Document Title	Specification of DDS Service Communication Protocol
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1110

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R24-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Contents

1 Introduction and overview

This protocol specification details the OMG® DDS® data types, QoS policies, sequences and semantics of the AUTOSAR Protocol for Service-Oriented Communication over DDS.

AUTOSAR platforms employ, among others, the Service-Oriented Architecture (SOA) communications paradigm. In SOA systems, `Service Interface`s cohesively grouping Elements of various kinds are provided, required or both as `Service Instance`s by Applications.

In this context, OMG® DDS®, as enabled by the Classic Platform Dds Basic Software Module and the Adaptive Platform DDS Network Binding, can be used to exercise Service-Oriented Communication between Applications providing (offering) `Service Instance`s and Applications requiring (consuming) them.

1.1 Protocol purpose and objectives

This protocol defines how the OMG® DDS® middleware standard can be employed to exercise Service-Oriented Communication between Applications providing (offering) `Service Instance`s and Applications requiring (consuming) them.

1.2 Applicability of the protocol

This protocol applies to DDS Service-Oriented Communication in:

- The AUTOSAR Classic Platform
- The AUTOSAR Adaptive Platform
- Non-AUTOSAR platforms targeting AUTOSAR interoperability

1.2.1 Constraints and assumptions

The following constraints and assumptions apply to the present document:

- The OMG® DDS® family of standards already define wire protocols, data type description formats, QoS policies and APIs
- Conversely, in this document "protocols" are described in terms of OMG® DDS® API interactions, QoS policies configuration and data type definitions involved in `Service Interface` advertisement and discovery processes

1.2.2 Limitations

Not applicable.

1.3 Dependencies

1.3.1 Dependencies to other protocol layers

The protocols described in this document rely, indirectly, upon the following OMG® DDS® protocol standards:

- DDS Wire Interoperability protocol (DDSI-RTPS) defined in [1]
- DDS-XTYPES Minimal Programming Interface and Network Interoperability Profiles defined in [2]

1.3.2 Dependencies to other standards and norms

The protocols described in this document target the following OMG® DDS® standards and profiles:

- DDS Minimum Profile defined in [3]
- DDS Wire Interoperability protocol (DDSI-RTPS) defined in [1]
- DDS-XTYPES Minimal Programming Interface and Network Interoperability Profiles defined in [2]

1.3.3 Dependencies to the Application Layer

Not applicable.

2 Use Cases

ID	*Name*	*Description*
UC_001	Provide event-based communication	An Application offers, and publishes samples of, an Event of a Service Interface to other Applications in the network.
UC_002	Require event-based communication	An Application subscribes to, and receives samples of, an Event of a Service Interface, possibly offered by other Applications in the network.
UC_003	Provide trigger-based communication	An Application offers, and publishes samples of, a Trigger of a Service Interface to other Applications in the network.
UC_004	Require trigger-based communication	An Application subscribes to, and receives samples of, a Trigger of a Service Interface, possibly offered by other Applications in the network.
UC_005	Provide method-based communication	An Application offers, by publishing/receiving samples of, a Method of a Service Interface to other Applications in the network.
UC_006	Require method-based communication	An Application calls, by publishing/receiving samples of, a Method of a Service Interface, possibly offered by other Applications in the network.
UC_007	Provide field-based communication	An Application offers, by publishing/receiving samples of, a Field of a Service Interface to other Applications in the network.
UC_008	Require field-based communication	An Application calls/subscribes to, by publishing/receiving samples of, a Field of a Service Interface, possibly offered by other Applications in the network.

3 Related documentation

3.1 Input documents & related standards and norms

[1] DDS Interoperability Wire Protocol, Version 2.2
http://www.omg.org/spec/DDSI-RTPS/2.2

[2] Extensible and Dynamic Topic Types for DDS, Version 1.2
https://www.omg.org/spec/DDS-XTypes/1.2

[3] Data Distribution Service (DDS), Version 1.4
http://www.omg.org/spec/DDS/1.4

[4] RPC over DDS, Version 1.0
https://www.omg.org/spec/DDS-RPC/1.0

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

[6] E2E Protocol Specification
AUTOSAR_FO_PRS_E2EProtocol

3.2 Related specification

Not applicable.

4 Protocol Requirements

Implementation of this protocol requires an OMG® DDS® middleware implementation supporting:

- DDS Minimum Profile defined in [3]
- DDS Wire Interoperability protocol (DDSI-RTPS) defined in [1]
- DDS-XTYPES Minimal Programming Interface and Network Interoperability Profiles defined in [2]

4.1 Requirements Traceability

Requirement	Description	Satisfied by
[FO_RS_Dds_00001]	DDS Compliance	[FO_PRS_DDS_00100] [FO_PRS_DDS_00101] [FO_PRS_DDS_00102] [FO_PRS_DDS_00103] [FO_PRS_DDS_00104] [FO_PRS_DDS_00105] [FO_PRS_DDS_00106] [FO_PRS_DDS_00107] [FO_PRS_DDS_00108] [FO_PRS_DDS_00109] [FO_PRS_DDS_00110] [FO_PRS_DDS_00111] [FO_PRS_DDS_00112] [FO_PRS_DDS_00113] [FO_PRS_DDS_00200] [FO_PRS_DDS_00201] [FO_PRS_DDS_00202] [FO_PRS_DDS_00203] [FO_PRS_DDS_00204] [FO_PRS_DDS_00205] [FO_PRS_DDS_00206] [FO_PRS_DDS_00207] [FO_PRS_DDS_00208] [FO_PRS_DDS_00209] [FO_PRS_DDS_00210] [FO_PRS_DDS_00211] [FO_PRS_DDS_00212] [FO_PRS_DDS_00300] [FO_PRS_DDS_00301] [FO_PRS_DDS_00302] [FO_PRS_DDS_00303] [FO_PRS_DDS_00304] [FO_PRS_DDS_00305] [FO_PRS_DDS_00306] [FO_PRS_DDS_00307] [FO_PRS_DDS_00308] [FO_PRS_DDS_00309] [FO_PRS_DDS_00310] [FO_PRS_DDS_00311] [FO_PRS_DDS_00312] [FO_PRS_DDS_00313] [FO_PRS_DDS_00400] [FO_PRS_DDS_00401] [FO_PRS_DDS_00402] [FO_PRS_DDS_00403] [FO_PRS_DDS_00404] [FO_PRS_DDS_00405] [FO_PRS_DDS_00406] [FO_PRS_DDS_00407] [FO_PRS_DDS_00408] [FO_PRS_DDS_00409] [FO_PRS_DDS_00410] [FO_PRS_DDS_00411] [FO_PRS_DDS_00412] [FO_PRS_DDS_00413] [FO_PRS_DDS_00414] [FO_PRS_DDS_00415] [FO_PRS_DDS_00416] [FO_PRS_DDS_00417] [FO_PRS_DDS_00418] [FO_PRS_DDS_00419] [FO_PRS_DDS_00500] [FO_PRS_DDS_00501] [FO_PRS_DDS_00502] [FO_PRS_DDS_00503] [FO_PRS_DDS_00504] [FO_PRS_DDS_00505] [FO_PRS_DDS_00506] [FO_PRS_DDS_00507] [FO_PRS_DDS_00508] [FO_PRS_DDS_00509] [FO_PRS_DDS_00510]
[FO_RS_Dds_00002]	DDS standard serialization rules	[FO_PRS_DDS_00500] [FO_PRS_DDS_00501] [FO_PRS_DDS_00502] [FO_PRS_DDS_00503] [FO_PRS_DDS_00504] [FO_PRS_DDS_00505] [FO_PRS_DDS_00506] [FO_PRS_DDS_00507] [FO_PRS_DDS_00508] [FO_PRS_DDS_00509] [FO_PRS_DDS_00510]

△

Requirement	Description	Satisfied by
[FO_RS_Dds_00005]	DDS Quality of Service	[FO_PRS_DDS_00100] [FO_PRS_DDS_00104] [FO_PRS_DDS_00200] [FO_PRS_DDS_00204] [FO_PRS_DDS_00300] [FO_PRS_DDS_00304] [FO_PRS_DDS_00305] [FO_PRS_DDS_00306] [FO_PRS_DDS_00307] [FO_PRS_DDS_00400] [FO_PRS_DDS_00404] [FO_PRS_DDS_00405] [FO_PRS_DDS_00407] [FO_PRS_DDS_00410] [FO_PRS_DDS_00411] [FO_PRS_DDS_00412] [FO_PRS_DDS_00413] [FO_PRS_DDS_00603]
[FO_RS_Dds_00007]	Type Definition	[FO_PRS_DDS_00100] [FO_PRS_DDS_00101] [FO_PRS_DDS_00200] [FO_PRS_DDS_00201] [FO_PRS_DDS_00301] [FO_PRS_DDS_00302] [FO_PRS_DDS_00303] [FO_PRS_DDS_00401] [FO_PRS_DDS_00408] [FO_PRS_DDS_00409] [FO_PRS_DDS_00501] [FO_PRS_DDS_00502] [FO_PRS_DDS_00503] [FO_PRS_DDS_00504] [FO_PRS_DDS_00505] [FO_PRS_DDS_00506] [FO_PRS_DDS_00507] [FO_PRS_DDS_00508] [FO_PRS_DDS_00509] [FO_PRS_DDS_00510]
[FO_RS_Dds_00008]	Customization	[FO_PRS_DDS_00100] [FO_PRS_DDS_00104] [FO_PRS_DDS_00105] [FO_PRS_DDS_00106] [FO_PRS_DDS_00107] [FO_PRS_DDS_00108] [FO_PRS_DDS_00109] [FO_PRS_DDS_00110] [FO_PRS_DDS_00111] [FO_PRS_DDS_00112] [FO_PRS_DDS_00113] [FO_PRS_DDS_00200] [FO_PRS_DDS_00202] [FO_PRS_DDS_00203] [FO_PRS_DDS_00204] [FO_PRS_DDS_00205] [FO_PRS_DDS_00206] [FO_PRS_DDS_00207] [FO_PRS_DDS_00208] [FO_PRS_DDS_00209] [FO_PRS_DDS_00210] [FO_PRS_DDS_00211] [FO_PRS_DDS_00212] [FO_PRS_DDS_00300] [FO_PRS_DDS_00304] [FO_PRS_DDS_00305] [FO_PRS_DDS_00306] [FO_PRS_DDS_00307] [FO_PRS_DDS_00308] [FO_PRS_DDS_00309] [FO_PRS_DDS_00310] [FO_PRS_DDS_00311] [FO_PRS_DDS_00312] [FO_PRS_DDS_00313] [FO_PRS_DDS_00400] [FO_PRS_DDS_00402] [FO_PRS_DDS_00403] [FO_PRS_DDS_00404] [FO_PRS_DDS_00405] [FO_PRS_DDS_00406] [FO_PRS_DDS_00407] [FO_PRS_DDS_00410] [FO_PRS_DDS_00411] [FO_PRS_DDS_00412] [FO_PRS_DDS_00413] [FO_PRS_DDS_00414] [FO_PRS_DDS_00415] [FO_PRS_DDS_00416] [FO_PRS_DDS_00417] [FO_PRS_DDS_00418] [FO_PRS_DDS_00419]
[FO_RS_Dds_00010]	Safety mechanism	[FO_PRS_DDS_00601] [FO_PRS_DDS_00602] [FO_PRS_DDS_00603] [FO_PRS_DDS_00604]
[FO_RS_Dds_00015]	Publish	[FO_PRS_DDS_00102] [FO_PRS_DDS_00202] [FO_PRS_DDS_00304] [FO_PRS_DDS_00307] [FO_PRS_DDS_00308] [FO_PRS_DDS_00309] [FO_PRS_DDS_00313] [FO_PRS_DDS_00402] [FO_PRS_DDS_00414] [FO_PRS_DDS_00415] [FO_PRS_DDS_00419]

▽

△

Requirement	Description	Satisfied by
[FO_RS_Dds_00016]	Subscribe	[FO_PRS_DDS_00103] [FO_PRS_DDS_00104] [FO_PRS_DDS_00105] [FO_PRS_DDS_00203] [FO_PRS_DDS_00204] [FO_PRS_DDS_00205] [FO_PRS_DDS_00211] [FO_PRS_DDS_00212] [FO_PRS_DDS_00305] [FO_PRS_DDS_00306] [FO_PRS_DDS_00310] [FO_PRS_DDS_00311] [FO_PRS_DDS_00312] [FO_PRS_DDS_00403] [FO_PRS_DDS_00404] [FO_PRS_DDS_00405] [FO_PRS_DDS_00416]
[RS_CM_00204]	The Communication Management shall map the protocol independent Service Oriented Communication to the configured protocol binding and shall execute the protocol accordingly.	[FO_PRS_DDS_00416]
[RS_CM_00212]	Communication Management shall provide an interface to call methods of other applications synchronously	[FO_PRS_DDS_00416]
[RS_CM_00213]	Communication Management shall provide an interface to call service methods asynchronously	[FO_PRS_DDS_00416]
[RS_CM_00220]	Communication Management shall trigger the set method of the application which provides the field	[FO_PRS_DDS_00416]
[RS_CM_00221]	Communication Management shall trigger the get method of the application which provides the field	[FO_PRS_DDS_00416]

Table 4.1: Requirements Tracing

5 Definition of terms and acronyms

5.1 Acronyms and abbreviations

Abbreviation / Acronym:	Description:
OMG	Object Management Group
QoS	Quality of Service
DDSI	Data Distribution Service Interoperability
RTPS	Real-Time Publish-Subscribe
XTYPES	eXtensible Types

Table 5.1: Acronyms and abbreviations used in the scope of this Document

5.2 Definition of terms

Terms:	Description:
Entity	The base class for all other DDS Entities.
DomainParticipant	Represents the participation of the application on a communication plane that isolates applications running on the same set of physical computers from each other.
Topic	Reprepsents the most basic description of the data to be published and subscribed.
Publisher	Provides the actual dissemination of publications.
DataWriter	Provides the application functionality to set the value of the data to be published under a given Topic.
Subscriber	Provides the actual reception of the data resulting from its subscriptions.
DataReader	Provides the application with (1) functionality to declare the data it wishes to receive (i.e., make a subscription) and (2) to access the data received by the attached Subscriber.
QoS Profile	Grouping of QoS Policy values applicable to one or more DDS Entities.

Table 5.2: Definition of terms in the scope of this Document

6 Protocol specification

6.1 Introduction

In the scope of the Service-Oriented Discovery protocol protocol three distinct Resource Identification Mechanisms can be configured, defining how Service Interfaces and their individual Instances (the "Resources") are uniquely instantiated and addressable with a particular DDS Domain:

- The Partition -based mechanism, where DDS Publisher and Subscriber Entity PARTITION QoS policy is leveraged to isolate each Service Instance and their consumers into a uniquely named DDS Partition. De-facto choice in:
  - DDS DomainParticipant QoS -based discovery protocol.

  But also available in:
  - DDS Topic -based discovery protocol.
- The DDS Topic Prefix -based mechanism, where unique Service Instance Identifiers are included in all the DDS Topic names conforming the Service Interface. Available in:
  - DDS Topic -based discovery protocol.
- The Instance -based mechanism, where in-band (i.e. included in AUTOSAR DDS Data Types) unique Service Instance Identifier Fields are used to uniquely identify different Service Instances. Available in:
  - DDS Topic -based discovery protocol.

As shown in the protocol specification items to follow, the choice of Resource Identification Mechanism influences how different Service Instances convey Elements such as Events, Triggers, Methods and Fields over DDS.

6.2 Message format

Message format is defined by the OMG® DDSI-RTPS standard ([1]).

6.3 Message types

Message types are defined by the OMG® DDSI-RTPS standard ([1]).

6.4 Services / Commands

Not applicable.

6.5 Sequences (lower layer)

Sequences are defined by the OMG® DDS® standard ([3]).

6.6 Error messages

Error messages are defined by the OMG® DDSI-RTPS standard ([1]).

6.7 Handling Events

[FO_PRS_DDS_00100] Mapping `Events` to DDS Topics

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00007, FO_RS_Dds_00008

⌈Every `Event` of a `ServiceInterface` shall be mapped to a DDS `Topic`. This DDS `Topic` shall be configured as follows:

- The DDS `Topic` name shall be derived according to the following rules:
  - If the Service Instance has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION` or `SERVICE_INSTANCE_RESOURCE_INSTANCE_ID`, then the DDS `Topic` name shall be set to `ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<eventTopicName>`
  - Additionally, if the provided or consumed Service Instance has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, then samples of this DDS `Topic` shall be sent and received via DDS `DataWriter`s and `DataReader`s whose respective parent DDS `Publisher` and `Subscriber` objects include the following partition in the PARTITION QoS policy: `ara.com://services/<svcId>/<svcInId>`
  - Finally, if the provided or consumed Service Instance has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set to `ara.com://services/<svcId>/<svcInId>/<eventTopicName>`
  - Where:

    `<svcId>` is the value of `<DDSServiceInterfaceID>` for the Service Interface.

    `<svcInId>` is the stringified value of `<DDSServiceInstanceID>` for the Service Instance.

    `<svcMajVersion>` is the stringified value of `<DDSInterfaceMajorVersion>` for the Service Interface.

    `<svcMinVersion>` is the stringified value of `<DDSInterfaceMinorVersion>` for the Service Interface.

    `<eventTopicName>` is the value of `<DDSEventTopicName>` for the Service Interface Event.

- The Topic Data Type of the DDS `Topic` shall be defined as specified in [FO_PRS_DDS_00101], and shall be registered under the equivalent data type name.

⌋

[FO_PRS_DDS_00101] DDS `Topic` data type definition

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of the DDS `Topic` representing an `Event` shall be constructed according to the following IDL definition:

```
struct <EventTypeName>EventType {
    @key uint16 instance_id;
    <EventTypeName> data;
};
```

Where:

`<EventTypeName>` is the symbol defined for the `Implementation Data Type` associated with the `Event`

`instance_id` is a `@key` member of the type, which identifies all samples with the same `instance_id` as samples of the same DDS `Topic` Instance.

`data` is the actual value of the `Event`, which shall be constructed and encoded according to the DDS serialization rules. The `@external` annotation is optionally allowed, for cases where references yield implementation benefits over values.

⌋

[FO_PRS_DDS_00102] Sending an `Event` sample

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00015

⌈When instructed to send an `Event` sample, a new sample shall be constructed of the equivalent DDS `Topic` data type according to [FO_PRS_DDS_00101].

- The Instance Id field (`instance_id`) shall be set to the value of `<DDSServiceInstanceID>`.
- The Data field (`data`) shall be set to the data to be sent.

This sample shall be then passed as a parameter to the `write()` function of the DDS `DataWriter` associated with the `Event`, which shall serialize the sample according to the serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

[FO_PRS_DDS_00103] Subscribing to an `Event`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00016

⌈When instructed to subscribe to an `Event` with a given cache size, a DDS `DataReader` (see [FO_PRS_DDS_00104]) shall be created using a DDS `Subscriber` according to [FO_PRS_DDSSD_00101], [FO_PRS_DDSSD_00105], [FO_PRS_DDSSD_00201] and [FO_PRS_DDSSD_00205].⌋

[FO_PRS_DDS_00104] Creating a DDS `DataReader` for `Event` subscription

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00016, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈A DDS `DataReader` for the DDS `Topic` associated with the `Event` of a `ServiceInterface` (see [FO_PRS_DDS_00100]) shall be created. If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the service instance it is bound to, the DDS DDS `Subscriber` created in [FO_PRS_DDSSD_00105] or [FO_PRS_DDSSD_00205] shall be used to create the DDS `DataReader`.

The DDS `DataReader` shall be configured as follows:

- `DataReaderQos`: Defines the DDS `QoS Profile` to be used for the DDS `DataReader`, obtained from `<DDSServiceInstanceEventQosProfile>`. To configure the `DataReader`'s cache size, the value of the `DataReader`'s HISTORY QoS shall be overridden as follows:
  - `history.kind = KEEP_LAST_HISTORY_QOS`
  - `history.depth = <cache size value>`
- `Listener`: A DDS `DataReader Listener` listener as per [FO_PRS_DDS_00105].
- `StatusMask`: Shall be set to `STATUS_MASK_NONE`

⌋

[FO_PRS_DDS_00105] Defining a DDS `DataReader Listener` for a subscribed `Event`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈A DDS DDS `DataReader Listener` instance shall be created, capable of handling notifications when a new sample is received and/or when the matched status of the `Event` subscription changes. This object shall handle samples of the DDS `Topic` data type specified in [FO_PRS_DDS_00101].

The DDS DataReader Listener shall provide the following callbacks according to the specified instructions:

- An on_data_available() callback that dispatches received event samples to upper layer handler when valid samples become avilable in the DDS DataReader cache.
- An on_subscription_matched() callback that forwards the subscription state to upper layers handlers.

⌋

[FO_PRS_DDS_00106] Unsubscribing from an Event

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service Event, the DDS DataReader associated with the Event shall be deleted.⌋

[FO_PRS_DDS_00107] Obtaining subscription state from an Event

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the subscription state, the existence of a DDS DataReader associated with the Event subscription (see [FO_PRS_DDS_00103]) shall be checked:

- If the DDS DataReader does exist, the DDS DataReader's get_subscription_matched_status() function shall be called, then:
  - If the total_count attribute of the resulting SubscriptionMatchedStatus is greater than zero, the subscription state shall be determined as subscribed.
  - Otherwise, the subscription state shall be determined as pending.
- Otherwise, if the DDS DataReader does not exist, the subscription state shall be determined as not subscribed.

⌋

[FO_PRS_DDS_00108] Retrieving new data samples from an Event

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to retrieve new Event data samples, a take() operation shall be performed on the DDS DataReader associated to the Event.⌋

[FO_PRS_DDS_00109] Requesting number of free sample slots from an `Event`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the number of free sample slots, the number of free sample slots in the DDS `DataReader` cache shall be returned.⌋

[FO_PRS_DDS_00110] Registering an `Event` reception handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register an `Event` reception handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00105]) using the `get_listener()` function.
- Set the `Listener`'s `on_data_available` callback to the new event reception handler.
- Update the DDS `DataReader`'s `Listener` by calling `set_listener()` with `listener` equal to the new `Listener` object.
- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋

[FO_PRS_DDS_00111] Unregistering an `Event` reception handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister an `Event` reception handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00105]) using the `get_listener()` function.
- Set the `Listener`'s `on_data_available` callback to `NULL`.
- Update the DDS `DataReader`'s listener by calling `set_listener()` with `listener` equal to the new `Listener` object.

- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.

⌋

[FO_PRS_DDS_00112] Registering an `Event` subscription state change handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register an `Event` subscription state change handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00105]) using the `get_listener()` function.
- Set the `Listener`'s `set_subscription_state_change_handler` callback to the new event reception handler.
- Update the DDS `DataReader`'s listener by calling `set_listener()` with `listener` equal to the new `Listener` object.
- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋

[FO_PRS_DDS_00113] Unregistering an `Event` subscription state change handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister an `Event` subscription state change handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00105]) using the `get_listener()` function.
- Set the `Listener`'s `set_subscription_state_change_handler` callback to `NULL`.
- Update the DDS `DataReader`'s listener by calling `set_listener()` with `listener` equal to the new `Listener` object.
- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `SUBSCRIPTION_MATCHED_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS`.

⌋

6.8 Handling Triggers

[FO_PRS_DDS_00200] Mapping `Triggers` to DDS Topics

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00007, FO_RS_Dds_00008

⌈Every `Trigger` of a `ServiceInterface` to a DDS `Topic`. This DDS `Topic` shall be configured as follows:

- The DDS `Topic` name shall be derived according to the following rules:
  - If the `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION` or `SERVICE_INSTANCE_RESOURCE_INSTANCE_ID`, then the DDS `Topic` name shall be set to `ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<triggerTopicName>`
  - Additionally, if the provided or consumed `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, then samples of this DDS `Topic` shall be sent and received via DDS `DataWriter`s and `DataReader`s whose respective parent DDS `Publisher` and `Subscriber` objects include the following partition in the PARTITION QoS policy: `ara.com://services/<svcId>/<svcInId>`
  - Finally, if the provided or consumed `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set to `ara.com://services/<svcId>/<svcInId>/<triggerTopicName>`
  - Where:

    `<svcId>` is the value of `<DDSServiceInterfaceID>` for the `Service Interface`.

    `<svcInId>` is the stringified value of `<DDSServiceInstanceID>` for the `Service Instance`.

    `<svcMajVersion>` is the stringified value of `<DDSInterfaceMajorVersion>` for the `Service Interface`.

    `<svcMinVersion>` is the stringified value of `<DDSInterfaceMinorVersion>` for the `Service Interface`.

    `<triggerTopicName>` is the value of `<DDSTriggerTopicName>` for the `Service Interface Event`.

- The DDS `Topic` Data Type shall be defined as specified in [FO_PRS_DDS_00201], and shall be registered under the equivalent data type name.

⌋

[FO_PRS_DDS_00201] DDS `Topic` data type definition

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of the DDS `Topic` representing a `Trigger` shall be constructed according to the following IDL definition:

```
1 struct TriggerType {
2     @key uint16 instanceIdentifier;
3 };
```

Where:

`instance_id` is a `@key` member of the type, which identifies all samples with the same `instance_id` as samples of the same DDS `Topic` Instance.

⌋

[FO_PRS_DDS_00202] Sending a `Trigger` sample

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to send a `Trigger` sample, a new sample shall be constructed of the equivalent DDS `Topic` data type according to [FO_PRS_DDS_00201].

This sample shall be then passed as a parameter to the `write()` function of the DDS `DataWriter` associated with the `Trigger`, which shall publish it over DDS.⌋

[FO_PRS_DDS_00203] Subscribing to a `Trigger`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to subscribe to a `Trigger`, a DDS `DataReader` (see [FO_PRS_DDS_00204]) shall be created using a DDS `Subscriber` according to [FO_PRS_DDSSD_00101], [FO_PRS_DDSSD_00105], [FO_PRS_DDSSD_00201] and [FO_PRS_DDSSD_00205].⌋

[FO_PRS_DDS_00204] Creating a DDS `DataReader` for `Trigger` subscription

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈A DDS `DataReader` for the DDS `Topic` associated with the `Trigger` of a `ServiceInterface` (see [FO_PRS_DDS_00200]) shall be created. If the provided or consumed `Service Instance` has been advertised with the `identifier_type` at-

tribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure the proxy communicates only with the Service Instance it is bound to, the DDS Subscriber created in [FO_PRS_DDSSD_00105] or [FO_PRS_DDSSD_00205] shall be used to create the DDS DataReader.

The DDS DataReader shall be configured as follows:

- DataReaderQos: Defines the DDS QoS Profile to be used for the DDS DataReader, obtained from <DDSServiceInstanceTriggerQosProfile> .
- Listener: A DDS DataReader Listener instance as per [FO_PRS_DDS_00205].
- StatusMask: Shall be set to STATUS_MASK_NONE

⌋

[FO_PRS_DDS_00205] Defining a DDS DataReader Listener for a subscribed Trigger

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈A DDS DataReader Listener object shall be created, capable of handling notifications when a new sample is received and/or when the matched status of the Trigger subscription changes. This object shall handle samples of the DDS Topic data type specified in [FO_PRS_DDS_00201].

The DDS DataReader Listener shall provide the following callbacks according to the specified instructions:

- An on_data_available() callback that dispatches received Trigger samples to upper layer handler when valid samples become avilable in the DDS DataReader cache.
- An on_subscription_matched() callback that forwards the subscription state to upper layers handlers.

⌋

[FO_PRS_DDS_00206] Unsubscribing from an Trigger

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service Trigger, the DDS DataReader associated with the Trigger shall be deleted.⌋

[FO_PRS_DDS_00207] Obtaining subscription state from a `Trigger`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the subscription state, the existence of a DDS `DataReader` associated with the `Trigger` subscription (see [FO_PRS_DDS_00204]) shall be checked:

- If the DDS `DataReader` does exist, the DDS `DataReader`'s `get_subscription_matched_status()` function shall be called, then:
  - If the `total_count` attribute of the resulting `SubscriptionMatchedStatus` is greater than zero, the subscription state shall be determined as subscribed.
  - Otherwise, the subscription state shall be determined as pending.
- Otherwise, if the DDS `DataReader` does not exist, the subscription state shall be determined as not subscribed.

⌋

[FO_PRS_DDS_00208] Retrieving new notification from a `Trigger`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to retrieve new `Trigger` notifications, a `take()` operation shall be performed on the DDS `DataReader` associated to the `Trigger`, recording the total cound and discarding the samples themselves.⌋

[FO_PRS_DDS_00209] Registering a `Trigger` reception handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register a `Trigger` reception handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00204]) using the `get_listener()` function.
- Set the `Listener`'s `on_data_available` callback to the new `Trigger` reception handler.
- Update the DDS `DataReader`'s `Listener` by calling `set_listener()` with `listener` equal to the new `Listener` object.
- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.

- If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
- If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋

[FO_PRS_DDS_00210] Unregistering a `Trigger` reception handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister an `Trigger`, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00204]) using the `get_listener()` function.
- Set the `Listener`'s `on_data_available` callback to `NULL`.
- Update the DDS `DataReader`'s listener by calling `set_listener()` with `listener` equal to the new `Listener` object.
- Set `StatusMask` as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.

⌋

[FO_PRS_DDS_00211] Registering a `Trigger` subscription state change handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to register a `Trigger` subscription state change handler, the following operations shall be performed:

- Get a reference to the DDS `DataReader`'s `Listener` (see [FO_PRS_DDS_00204]) using the `get_listener()` function.
- Set the `Listener`'s `set_subscription_state_change_handler` callback to the new `Trigger` reception handler.

- Update the DDS DataReader's listener by calling set_listener() with listener equal to the new Listener object.
- Set StatusMask as follows:
  - If the original value of StatusMask was STATUS_MASK_NONE or SUBSCRIPTION_MATCHED_STATUS, set it to SUBSCRIPTION_MATCHED_STATUS.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS, set it to DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS, set it to DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS.

⌋

[FO_PRS_DDS_00212] Unregistering a `Trigger` subscription state change handler

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to unregister a Trigger subscription state change handler, the following operations shall be performed:

- Get a reference to the DDS DataReader's Listener (see [FO_PRS_DDS_00204]) using the get_listener() function.
- Set the Listener's set_subscription_state_change_handler callback to NULL.
- Update the DDS DataReader's listener by calling set_listener() with listener equal to the new Listener object.
- Set StatusMask as follows:
  - If the original value of StatusMask was STATUS_MASK_NONE or SUBSCRIPTION_MATCHED_STATUS, set it to STATUS_MASK_NONE.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS, set it to DATA_AVAILABLE_STATUS.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS, set it to DATA_AVAILABLE_STATUS.

⌋

6.9 Handling Method Calls

The RPC over DDS Specification (DDS-RPC) [4] introduces the concept of DDS Services. These Services provide the mechanisms required to define and implement methods that can be invoked remotely by DDS "client" applications using the building blocks of the DDS data-centric publish-subscribe middleware [3]. In this section, we specify how to handle service-oriented method calls over DDS by defining the appropriate mapping between ara::com service Methods and DDS service methods.

[FO_PRS_DDS_00300] Mapping Methods to DDS Topics

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈Every Service Interface containing one or more Methods shall have an associated set of DDS Topics enabling Service Instances to offer those Methods, and to enable client applications to invoke them.

DDS Topics shall be constructed according to the Basic Service Mapping Profile of the RPC over DDS specification [4], which assigns two DDS Topics to every DDS Service: a Request DDS Topic and a Reply Topic. Thus, every Service Interface containing one or more Methods shall prompt the creation of two equivalent DDS Topics.

The equivalent DDS Request Topic shall be configured as follows:

- The Request DDS Topic Name shall be derived from the Manifest according to the following rules:
  - If the Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the DDS Topic name shall be set to ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<methodRequestTopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this DDS Topic shall be sent and received via DDS DataWriters and DataReaders whose respective parent DDS Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<svcId>/<svcInId>
  - Finally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<svcId>/<svcInId>/<methodRequestTopicName>
  - Where:

<svcId> is the value of <DDSServiceInterfaceID> for the Service Interface.

<svcInId> is the stringified value of <DDSServiceInstanceID> for the Service Instance.

<svcMajVersion> is the stringified value of <DDSInterfaceMajorVersion> for the Service Interface.

<svcMinVersion> is the stringified value of <DDSInterfaceMinorVersion> for the Service Interface.

<methodRequestTopicName> is the value of <DDSMethodRequestTopicName> for the Service Interface Event.

- The Request DDS Topic Data Type shall be defined as specified in [FO_PRS_DDS_00301], and shall be registered under the equivalent data type's name.

The equivalent DDS Reply Topic shall be configured as follows:

- The Reply DDS Topic Name shall be derived from the Manifest according to the following rules:
  - If the Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the DDS Topic name shall be set to ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<methodReplyTopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this DDS Topic shall be sent and received via DDS DataWriters and DDS DataReaders whose respective parent DDS Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<svcId>/<svcInId>
  - Finally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<svcId>/<svcInId>/<methodReplyTopicName>
  - Where:

    <svcId> is the value of <DDSServiceInterfaceID> for the Service Interface.

    <svcInId> is the stringified value of <DDSServiceInstanceID> for the Service Instance.

<svcMajVersion> is the stringified value of <DDSInterfaceMajorVersion> for the Service Interface.

<svcMinVersion> is the stringified value of <DDSInterfaceMinorVersion> for the Service Interface.

<methodReplyTopicName> is the value of <DDSMethodReplyTopicName> for the Service Interface Event.

- The Reply DDS Topic Data Type shall be defined as specified in [FO_PRS_DDS_00302], and shall be registered under the equivalent data type's name.

⌋

[FO_PRS_DDS_00301] Mapping Methods to DDS Topic request data type definition

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈As specified in section 7.5.1.1.6 of [4], the Request DDS Topic data type is a structure composed of a Request Header with meta-data, and a Call Structure with data. The IDL definition of the Request DDS Topic data type is the following:

```
struct <svcId>Method_Request {
    dds::rpc::RequestHeader header;
    <ServiceInterafaceID>Method_Call data;
};
```

Where:

<svcId> is the value of <DDSServiceInterfaceID>.

dds::rpc::RequestHeader is the standard Request Header defined in section 7.5.1.1.1 of [4].

<svcId>Method_Call is the union that holds the value of the input parameters of the corresponding Methods, according to the rules specified in section 7.5.1.1.6 of [4].

dds::rpc::RequestHeader shall be constructed as specified in section 7.5.1.1.1 of [4]. On top of that, instanceName (a member of the RequestHeader structure that specifies the DDS Service Instance name) shall be set to a string representation of the <DDSServiceInstanceID> value of the Service Instance that provides the Methods.

<svcId>Method_Call shall be constructed as specified in section 7.5.1.1.6 of [4]:

- The name of the union shall be <svcId>Method_Call.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type dds::rpc::UnknownOperation (defined in section 7.5.1.1.1 of [4]) for unsupported and unknown operations.

- The union shall have a case label for each `Method` defined in the `Service Interface`, where:
  - The integer value of the case label shall be a 32-bit hash of the string representation of `<DDSServiceInterfaceMethodName>`. The DDS Communication Protocol implementation shall compute the hash as specified in section 7.5.1.1.2 of [4]. Representations of the `Service Interface` in OMG IDL [5] shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Method_<methodName>_Hash;` where `<methodName>` is `<DDSServiceInterfaceMethodName>`), in order to simplify the representation of the union cases (see below).
  - The member name for the case label shall be the value of `<DDSServiceInterfaceMethodName>`.
  - The type for each case label shall be `<svcId>Method_<methodName>_In`, which shall be constructed as specified in section 7.5.1.1.4 of [4] (see below).

The IDL definition of `<svcId>Method_Call` is the following:

```
union <svcId>Method_Call switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Method_<method0Name>_Hash:
    <svcId>Method_<method0Name>_In <method0Name>;
case <svcId>Method_<method1Name>_Hash:
    <svcId>Method_<method1Name>_In <method1Name>;
// ...
case <svcId>Method_<methodNName>_Hash:
    <svcId>Method_<methodNName>_In <methodNName>;
};
```

As defined in section 7.5.1.1.4 of [4], the `<svcId>Method_<methodName>_In` structure shall contain as members all the input and input/output parameters of the `Method`. The IDL representation of `<svcId>Method_<methodName>_In` is the following:

```
struct <svcId>Method_<methodName>_In {
    <ArgumentDataPrototype[0]>;
    <ArgumentDataPrototype[1]>;
    // ...
    <ArgumentDataPrototype[n]>;
};
```

In accordance with [4], for `Method`s with no input parameters, a `<svcId>Method_<methodName>_In` structure with a single member named `dummy` of type `dds::rpc::UnusedMember` (see section 7.5.1.1.1 of [4]) shall be generated.

The resulting Request DDS `Topic` data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Method_Call` union, shall be serialized as specified in section 7.4.3.5 of [2].⌋

[FO_PRS_DDS_00302] Mapping `Methods` to DDS `Topic` reply data type definition

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈As specified in section 7.5.1.1.7 of [4], the Reply DDS `Topic` data type is a structure composed of a Reply Header with meta-data and a Return Structure with data. The IDL definition of the Reply DDS `Topic` data type is the following:

```
struct <svcId>Method_Reply {
    dds::rpc::ReplyHeader header;
    <svcId>Method_Return data;
};
```

Where:

`<svcId>` is the value of `<DDSServiceInterfaceID>`.

`dds::rpc::ReplyHeader` is the standard Reply Header defined in section 7.5.1.1.1 of [4].

`<svcId>Method_Return` is the union that holds the return values (i.e., return values, output parameter values, and/or errors) of the corresponding response, according to the rules specified in section 7.5.1.1.7 of [4].

`dds::rpc::ReplyHeader` shall be constructed as specified in section 7.5.1.1.1 of [4].

`<svcId>Method_Return` shall be constructed as specified in section 7.5.1.1.7 of [4]:

- The name of the union shall be `<svcId>Method_Return`.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type `dds::rpc::UnknownOperation` (defined in section 7.5.1.1.1 of [4]) for unsupported and unknown operations.
- The union shall have a case label for each `Method` defined in the `Service Interface`, where:
  - The integer value of the case label shall be a 32-bit hash of the string representation of `<DDSServiceInterfaceMethodName>`. The DDS Communication Protocol implementation shall compute the hash as specified in section 7.5.1.1.2 of [4]. Representations of the `Service Interface` in OMG IDL [5] shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Method_<methodName>_Hash`; where `<methodName>` is `<DDSServiceInterfaceMethodName>`), in order to simplify the representation of the union cases (see below).
  - The member name for the case label shall be the value of `<DDSServiceInterfaceMethodName>`.
  - The type for each case label shall be `<svcId>Method_<methodName>_Result`, which shall be constructed as specified in section 7.5.1.1.4 of [4] (see below).

The IDL definition of `<svcId>Method_Return` is the following:

```
union <svcId>Method_Return switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Method_<method0Name>_Hash:
    <svcId>Method_<method0Name>_Result <method0Name>;
case <svcId>Method_<method1Name>_Hash:
    <svcId>Method_<method1Name>_Result <method1Name>;
// ...
case <svcId>Method_<methodNName>_Hash:
    <svcId>Method_<methodNName>_Result <methodNName>
};
```

As defined in section 7.5.1.1.5 of [4], the `<svcId>Method_<methodName>_Result` union shall be constructed as follows:

- The union discriminator shall be a 32-bit signed integer.
- The union shall have a case with label `dds::RETCODE_OK` to represent a successful return:
  - The value of `RETCODE_OK` shall be 0x00, as specified in section 2.3.3 of [3].
  - The successful case shall have a single member named `result` of type `<svcId>Method_<methodName>_Out` (see below).
- The union shall also have a case with label `dds::RETCODE_ERROR` to represent the possible application-layer errors the `Method` may return:
  - The value of `RETCODE_ERROR` shall be 0x01, as specified in section 2.3.3 of [3].
  - The error case shall have a single member named `error` of type `ara::core::ErrorCode` (see [FO_PRS_DDS_00303] below).

The IDL representation of `<svcId>Method_<methodName>_Result` is the following:

```
union <svcId>Method_<methodName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Method_<methodName>_Out result;
case dds::RETCODE_ERROR:
    ara::core::ErrorCode error;
};
```

Lastly, as defined in section 7.5.1.1.5 of [4], the `<svcId>Method_<methodName>_Out` structure be constructed as follows:

- The structure shall contain as members all the input/output and output parameters of the `Method`.
- The members of the structure representing input/output and output arguments shall appear in the structure in the same order as they were declared.

- If the `Method` has no input/output and no output arguments, the structure shall contain a single member named `dummy` of type `dds::rpc::UnusedMember` (in accordance with section 7.5.1.1.1 of [4]).

The IDL representation of `<svcId>Method_<methodName>_Out` is the following:

```
struct <svcId>Method_<methodName>_Out {
    <ArgumentDataPrototype[0]>;
    <ArgumentDataPrototype[1]>;
    // ...
    <ArgumentDataPrototype[n]>;
};
```

The resulting Reply DDS `Topic` data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Method_<methodName>_Result` union, shall be serialized as specified in section 7.4.3.5 of [2].⌋

[FO_PRS_DDS_00303] Mapping of ara::core::ErrorCode

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Application-layer errors shall be represented according to the following IDL [5]:

```
module dds {
module ara {
module core {

struct ErrorCode {
    uint64 error_domain_value;
    int32 error_code;
};

}; // module core
}; // module ara
}; // module dds
```

Where:

`error_domain_value` is a 64-bit unsigned integer representing the application domain of the error (see `<DDSServiceInterfaceMethodErrorDomain>`).

`error_code` is a 32-bit signed integer representing the actual error code (see `<DDSServiceInterfaceMethodErrorCode>`).

`ara::core::ErrorCode` shall be serialized according to the DDS serialization rules.⌋

The DDS serialization rules are defined in section 6.11.

[FO_PRS_DDS_00304] Creating a DDS `DataWriter` to handle `Method` requests on the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈A DDS `DataWriter` shall be created for the Request DDS `Topic` associated with the `Method`s of the `Service Interface` (see [FO_PRS_DDS_00301]) upon client instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the client communicates only with the `Service Instance` it is bound to, the DDS `Publisher` created in [FO_PRS_DDSSD_00105] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataWriter`.

The DDS `DataWriter` shall be configured as follows:

- `DataWriterQos` shall be set to `<DDSServiceInstanceQosProfile>`.

⌋

[FO_PRS_DDS_00305] Creating a DDS `DataReader` to handle `Method` responses on the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈A DDS `DataReader` shall be created for the Reply DDS `Topic` associated with the `Method`s of the `Service Interface` (see [FO_PRS_DDS_00302]) upon client instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the client communicates only with the `Service Instance` it is bound to, the DDS `Subscriber` created in [FO_PRS_DDSSD_00105] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataReader`.

The DDS `DataReader` shall be configured as follows:

- `DataReaderQos` shall be set to `<DDSServiceInstanceQosProfile>`.

⌋

[FO_PRS_DDS_00306] Creating a DDS `DataReader` to handle `Method` requests on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈A DDS `DataReader` shall be created for the Request DDS `Topic` associated with the `Method`s of the `Service Interface` (see [FO_PRS_DDS_00301]) upon server instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the clients communicate only with the `Service Instance` it is bound to, the DDS `Subscriber` created in [FO_PRS_DDSSD_00101] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataReader`.

The DDS `DataReader` shall be configured as follows:

- `DataReaderQos` shall be set to `<DDSServiceInstanceQosProfile>`.
- `Listener` and `StatusMask` shall be set according to the desired `Method` call processing mode:
  - For an asynchronous call processing mode, `Listener` shall be set to an instance of the DDS `DataReader Listener` class specified in [FO_PRS_DDS_00311], and `StatusMask` shall be set to `DATA_AVAILABLE_STATUS`.
  - For a synchronous call processing mode, `Listener` shall remain unset, and `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋

[FO_PRS_DDS_00307] Creating a DDS `DataWriter` to handle `Method` responses on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈A DDS `DataWriter` shall be created for the Reply DDS `Topic` associated with the `Method`s of the `Service Interface` (see [FO_PRS_DDS_00302]) upon server instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the clients communicate only with the `Service Instance` it is bound to, the DDS `Publisher` created in [FO_PRS_DDSSD_00101] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataWriter`.

The DDS `DataWriter` shall be configured as follows:

- DataReaderQos shall be set to <DDSServiceInstanceQosProfile>.

⌋

[FO_PRS_DDS_00308] Calling a service Method from the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to call a Method from the client side, a new sample of the Request Topic—an instance of the Request DDS Topic data type defined in [FO_PRS_DDS_00301])—shall be constructed as follows:

- To initialize the RequestHeader object,
  - requestId shall be set according to the rules specified in [4].
  - instanceName shall be set to the string-representation value of <DDSServiceInstanceID> for the remote Service Instance.
- To initialize the <svcId>Method_Call object, the appropriate union case (as specified in [FO_PRS_DDS_00301], the hash of the Method's name is the union discriminator that selects the union case) shall be selected, and then set accordingly the structure containing all the input and input/output arguments.

That sample shall then be passed as a parameter to the write() Method of the DDS DataWriter created in [FO_PRS_DDS_00304] to handle Method requests on the client side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

[FO_PRS_DDS_00309] Notifying the client of a response to a Method call

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈To notify the client application of a response as a result of a Method call, either the result or the error values shall be extracted from the Reply DDS Topic sample.

If the discriminator of the <svcId>Method_<methodName>_Result union holding the response for the specific Method call in the received DDS Reply DDS Topic sample is dds::RETCODE_OK (i.e., 0 as defined in [3]), the Method result values shall be set using the members representing the input/output and output arguments in the corresponding <DDSServiceInterfaceID>Method_<methodName>_Out result (see [FO_PRS_DDS_00302]).

Else, for any other discriminator value, the Method error values shall be set to the received ara::core::ErrorCode (see [FO_PRS_DDS_00302]).

In either case, the associated processing shall be performed upon the reception of a new Reply DDS `Topic` sample by the corresponding DDS `DataReader` (see [FO_PRS_DDS_00305]).

The DDS `DataReader`'s `take()` operation shall be used to process the sample. Moreover, to correlate a request with a response, the `header.relatedRequestId` of the received sample shall be compared with the original `requestId` that was set and sent in [FO_PRS_DDS_00308]. [1] [2]

If a received `relatedRequestId` does not correspond to a `requestId` that has been sent by the client, the response shall be discarded.⌋

[FO_PRS_DDS_00310] Processing a `Method` call on the server side (asynchronous)

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case asynchronous request processing is expected, a DDS `DataReader Listener` shall be created to process the requests asynchronously—as described in [FO_PRS_DDS_00311]—with an instance of it attached to the DDS `DataReader` processing the requests in accordance with [FO_PRS_DDS_00306]. The listener is responsible for identifying the internal callback processing the request, and dispatching request data to it (see [FO_PRS_DDS_00311]).⌋

[FO_PRS_DDS_00311] Creating a DDS `DataReader` Listener to process asynchronous requests on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈According to [FO_PRS_DDS_00306], asynchronous request processing requires the instantiation of a DDS `DataReader Listener` to process asynchronously requests on the server side. This object shall handle samples of the DDS `Topic` data type specified in [FO_PRS_DDS_00301].

The DDS `DataReader Listener` shall implement the following `Method`s according to the specified instructions:

- An `on_data_available() Method` responsible for reading the received requests from the DDS `DataReader`'s cache—using the `take()` operation—and dispatching them to the appropriate `Method`s for processing. To identify the call-

[1] The RPC over DDS specification [4] does not mandate a specific mechanism or context to invoke the `take()` operation on the DDS `DataReader` that subscribes to `Method` replies. Implementers of this specification may therefore follow different approaches to address this issue.

[2] For instance, a proxy could use a dictionary-like data structure to temporarily hold the request data to every request (keyed by their `dds::SampleIdentity requestId`), and install a DDS `DataReader Listener` (on the DDS `DataReader` created in [FO_PRS_DDS_00305]) with an `on_data_available() Method` that could notify of reply reception to the Application layer, using the `relatedRequestId` of the received Reply DDS `Topic` sample to address it. Alternatively, a compliant solution could also call `take()` in an asynchronous context using a `dds::core::Waitset` [3] to block until the reception of the expected sample.

back that shall process each request, `on_data_available()` shall use the union discriminator of the `<svcId>Method_Call` and provide the destination callback with the specific arguments in the union case.

⌋

[FO_PRS_DDS_00312] Processing a `Method` call on the server side (synchronous)

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case synchronous request processing is expected, the protocol implementaiton is responsible for calling `take()` on the DDS `DataReader` processing the Request DDS `Topic` associated with the service (see [FO_PRS_DDS_00306]).

Each synchronous operation shall `take()` only the first sample from the DDS `DataReader`'s cache and dispatch the call to the appropriate callback according to the value of the of the discriminator of the `<svcId>Method_Call` union and provide the destination `Method` with the specific arguments in the union case.⌋

[FO_PRS_DDS_00313] Sending a `Method` call response from the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈Upon the return (either as a result of a normal return or through one of the possible Application-layer errors) of the `Service Instance Method` a response shall be sent.

To send this response, a new sample of the Reply DDS `Topic` —an instance of the Reply DDS `Topic` data type defined in [FO_PRS_DDS_00302])—shall be constructed as follows:

- To initialize the `ReplyHeader` object,
  - `relatedRequestId` shall be set to the value of the `header.requestId` attribute of the request that triggered the `Method` call (see [FO_PRS_DDS_00308]).
- To initialize the `<svcId>Method_Return` object:
  - Select the appropriate union case (as specified in [FO_PRS_DDS_00302], the hash of the `Method`'s name is the union discriminator that selects the union case).
  - Set the `<svcId>Method_<methodName>_Result` union selecting its union discriminator based on whether the operation generated a nominal result or raised an error:
    - If operation generated a nominal result, union case for `dds::RETCODE_OK` shall be selected, setting the `<svcId>Method_<methodName>_Out` structure with all the output and input/output arguments.

* Otherwise, if the operation raised an error, the union case `0x01` shall be selected and the corresponding `ara::core::ErrorCode` constructed (see [FO_PRS_DDS_00302]).

The sample shall then be passed as a parameter to the `write() Method` of the DDS `DataWriter` created in [FO_PRS_DDS_00306] to handle `Method` responses on the server side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

6.10 Handling Fields

[FO_PRS_DDS_00400] Mapping `Field Notifiers` to DDS Topics

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈Every `Field Notifier` of a `ServiceInterface` shall be mapped to a DDS `Topic`. This DDS `Topic` shall be configured as follows:

- The DDS `Topic` name shall be derived according to the following rules:
  - If the `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION` or `SERVICE_INSTANCE_RESOURCE_INSTANCE_ID`, then the DDS `Topic` name shall be set to `ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<fieldTopicName>`
  - Additionally, if the provided or consumed `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, then samples of this DDS `Topic` shall be sent and received via DDS `DataWriter`s and `DataReader`s whose respective parent DDS `Publisher` and `Subscriber` objects include the following partition in the PARTITION QoS policy: `ara.com://services/<svcId>/<svcInId>`
  - Finally, if the provided or consumed `Service Instance` has been advertised with the `<DDSServiceInstanceResourceIdentifierType>` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set to `ara.com://services/<svcId>/<svcInId>/<fieldTopicName>`
  - Where:

    `<svcId>` is the value of `<DDSServiceInterfaceID>` for the `Service Interface`.

    `<svcInId>` is the stringified value of `<DDSServiceInstanceID>` for the `Service Instance`.

    `<svcMajVersion>` is the stringified value of `<DDSInterfaceMajorVersion>` for the `Service Interface`.

    `<svcMinVersion>` is the stringified value of `<DDSInterfaceMinorVersion>` for the `Service Interface`.

    `<fieldTopicName>` is the value of `<DDSEventTopicName>` for the `Service Interface Event`.
- The Data Type of the DDS `Topic` shall be defined as specified in [FO_PRS_DDS_00101], and shall be registered under the equivalent data type name.

⌋

[FO_PRS_DDS_00401] `Field Notifier` DDS `Topic` data type definition

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of a DDS `Topic` representing a `Field Notifier` shall be constructed according to the following IDL definition:

```
struct <FieldTypeName>FieldNotifierType {
    @key uint16 instance_id;
    <FieldTypeName> data;
};
```

Where:

`<FieldTypeName>` is the symbol defined for the `Implementation Data Type`.

`instance_id` is a `@key` member of the type, which identifies all samples with the same `instance_id` as samples of the same DDS `Topic` Instance.

`data` is the actual value of the `Field` Notification, which shall be constructed and encoded according to the DDS serialization rules. The `@external` annotation is optionally allowed, for cases where references yield implementation benefits over values.

⌋

[FO_PRS_DDS_00402] Sending a `Field Notifier` sample

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to send an `Field` Notification sample, a new sample shall be constructed of the equivalent DDS `Topic` data type according to [FO_PRS_DDS_00401].

- The Instance Id field (`instance_id`) shall be set to `<DDSServiceInstanceID>`.
- The Data field (`data`) shall be set to the data to be sent.

This sample shall be then passed as a parameter to the `write()` function of the DDS `DataWriter` associated with the `Field Notifier`, which shall serialize the sample according to the serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

[FO_PRS_DDS_00403] Subscribing to a `Field Notifier`

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to subscribe to a `Field` Notification with a given cache size, a DDS `DataReader` (see [FO_PRS_DDS_00404]) shall be created using a DDS `Subscriber` according to [FO_PRS_DDSSD_00101], [FO_PRS_DDSSD_00105], [FO_PRS_DDSSD_00201] and [FO_PRS_DDSSD_00205].⌋

[FO_PRS_DDS_00404] Creating a DDS `DataReader` for `Field Notifier` subscription

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016, FO_RS_Dds_00005

⌈A DDS `DataReader` for the DDS `Topic` associated with the `Field Notifier` of a `Service Interface` (see [FO_PRS_DDS_00400]) shall be created. If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the `Service Instance` it is bound to, the DDS DDS `Subscriber` created in [FO_PRS_DDSSD_00105] or [FO_PRS_DDSSD_00205] shall be used to create the DDS `DataReader`.

The DDS `DataReader` shall be configured as follows:

- `DataReaderQos`: Defines the DDS `QoS Profile` to be used for the DDS `DataReader`, obtained from `<DDSServiceInstanceFieldNotifierQosProfile>`. To configure the `DataReader`'s cache size, the value of the `DataReader`'s HISTORY QoS shall be overridden as follows:
  - `history.kind = KEEP_LAST_HISTORY_QOS`
  - `history.depth = <cache size value>`
- `Listener`: An instance of the DDS `DataReader Listener` concept as per [FO_PRS_DDS_00405].
- `StatusMask`: Shall be set to `STATUS_MASK_NONE`

⌋

[FO_PRS_DDS_00405] Creating a `DataReaderListener` for `Field` subscription

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈The DDS implementation shall define a DDS `DataReader Listener` to handle `Field` notifications when a new sample is received and/or the matched status of the subscription changes following the instructions specified in [FO_PRS_DDS_00105].

The DDS DataReader Listener shall specify that the samples to be handled are of the DDS Topic data type specified in [FO_PRS_DDS_00401].⌋

[FO_PRS_DDS_00406] Unsubscribing from a Field Notifier

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service Field Notifier, the DDS DataReader associated with the Field shall be deleted.⌋

[FO_PRS_DDS_00407] Mapping of Field Get and Set operations to DDS Topics

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈Every Service Interface containing one or more Fields defined to have a Get and/or Set operations(s) shall have an associated set of DDS Topics to offer those Methods, and to enable client applications to invoke them.

In alignment with [FO_PRS_DDS_00300], these DDS Services shall be constructed according to the Basic Service Mapping Profile of the RPC over DDS specification [4]. Thus, every Service Interface containing one or more Fields defined to have a Get and/or Set operations(s) shall prompt the creation of a pair of DDS Topics: a Request DDS Topic and a Reply Topic.

The equivalent DDS Request DDS Topic shall be configured as follows:

- The Request DDS Topic Name shall be derived from the Manifest according to the following rules:
  - If the Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the DDS Topic name shall be set to ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<fieldRequestTopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this DDS Topic shall be sent and received via DDS DataWriters and DataReaders whose respective parent DDS Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<svcId>/<svcInId>
  - Finally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<svcId>/<svcInId>/<fieldRequestTopicName>

  - Where:

    <svcId> is the value of <DDSServiceInterfaceID> for the Service Interface.

    <svcInId> is the stringified value of <DDSServiceInstanceID> for the Service Instance.

    <svcMajVersion> is the stringified value of <DDSInterfaceMajorVersion> for the Service Interface.

    <svcMinVersion> is the stringified value of <DDSInterfaceMinorVersion> for the Service Interface.

    <fieldRequestTopicName> is the value of <DDSFieldRequestTopicName> for the Service Interface Event.

- The Request DDS Topic Data Type shall be defined as specified in [FO_PRS_DDS_00408].

The equivalent DDS Reply Topic shall be configured as follows:

- The Reply DDS Topic Name shall be derived from the Manifest according to the following rules:
  - If the Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the DDS Topic name shall be set to ara.com://services/<svcId>/<svcMajVersion>.<svcMinVersion>/<fieldReplyTopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this DDS Topic shall be sent and received via DDS DataWriters and DataReaders whose respective parent DDS Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<svcId>/<svcInId>
  - Finally, if the provided or consumed Service Instance has been advertised with the <DDSServiceInstanceResourceIdentifierType> attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<svcId>/<svcInId>/<fieldReplyTopicName>
  - Where:

    <svcId> is the value of <DDSServiceInterfaceID> for the Service Interface.

<svcInId> is the stringified value of <DDSServiceInstanceID> for the Service Instance.

<svcMajVersion> is the stringified value of <DDSInterfaceMajorVersion> for the Service Interface.

<svcMinVersion> is the stringified value of <DDSInterfaceMinorVersion> for the Service Interface.

<fieldReplyTopicName> is the value of <DDSFieldReplyTopicName> for the Service Interface Event.

- The Reply DDS Topic Data Type shall be defined as specified in [FO_PRS_DDS_00409].

⌋

[FO_PRS_DDS_00408] Request DDS Topic data type definition for Field Get and Set operations

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈As specified in section 7.5.1.1.6 of [4], the Request DDS Topic data type is a structure composed of a Request Header with meta-data and a Call Structure with data. The IDL definition of the Request DDS Topic data type for the DDS Service handling Field Get and Set operations is the following:

```
struct <svcId>Field_Request {
    dds::rpc::RequestHeader header;
    <svcId>Field_Call data;
};
```

Where:

<svcId> is the value of <DDSServiceInterfaceID>.

dds::rpc::RequestHeader is the standard Request Header defined in section 7.5.1.1.1 of [4].

<svcId>Field_Call is the union that holds the value of the input parameters of the corresponding Methods, according to the rules specified in section 7.5.1.1.6 of [4].

dds::rpc::RequestHeader shall be constructed as specified in section 7.5.1.1.1 of [4]. On top of that, instanceName (a member of the RequestHeader structure that specifies the DDS Service Instance name) shall be set to a string representation of the <DDSServiceInstanceID> value of the Service Instance that provides the Fields (which have Get or Set operations).

<svcId>Field_Call shall be constructed as specified in section 7.5.1.1.6 of [4].

- The name of the union shall be <svcId>Field_Call.

- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type `dds::rpc::UnknownOperation` (defined in section 7.5.1.1.1 of [4]) for unsupported and unknown operations.
- The union shall have one case label for each `Field` defined to have a `Get` operation, and one case label for each `Field` defined to have a `Set` operation, where:
  - The integer value of the case label shall be a 32-bit hash of the `Field` `Get` or `Set` name. That is, `"Get<fieldName>"` and `"Set<fieldName>"`; where `<fieldName>` the value of <DDSServiceInterfaceFieldName>. The DDS Communication Protocol implementation shall compute the hash as specified in section 7.5.1.1.2 of [4]. Representations of the `Service Interface` in OMG IDL [5]shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Field_Get<fieldName>_Hash` or `const int32 <svcId>Field_Set<fieldName>_Hash`) to simplify the representation of the union cases (see below).
  - The member name for the case label shall be `get<fieldName>` for `Get` operations and `set<fieldName>` for `Set` operations.
  - The type for each case label shall be `<svcId>Field_Get<fieldName>_In` for `Get` operations, and `<svcId>Field_Set<fieldName>_In` for `Set` operations, which shall be constructed as specified in section 7.5.1.1.4 of [4] (see below).

The IDL definition of the `<svcId>Field_Call` union is the following:

```
union <svcId>Field_Call switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Field_Get<field0Name>_Hash:
    <svcId>Field_Get<field0Name>_In get<field0Name>;
case <svcId>Field_Set<field0Name>_Hash:
    <svcId>Field_Set<field0Name>_In set<field0Name>;
case <svcId>Field_Get<field1Name>_Hash:
    <svcId>Field_Get<field1Name>_In get<field1Name>;
case <svcId>Field_Set<field1Name>_Hash:
    <svcId>Field_Set<field1Name>_In set<field1Name>;
// ...
case <svcId>Field_Get<fieldNName>_Hash:
    <svcId>Field_Get<fieldNName>_In get<fieldNName>;
case <svcId>Field_Set<fieldNName>_Hash:
    <svcId>Field_Set<fieldNName>_In set<fieldNName>;
};
```

According to 7.5.1.1.4 of [4], `<svcId>Field_Set<FieldName>_In` structures shall contain one field representing the value of `Field` to be set. Conversely, `<svcId>Field_Get<fieldName>_In` shall contain a single member named `dummy` of type `dds::rpc::UnusedMember` (see section 7.5.1.1.1 of [4]) to indicate that the method has no input parameters.

The resulting Request DDS `Topic` data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Field_Call` union, shall be serialized as specified in section 7.4.3.5 of [2].⌋

[FO_PRS_DDS_00409] Reply DDS `Topic` data type definition for `Field Get` and `Set` operations

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈

As specified in section 7.5.1.1.7 of [4], the Reply DDS `Topic` data type is a structure composed of a Reply Header with meta-data and a Return Structure with data. The IDL definition of the Request DDS `Topic` data type for the DDS Service handling `Field Get` and `Set` operations is the following:

```
1 struct <svcId>Field_Reply {
2     dds::rpc::ReplyHeader header;
3     <svcId>Field_Return data;
4 };
```

Where:

`<svcId>` is the value of `<DDSServiceInterfaceID>`.

`dds::rpc::RequestHeader` is the standard Request Header defined in section 7.5.1.1.1 of [4].

`<svcId>Field_Return` is the union that holds the value of the input parameters of the corresponding response, according to the rules specified in section 7.5.1.1.7 of [4].

`dds::rpc::ReplyHeader` shall be constructed as specified in section 7.5.1.1.1 of [4].

`<svcId>Field_Return` shall be constructed as specified in section 7.5.1.1.7 of [4], where:

- The name of the union shall be `<svcId>Field_Return`.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type `dds::rpc::UnknownOperation` (defined in section 7.5.1.1.1 of [4]) for unsupported and unknown operations.
- The union shall have one case label for each `Field` defined to have a `Get` operation, and one case label for each `Field` defined to have a `Set` operation, where:
  - The integer value of the case label shall be a 32-bit hash of the `Field Get` or `Set` operation name. That is, `"Get<fieldName>"` and `"Set <fieldName>"`; where `<fieldName>` the value of `<DDSServiceInterfaceFieldName>`. The DDS Communication Protocol implementation shall

compute the hash as specified in section 7.5.1.1.2 of [4]. Representations of the `Service Interface` in OMG IDL [5]shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Field_Get<fieldName>_Hash` or `const int32 <svcId>Field_Set<fieldName>_Hash`) to simplify the representation of the union cases (see below).

- The member name of the case label shall be `get<fieldName>` for `Get` operations and `set<fieldName>` for `Set` operations.
- The type for each case label shall be `<svcId>Field_Get<fieldName>_Result` for `Get` operations and `<svcId>Field_Set<fieldName>_Result` for `Set` operationss, which shall be constructed as specified in section 7.5.1.1.4 of [4] (see below).

The IDL definition of `<svcId>Field_Return` is the following:

```
union <svcId>Field_Return switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Field_Get<field0Name>_Hash:
    <svcId>Field_Get<field0Name>_Result get<field0Name>;
case <svcId>Field_Set<field0Name>_Hash:
    <svcId>Field_Set<field0Name>_Result set<field0Name>;
case <svcId>Field_Get<field1Name>_Hash:
    <svcId>Field_Get<field1Name>_Result get<field1Name>;
case <svcId>Field_Set<field1Name>_Hash:
    <svcId>Field_Set<field1Name>_Result set<field1Name>;
// ...
case <svcId>Field_Get<fieldNName>_Hash:
    <svcId>Field_Get<fieldNName>_Result get<fieldNName>;
case <svcId>Field_Set<fieldNName>_Hash:
    <svcId>Field_Set<fieldNName>_Result set<fieldNName>;
};
```

`Get` and `Set` operations have the same output parameter. Therefore, in accordance with section 7.5.1.1.5 of [4], both the `<svcId>Field_Get<fieldName>_Result` and `<svcId>Field_Set<fieldName>_Result` unions shall be constructed as follows:

- The union discriminator shall be a 32-bit signed integer.
- The union shall have a case with label `dds::RETCODE_OK` to represent a successful return:
  - The value of `RETCODE_OK` shall be 0, as specified in section 2.3.3 of [3].
  - The successful case shall have a single member named `result_` of type `<svcId>Field_Get<fieldName>_Out` to hold the value to be returned to the `Get` operation, or type `<svcId>Field_Set<FieldName>_Out` to hold the value to be returned to the `Set` operation (see below).

The IDL representation of `<svcId>Field_Get<fieldName>_Result` is the following:

```
union <svcId>Field_Get<fieldName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Field_Get<fieldName>_Out result_;
};
```

Likewise, the IDL representation of `<svcId>Field_Set<fieldName>_Result` is the following:

```
union <svcId>Field_Set<fieldName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Field_Set<fieldName>_Out result_;
};
```

Both types `<svcId>Field_Get<fieldName>_Out` and its counterpart `<svcId>Field_Set<fieldName>_Out` shall map to a structure with a single member named `return_` of the data type defined for the `Field`.

The resulting Reply DDS `Topic` data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Field_Return` union, shall be serialized as specified in section 7.4.3.5 of [2].⌋

[FO_PRS_DDS_00410] Creating a DDS `DataWriter` to handle get/set requests on the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈A DDS `DataWriter` shall be created for the Request DDS `Topic` associated with the `Get` and `Set` operations of the `Service Interface Fields`s (see [FO_PRS_DDS_00408]) upon client instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the `Service Instance` it is bound to, the DDS `Publisher` created in [FO_PRS_DDSSD_00105] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataWriter`.

The DDS `DataWriter` shall be configured as follows:

- `DataWriterQos` shall be set to `<DDSServiceInstanceQosProfile>`.

⌋

[FO_PRS_DDS_00411] Creating a DDS `DataReader` to handle get/set responses on the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈A DDS `DataReader` shall be created for the Reply DDS `Topic` associated with the `Get`s and `Set`s of the `Service Interface Fields`s (see [FO_PRS_DDS_00409]) upon client instantiation.

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure the proxy communicates only with the Service Instance it is bound to, the DDS Subscriber created in [FO_PRS_DDSSD_00105] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") shall be used to create the DDS DataReader.

The DDS DataReader shall be configured as follows:

- DataReaderQos shall be set to <DDSServiceInstanceQosProfile>.

⌋

[FO_PRS_DDS_00412] Creating a DDS DataReader to handle get/set requests on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈A DDS DataReader shall be created for the Request DDS Topic associated with the Get and Set operations of the Service Interface Fields (see [FO_PRS_DDS_00408]) upon server instantiation.

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure the clients communicate only with the Service Instance it is bound to, the DDS Subscriber created in [FO_PRS_DDSSD_00101] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") shall be used to create the DDS DataReader.

The DDS DataReader shall be configured as follows:

- DataReaderQos shall be set to <DDSServiceInstanceQosProfile>.
- Listener and StatusMask shall be set according to the desired Method call processing mode:
  - For an asynchronous call processing mode, Listener shall be set to an instance of the DDS DataReader Listener class specified in [FO_PRS_DDS_00311], and StatusMask shall be set to DATA_AVAILABLE_STATUS.
  - For a synchronous call processing mode, Listener shall remain unset, and StatusMask shall be set to STATUS_MASK_NONE.

⌋

[FO_PRS_DDS_00413] Creating a DDS `DataWriter` to handle get/set responses on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈A DDS `DataWriter` shall be created for the Reply DDS `Topic` associated with the `Get` and `Set` operations of the `Service Interface Fields`s (see [FO_PRS_DDS_00409]) upon server instantiation.

If the provided or consumed `Service Instance` has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the clients communicate only with the `Service Instance` it is bound to, the DDS `Publisher` created in [FO_PRS_DDSSD_00101] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) shall be used to create the DDS `DataWriter`.

The DDS `DataWriter` shall be configured as follows:

- `DataReaderQos` shall be set to `<DDSServiceInstanceQosProfile>`.

⌋

[FO_PRS_DDS_00414] Calling `Get` and `Set` operations associated with a `Field` from the client side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to call the `Get` or `Set` operation associated with a `Field` from the client side, a new sample of the corresponding Request Topic—an instance of the Request DDS `Topic` data type defined in [FO_PRS_DDS_00408]— shall be constructed as follows:

- To initialize the `RequestHeader` object,
  - `requestId` shall be set according to the rules specified in [4].
  - `instanceName` shall be set to the string-representation value of `<DDSServiceInstanceID>` for the remote `Service Instance`.
- To initialize the `<svcId>Field_Call` object, the appropriate union case (as specified in [FO_PRS_DDS_00408], the hash of the `Field Get/Set` operations name is the union discriminator that selects the union case) shall be selected, and the set as follows:
  - If the call corresponds to a `Get` operation, the DDS Communication Protocol shall leave the `dummy` member of the `<svcId>Field_Get<fieldName>_In` structure unset.
  - Else, if the call corresponds to a `Set` operation, the DDS Communication Protocol shall set accordingly the only member of the `<svcId>Field_Set<fieldName>_In` structure with the new value for the `Field`.

That sample shall then be passed as a parameter to the `write() Method` of the DDS `DataWriter` created in [FO_PRS_DDS_00410] to handle get/set requests on the client side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

[FO_PRS_DDS_00415] Notifying the client of the response to the `Get` and `Set` operations call

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈To notify the client application of a response as a result of a call to a `Get` or `Set` operation, the result value shall be extracted from the Reply DDS `Topic` sample, from either the `<svcId>Field_Get<FieldName>_Result` structure, for get operations; or `<svcId>Field_Set<FieldName>_Out`, for set operations.

The associated set operation shall be performed upon the reception of a new Reply Topic sample by the corresponding DDS `DataReader` (see [FO_PRS_DDS_00411]). The DDS `DataReader`'s `take() Method` shall be used to receive sample. Moreover, to correlate a request with a response, the DDS Communication Protocol shall compare the `header.relatedRequestsId` of the received sample with the original `requestId` that was sent in [FO_PRS_DDS_00414][3].

If the `relatedRequestId` does not correspond to a `requestId` that has been sent by the client, the response shall be discarded.⌋

[FO_PRS_DDS_00416] Processing a `Get` and `Set` operations call associated with a `Field` on the server side (asynchronous)

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case asynchronous request processing is expected, a DDS `DataReader Listener` shall be created to process the requests asynchronously—as described in [FO_PRS_DDS_00417]—with an instance of it attached to the DDS `DataReader` processing the requests in accordance with [FO_PRS_DDS_00412]. The listener is responsible for identifying the internal callback processing the request, and dispatching request data to it (see [FO_PRS_DDS_00417]).⌋

[3] See footnotes in [FO_PRS_DDS_00309].

[FO_PRS_DDS_00417] Creating a `DataReaderListener` to process asynchronous requests for `Field` `Get` and `Set` operations on the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈According to [FO_PRS_DDS_00412], asynchronous request processing requires the instantiation of a DDS `DataReader` `Listener` to process asynchronously requests on the server side. This object shall handle samples of the DDS `Topic` data type specified in [FO_PRS_DDS_00408].

The DDS `DataReader` `Listener` shall implement the following callback according to the specified instructions:

- An `on_data_available()` callback responsible for reading the received requests from the DDS `DataReader`'s cache—using the `take()` operation—and dispatching it to the upper platform layers. To identify the `Field` of the `Service Instance`, and the callback to be invoked for it, `on_data_available()` shall use the union discriminator of the `<svcId>Field_Call` union (see [FO_PRS_DDS_00408]).
  - In the case of a `Set` operation, the only member of the received `<svcId>Field_<FieldName>_In` structure, which contains the new value to be set, shall be passed to the callback.
  - In the case of a `Get` operation, the callback shall provide the intended result of the `Get` operation.

⌋

[FO_PRS_DDS_00418] Processing a `Get` and `Set` operations call associated with a `Field` on the server side (synchronous)

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008

⌈In case synchronous request processing is expected, the protocol implementaiton is responsible for calling `take()` on the DDS `DataReader` processing the Request DDS `Topic` associated with the service (see [FO_PRS_DDS_00408]).

Each synchronous operation shall `take()` only the first sample from the DDS `DataReader`'s cache and dispatch the call to the appropriate callback according to the value of the of the discriminator of the `<svcId>Field_Call` union and provide the destination callback with the specific arguments in the union case.⌋

[FO_PRS_DDS_00419] Sending a response for a `Get` and `Set` operations call associated with a `Field` from the server side

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈Upon the return of a `Service Instance` `Field` `Get` or `Set` callback, a response shall be sent to the client.

To send this response, a new sample of the Reply DDS `Topic` —an instance of the Reply DDS `Topic` data type defined in [FO_PRS_DDS_00409])— shall be constructed as follows:

- To initialize the `ReplyHeader` object,
  - `relatedRequestId` shall be set to the value of the `header.requestId` attribute of the request that triggered the `Method` call (see [FO_PRS_DDS_00414]).
- To initialize the `<svcId>Field_Return` object:
  - Select the appropriate union case (as specified in [FO_PRS_DDS_00409]), the hash of the `Field`'s `Get`/`Set` operation is the union discriminator that selects the union case).
  - Set the appropriate `<svcId>Field_Get<FieldName>_Result`—for `Get` operations—or `<svcId>Field_Set<FieldName>_Result`—for `Set` operations—. In both cases, the union case for `dds::RETCODE_OK` shall be selected and the corresponding structure be set with the value retrieved upon the return of the `Field` `Get` or `Set` callback.

The sample shall then be passed as a parameter to the `write()` function of the DDS `DataWriter` created in [FO_PRS_DDS_00413] to handle `Get` and `Set` operation responses on the server side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋

The DDS serialization rules are defined in section 6.11.

6.11 Serialization of Payload

The present section outlines **generic** data type mappings, which AUTOSAR platforms then specialize for their own native type system.

[FO_PRS_DDS_00500] DDS standard serialization rules

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002

⌈The serialization of the payload shall be done according to the DDS standard serialization rules defined in section 7.4.3.5 of [2].⌋

6.11.1 Basic Data Types

[FO_PRS_DDS_00501] DDS serialization of primitive data types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Primitive data types shall be serialized according to the standard serialization rules for the equivalent `PRIMITIVE_TYPE` defined in section 7.4.3.5 of [2], as mapped by [FO_PRS_DDS_00510].⌋

[FO_PRS_DDS_00510] Mapping of DDS primitive data types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈

Type	DDS Type	Remark
Boolean	Boolean	
Unsigned 8-bit integer	Byte	Shall be encoded as a Byte type (opaque 8-bit type).
Unsigned 16-bit integer	UInt16	
Unsigned 32-bit integer	UInt32	
Unsigned 64-bit integer	UInt64	
Signed 8-bit integer	Byte	Shall be encoded as a Byte type (opaque 8-bit type).
Signed 16-bit integer	Int16	
Signed 32-bit integer	Int32	
Signed 64-bit integer	Int64	
32-bit floating-point decimal	Float32	

64-bit floating-point decimal	Float64	

⌋

6.11.2 Enumeration Data Types

[FO_PRS_DDS_00502] DDS serialization of enumeration data types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Enumeration data types shall be serialized according to the standard serialization rules for `ENUM_TYPE` defined in section 7.4.3.5 of [2].

The bit bound of the `ENUM_TYPE` shall be set to the size of the enumeration's underlying basic data type in bits.⌋

6.11.3 Structured Data Types (structs)

[FO_PRS_DDS_00503] DDS serialization of structure data types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Structure data types (also known as *records*) shall be serialized according to the standard serialization rules for `STRUCT_TYPE` defined in section 7.4.3.5 of [2].

Optional members of the structure shall be marked as optional as specified in section 7.2.2.4.4.5 of [2].⌋

6.11.4 Strings

[FO_PRS_DDS_00504] DDS serialization of string types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈String data types shall be serialized according to the standard serialization rules for `STRING_TYPE` defined in section 7.4.3.5 of [2].⌋

[FO_PRS_DDS_00505] Encoding Format and Endianness of Strings in DDS

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Section 7.4.1.1.2 of [2] specifies the standard character encoding format for `STRING_TYPE`: UTF-8. The serialized version shall not include a Byte Order Mark (BOM), as byte order information is already available in the RTPS Encapsulation Identifier and the XCDR serialization format [2].⌋

6.11.5 Vectors and Arrays

[FO_PRS_DDS_00506] DDS serialization of vector types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Vector (also known as *variable-length contiguous sequence*) types shall be serialized according to the standard serialization rules for `SEQUENCE_TYPE` defined in section 7.4.3.5 of [2].

DDS Communication Protocol implementations shall serialize vector types with more than one dimension, as nested DDS sequences.⌋

[FO_PRS_DDS_00507] DDS serialization of array types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Array (also known as *fixed-length contiguous sequence*) types shall be serialized according to the standard serialization rules for `ARRAY_TYPE` defined in section 7.4.3.5 of [2].⌋

6.11.6 Associative Maps

[FO_PRS_DDS_00508] DDS serialization of dictionary types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Dictionary (also known as *associative map*) types shall be serialized according to the standard serialization rules for `MAP_TYPE` defined in section 7.4.3.5 of [2].⌋

6.11.7 Variant

[FO_PRS_DDS_00509] DDS serialization of variant types

Status: DRAFT
Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Variant types shall be serialized according to the standard serialization rules for `UNION_TYPE` defined in section 7.4.3.5 of [2].⌋

6.12 End-to-end communication protection

The present DDS Communication Protocol is defined in terms of standard DDS types, QoS policies and APIs. Hence, End-to-end communication protection as described for other protocols doesn't apply, because API calls can't be checksummed or payloaded the same way serialized messages are.

By no means does this imply that DDS is exempt from E2E protection assurances, they are simply provided by the DDS middleware. Different kinds of faults defined in [6] (derived from ISO-26262-6:2011, annex D.2.4) and their corresponding DDS/RTPS protection mechanism are described by the following items.

[FO_PRS_DDS_00601] Repetition or Insertion of Information

Status: DRAFT
Upstream requirements: FO_RS_Dds_00010

⌈Submessage 64-bit sequence number, as defined in [1] section 8.3.5.4 "SequenceNumber", and additional SequenceNumber-typed fields in section 8.3.7 "RTPS Submessages" shall be used to guarantee safety mechanisms against Repetition or Insertion of Information faults.

Those mechanisms can be useful only to detect losses at receiver side; if detection is required also to sender side, the `RELIABILITY` DDS QoS policy (defined in [1], section 2.2.3.14 "RELIABILITY") shall be used in conjuction.

At receiving side, if a message with a duplicated counter is received, the message shall be discarded and the fault reported to upper platform layers.⌋

[FO_PRS_DDS_00602] Loss or Incorrect sequence of Information

Status: DRAFT
Upstream requirements: FO_RS_Dds_00010

⌈Submessage 64-bit sequence number, as defined in [1] section 8.3.5.4 "SequenceNumber", and additional SequenceNumber-typed fields in section 8.3.7 "RTPS Submessages" shall be used to guarantee safety mechanisms against Loss or Incorrect sequence of Information faults.

At receiving side, if a message with a non-consecutive counter is received, the message shall be discarded and the fault reported to upper platform layers.⌋

[FO_PRS_DDS_00603] Delay of Information or Blocking Access to a Communication Channel

Status: DRAFT
Upstream requirements: FO_RS_Dds_00005, FO_RS_Dds_00010

⌈DDS QoS policies shall be used to monitor Delay of Information faults, such as `DEADLINE`, `LATENCY_BUDGET`, `LIFESPAN` and `LIVELINESS` (refer to [3] for details on those QoS policies).

If timing constraints are not fulfilled on either receiver or sender side, the fault shall be reported to upper platform layers.⌋

[FO_PRS_DDS_00604] Corruption of Information

Status: DRAFT
Upstream requirements: FO_RS_Dds_00010

⌈`rtpsMessageChecksum` under `HeaderExtension` submessage (RTPS 2.5 or higher) shall be used to guarantee safety mechanisms against Corruption of Information faults.

Those mechanisms can be useful only to detect corruption at receiver side.

At receiving side, if a message with an invalid checksum is received, the message shall be discarded and the fault reported to upper platform layers.⌋

7 Configuration parameters

This chapter lists all parameters the DDS Communication protocol uses.

7.1 Service Oriented Communication

DDS Protocol Parameter	Description	AP Config	CP Config
<DDSServiceInterfaceID>	ID of the DDS Service Interface	DdsServiceInterfaceDeployment.serviceInterfaceId	-
<DDSServiceInstanceID>	ID of the DDS Service Instance	DdsProvidedServiceInstance.serviceInstanceId and DdsRequiredServiceInstance.requiredServiceInstanceId	-
<DDSInterfaceMajorVersion>	Major Version of the DDS Service Interface	ServiceInterface.majorVersion	-
<DDSInterfaceMinorVersion>	Minor Version of the DDS Service Interface	ServiceInterface.minorVersion	-
<DDSServiceInstanceResourceIdentifierType>	Resource Identification scheme for DDS Service Instance	DdsProvidedServiceInstance.resourceIdentifierType	-
<DDSEventTopicName>	Suffix of the DDS Topic name for an Event within a DDS Service Interface	DdsEventDeployment.topicName	-
<DDSServiceInstanceEventQoSProfile>	QoS Profile for an Event within a specific DDS Service Instance	DdsQosProps.qosProfile	-
<DDSTriggerTopicName>	Suffix of the DDS Topic name for an Trigger within a DDS Service Interface	DdsEventDeployment.topicName	-
<DDSServiceInstanceTriggerQoSProfile>	QoS Profile for an Trigger within a specific DDS Service Instance	DdsQosProps.qosProfile	-
<DDSMethodRequestTopicName>	Suffix of the DDS Topic name conveying all Service Instance Method Requests of a DDS Service Interface	DdsServiceInterfaceDeployment.methodRequestTopicName	-
<DDSMethodReplyTopicName>	Suffix of the DDS Topic name conveying all Service Instance Method Replies of a DDS Service Interface	DdsServiceInterfaceDeployment.methodReplyTopicName	-
<DDSServiceInterfaceMethodName>	Name of a Service Interface Method	ClientServerOperation.shortName	-
<DDSServiceInterfaceMethodErrorDomain>	Error Domain of an Application-layer Error	ApApplicationErrorDomain.value	-
<DDSServiceInterfaceMethodErrorCode>	Error Code of an Application-layer Error	ApApplicationError.errorCode	-
<DDSServiceInstanceQosProfile>	QoS Profile for all Methods and Field Methods within a specific DDS Service Instance	DdsRequiredServiceInstance.qosProfile	-
<DDSServiceInstanceFieldNotifierQosProfile>	QoS Profile for a Field Notifier within a specific DDS Service Instance	DdsQosProps.qosProfile	-
<DDSFieldRequestTopicName>	Suffix of the DDS Topic name conveying all Service Instance Field Requests of a DDS Service Interface	DdsServiceInterfaceDeployment.fieldRequestTopicName	-

▽

△

<DDSFieldReplyTopicName>	Suffix of the DDS Topic name conveying all Service Instance Field Replies of a DDS Service Interface	DdsServiceInterfaceDeployment.fieldReplyTopicName	-
<DDSServiceInterfaceFieldName>	Name of a Service Interface Field	ClientServerOperation.shortName	-

Table 7.1: Mapping Table - DDS Protocol Parameters

8 Protocol usage and guidelines

This section is intentionally left empty.

A Appendix

This section is intentionally left empty.

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R24-11

B.1.1 Added Specification Items in R24-11

Number	Heading
[FO_PRS_DDS_-00100]	Mapping `Event`s to DDS Topics
[FO_PRS_DDS_-00101]	DDS `Topic` data type definition
[FO_PRS_DDS_-00102]	Sending an `Event` sample
[FO_PRS_DDS_-00103]	Subscribing to an `Event`
[FO_PRS_DDS_-00104]	Creating a DDS `DataReader` for `Event` subscription
[FO_PRS_DDS_-00105]	Defining a DDS `DataReader` `Listener` for a subscribed `Event`
[FO_PRS_DDS_-00106]	Unsubscribing from an `Event`
[FO_PRS_DDS_-00107]	Obtaining subscription state from an `Event`
[FO_PRS_DDS_-00108]	Retrieving new data samples from an `Event`
[FO_PRS_DDS_-00109]	Requesting number of free sample slots from an `Event`
[FO_PRS_DDS_-00110]	Registering an `Event` reception handler
[FO_PRS_DDS_-00111]	Unregistering an `Event` reception handler
[FO_PRS_DDS_-00112]	Registering an `Event` subscription state change handler
[FO_PRS_DDS_-00113]	Unregistering an `Event` subscription state change handler
[FO_PRS_DDS_-00200]	Mapping `Trigger`s to DDS Topics

△

Number	Heading
[FO_PRS_DDS_-00201]	DDS `Topic` data type definition
[FO_PRS_DDS_-00202]	Sending a `Trigger` sample
[FO_PRS_DDS_-00203]	Subscribing to a `Trigger`
[FO_PRS_DDS_-00204]	Creating a DDS `DataReader` for `Trigger` subscription
[FO_PRS_DDS_-00205]	Defining a DDS `DataReader` `Listener` for a subscribed `Trigger`
[FO_PRS_DDS_-00206]	Unsubscribing from an `Trigger`
[FO_PRS_DDS_-00207]	Obtaining subscription state from a `Trigger`
[FO_PRS_DDS_-00208]	Retrieving new notification from a `Trigger`
[FO_PRS_DDS_-00209]	Registering a `Trigger` reception handler
[FO_PRS_DDS_-00210]	Unregistering a `Trigger` reception handler
[FO_PRS_DDS_-00211]	Registering a `Trigger` subscription state change handler
[FO_PRS_DDS_-00212]	Unregistering a `Trigger` subscription state change handler
[FO_PRS_DDS_-00300]	Mapping `Method`s to DDS `Topic`s
[FO_PRS_DDS_-00301]	Mapping `Method`s to DDS `Topic` request data type definition
[FO_PRS_DDS_-00302]	Mapping `Method`s to DDS `Topic` reply data type definition
[FO_PRS_DDS_-00303]	Mapping of ara::core::ErrorCode
[FO_PRS_DDS_-00304]	Creating a DDS `DataWriter` to handle `Method` requests on the client side
[FO_PRS_DDS_-00305]	Creating a DDS `DataReader` to handle `Method` responses on the client side
[FO_PRS_DDS_-00306]	Creating a DDS `DataReader` to handle `Method` requests on the server side
[FO_PRS_DDS_-00307]	Creating a DDS `DataWriter` to handle `Method` responses on the server side
[FO_PRS_DDS_-00308]	Calling a service `Method` from the client side
[FO_PRS_DDS_-00309]	Notifying the client of a response to a `Method` call

▽

△

Number	Heading
[FO_PRS_DDS_-00310]	Processing a `Method` call on the server side (asynchronous)
[FO_PRS_DDS_-00311]	Creating a DDS `DataReader` Listener to process asynchronous requests on the server side
[FO_PRS_DDS_-00312]	Processing a `Method` call on the server side (synchronous)
[FO_PRS_DDS_-00313]	Sending a `Method` call response from the server side
[FO_PRS_DDS_-00400]	Mapping `Field Notifier`s to DDS Topics
[FO_PRS_DDS_-00401]	`Field Notifier` DDS `Topic` data type definition
[FO_PRS_DDS_-00402]	Sending a `Field Notifier` sample
[FO_PRS_DDS_-00403]	Subscribing to a `Field Notifier`
[FO_PRS_DDS_-00404]	Creating a DDS `DataReader` for `Field Notifier` subscription
[FO_PRS_DDS_-00405]	Creating a `DataReaderListener` for `Field` subscription
[FO_PRS_DDS_-00406]	Unsubscribing from a `Field Notifier`
[FO_PRS_DDS_-00407]	Mapping of `Field Get` and `Set` operations to DDS `Topic`s
[FO_PRS_DDS_-00408]	Request DDS `Topic` data type definition for `Field Get` and `Set` operations
[FO_PRS_DDS_-00409]	Reply DDS `Topic` data type definition for `Field Get` and `Set` operations
[FO_PRS_DDS_-00410]	Creating a DDS `DataWriter` to handle get/set requests on the client side
[FO_PRS_DDS_-00411]	Creating a DDS `DataReader` to handle get/set responses on the client side
[FO_PRS_DDS_-00412]	Creating a DDS `DataReader` to handle get/set requests on the server side
[FO_PRS_DDS_-00413]	Creating a DDS `DataWriter` to handle get/set responses on the server side
[FO_PRS_DDS_-00414]	Calling `Get` and `Set` operations associated with a `Field` from the client side
[FO_PRS_DDS_-00415]	Notifying the client of the response to the `Get` and `Set` operations call
[FO_PRS_DDS_-00416]	Processing a `Get` and `Set` operations call associated with a `Field` on the server side (asynchronous)
[FO_PRS_DDS_-00417]	Creating a `DataReaderListener` to process asynchronous requests for `Field Get` and `Set` operations on the server side

▽

△

Number	Heading
[FO_PRS_DDS_-00418]	Processing a `Get` and `Set` operations call associated with a `Field` on the server side (synchronous)
[FO_PRS_DDS_-00419]	Sending a response for a `Get` and `Set` operations call associated with a `Field` from the server side
[FO_PRS_DDS_-00500]	DDS standard serialization rules
[FO_PRS_DDS_-00501]	DDS serialization of primitive data types
[FO_PRS_DDS_-00502]	DDS serialization of enumeration data types
[FO_PRS_DDS_-00503]	DDS serialization of structure data types
[FO_PRS_DDS_-00504]	DDS serialization of string types
[FO_PRS_DDS_-00505]	Encoding Format and Endianness of Strings in DDS
[FO_PRS_DDS_-00506]	DDS serialization of vector types
[FO_PRS_DDS_-00507]	DDS serialization of array types
[FO_PRS_DDS_-00508]	DDS serialization of dictionary types
[FO_PRS_DDS_-00509]	DDS serialization of variant types
[FO_PRS_DDS_-00510]	Mapping of DDS primitive data types
[FO_PRS_DDS_-00601]	Repetition or Insertion of Information
[FO_PRS_DDS_-00602]	Loss or Incorrect sequence of Information
[FO_PRS_DDS_-00603]	Delay of Information or Blocking Access to a Communication Channel
[FO_PRS_DDS_-00604]	Corruption of Information

Table B.1: Added Specification Items in R24-11

B.1.2 Changed Specification Items in R24-11

none

B.1.3 Deleted Specification Items in R24-11

none