only need to use the destination MAC address and VLAN ID to identify a Stream. In contrast, if the nearest Bridge is responsible for performing Stream Transformation, the appropriate **DataFrameSpecification** must be provided to the CNC, so it can configure IEEE 802.1CB functions in the nearest Bridge to identify the Stream and perform the appropriate transformations. This example, includes a **DataFrameSpecification** to show how it would be defined, and to provide information for those performing manual configurations, which may require configuring hosts or Bridges to identify Streams based on a combination of the IPv4 6-tuple.

The Listener Group for **SquareReaderListener_3** takes as an input the **MyDeploymentLibraryEthernet::Deployment_Host_3** configuration:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "CC-CC-CC-CC-CC-CC"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The Listener Group for **SquareReaderListener_5** takes as an input the **MyDeploymentLibraryEthernet::Deployment_Host_5** configuration:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "EE-EE-EE-EE-EE-EE"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
```

```
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The schedule for **SquareStream** resulting from a Status response is shown below:

```
{
    "stream-id": "AA-AA-AA-AA-AA-AA-00-01",
    "status-info": {
        "talker-status": "ready",
        "listener-status": "ready",
        "failure-code": 0
    },
    "accumulated-latency": 150000,
    "interface-configuration": {
        "interface-list": [
            {
                "ieee802-mac-addresses": {
                    "destination-mac-address": "EE-DD-CC-BB-AA-00",
                    "source-mac-address": "AA-AA-AA-AA-AA-AA"
                },
                "ieee802-vlan-tag": {
                    "priority-code-point": 3,
                    "vlan-id": 4500
                },
                "time-aware-offset": 25000
            }
        ]
    }
}
```

B.3.1.2 Triangle Stream Configuration

The next step is to provide the requirements for **TriangleStream** sending join requests with the Talker Group for **TriangleWriterTalker_2**, and the Listener Groups for **TriangleReaderListener_4** and **TriangleReaderListener_5**.

The Talker Group for **TriangleWriterTalker_2** takes as an input the **MyDeploymentLibraryEthernet::Deployment_Host_2** configuration:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "stream-rank": {
        "rank": 1
    },
    "end-station-interfaces": [
        {
            "mac-address": "BB-BB-BB-BB-BB-BB"
        }
    ],
    "data-frame-specification": [
        {
            "ieee802-mac-addresses": {
                "destination-mac-address": "FF-FF-FF-FF-FF-FF",
                "source-mac-address": "BB-BB-BB-BB-BB-BB",
            },
            "ieee802-vlan-tag": {
                "priority-code-point": 3,
                "vlan-id": 2500
            }
        }
```

```
    ],
    "traffic-specification": {
        "interval": {
            "numerator": 2,
            "denominator": 1000
        },
        "max-frames-per-interval": 1,
        "max-frame-size": 1000,
        "transmission-selection": 0,
        "time-aware": {
            "earliest-transmit-offset": 0,
            "latest-transmit-offset": 2000000,
            "jitter": 5000
        }
    },
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The Listener Group for **`TriangleReaderListener_4`** takes as an input the **`MyDeploymentLibraryEthernet::Deployment_Host_4`** configuration:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "DD-DD-DD-DD-DD-DD"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The Listener Group for **`TriangleReaderListener_5`** takes as an input the **`MyDeploymentLibraryEthernet::Deployment_Host_5`** configuration:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "end-station-interfaces": [
        {
            "mac-address": "EE-EE-EE-EE-EE-EE"
        }
    ],
    "user-to-network-requirements": {
        "num-seamless-trees": 1,
        "max-latency": 2000000
    },
    "interface-capabilities": {
        "vlan-tag-capable": true,
```

```
        "cb-stream-iden-type-list": []
        "cb-sequence-type-list": []
    }
}
```

The schedule for **TriangleStream** resulting from a Status response is shown below:

```
{
    "stream-id": "BB-BB-BB-BB-BB-BB-00-01",
    "status-info": {
        "talker-status": "ready",
        "listener-status": "ready",
        "failure-code": 0
    },
    "accumulated-latency": 150000,
    "interface-configuration": {
        "interface-list": [
            {
                "ieee802-mac-addresses": {
                    "destination-mac-address": "EE-DD-CC-BB-AA-01",
                    "source-mac-address": "BB-BB-BB-BB-BB-BB"
                },
                "ieee802-vlan-tag": {
                    "priority-code-point": 3,
                    "vlan-id": 4500
                },
                "time-aware-offset": 25000
            }
        ]
    }
}
```

B.3.2 Host Configuration

After calculating the Stream configuration, each host needs to be prepared for the execution of the different applications. In a fully centralized configuration model that would be the responsibility of the CUC. In other models, either applications or a system integrator would be responsible for such configuration.

If the host needs to be configured to perform Stream Transformation, the corresponding IEEE 802.1CB functions for Stream Identification must be configured to perform the following transformations:

- In **Host_1**, replace the DestinationMacAddress and VlanTag fields of Ethernet frames matching the **ieee802-mac-addresses** and **ieee802-vlan-tag** in **SquareWriterTalker_1** with the value of **interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address** and **interface-configuration.interface-list[0].ieee802-vlan-tag** in the Status response for **SquareStream**.
- In **Host_2**, replace the DestinationMacAddress and VlanTag fields of Ethernet frames matching the **ieee802-mac-addresses** and **ieee802-vlan-tag** in **TriangleWriterTalker_2** with the value of **interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address** and **interface-configuration.interface-list[0].ieee802-vlan-tag** in the Status response for **TriangleStream**..
- In **Host_3**, restore the MacAddresses and VlanTag fields of Ethernet frames matching the **interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address** and **interface-configuration.interface-list[0].ieee802-vlan-tag** in the Status response for **SquareStream** to their original value.
- In **Host_4**, restore the MacAddresses and VlanTag fields of Ethernet frames matching the **interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address** and **interface-configuration.interface-list[0].ieee802-vlan-tag** in the Status response for **TriangleStream** to their original value.

- In **`Host_5`**, restore the MacAddresses and VlanTag fields of Ethernet frames matching the **`interface-configuration.interface-list[0].ieee802-mac-addresses.destination-mac-address`** and **`interface-configuration.interface-list[0].ieee802-vlan-tag`** in the Status response for **`SquareStream`** and **`TriangleStream`** to their original value.

If Stream Transformation is performed in the nearest Bridge, then such reconfiguration is unnecessary (the **`interface-configuration`** group will not be part of the Status group).

It is worth noting, that on top of Stream Transformation configuration, the host shall be configured so that each data sample is sent according to the **`time-aware-offset`** indicated in **`interface-configuration.interface-list[0]`**.

B.3.3 DDS Application Configuration and Schedule Execution

Once the hosts are configured, everything is ready to begin the publication cycle. It is worth noting that DDS Applications may need to be adjusted to comply with the Stream schedule by:

- Adjusting entity QoS Policies to the suggestions in subclause 8.2.3.
- Using a dedicated PARTITION QoS to the **`stream_name`** to restrict DataReaders and DataWriters from discovering counterparts unrelated to the TSN Stream (if such restriction is required).
- Configuring the interfaces through which time-sensitive traffic must be sent.

As prescribed by the configuration, Talkers execute in intervals of 2ms. That implies, processing information in the application, and invoking the DataWriter's **`write()`** operation every 2ms. The underlying DDS applications copy, serialize the data input, and send it over the appropriate network interface using the DDSI-RTPS Ethernet PSM implementation. The selected interval must be enough to accommodate for the execution of all the steps listed above within the specified period. The mechanisms to calculate serialization and execution times are out of the scope of this specification.