Date: March 2017

OBJECT MANAGEMENT GROUP

DDS Consolidated XML Syntax

Version 1.0

OMG Document Number: mars/2017-03-01
Standard document URL:

IPR mode: ***Non-Assert***

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

OMG's Issue Reporting Procedure

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

Table of Contents

Tables

Figures

Preface

About the Object Management Group

OMG

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

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

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

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. All OMG specifications are available from the OMG website at: *http://www.omg.org/spec*

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

OMG Headquarters 109 Highland Avenue Needham, MA 02494 USA
Tel: +1-781-444-0404 Fax: +1-781-444-0320 Email: *pubs@omg.org*

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

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents new terms or concepts or the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/report issue.htm*.

1 Scope

Historically various specifications have defined XML syntax to represent particular subsets of DDS-related resources:

- The [DDS4CCM] specification defines XML syntax to represent the DDS QoS Policies of DDS Entities: DomainParticipantQos, TopicQos, PublisherQos, SubscriberQos, DataWriterQos, and DataReaderQos.
- The [DDS-XTYPES] specification defines XML syntax to represent DDS Data Types and Data Samples.
- The [DDS-WEB] specification defines XML syntax to represent DDS Applications, DDS Domains, and DDS entities (i.e., DomainParticipant, Topic, Publisher, Subscriber, DataWriter, and DataReader).

This specification consolidates all this XML syntax into a single document. There are no significant syntactic changes in this document relative to referenced specifications.

2 Conformance Criteria

This document contains no independent conformance points. Rather, it defines XML Schemas to be used to describe DDS resources such that they can be referenced by other specifications leaving the definition of conformance criteria to the referencing specifications. Nevertheless, the general organization of the clauses (by means of atomic building blocks and building block sets that group them) is intended to ease conformance description and scoping.

Use of this standard must follow these rules:

1. Future specifications that describe DDS resources in XML shall reference this specification or a future revision thereof.
2. Future revisions of current specifications that describe DDS resources in XML should reference this specification or a future revision thereof. Reference to this standard shall result in a selection of building blocks where all selected building blocks shall be supported entirely.

3 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- [XML] Extensible Markup Language (XML) 1.0 (Fifth Edition) Specification. Available from: https://www.w3.org/TR/2008/REC-xml-20081126.
- [XSD-1] XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Available from: https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405.
- [XSD-2] XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. Available from: https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.

The following referenced documents were used as input to this specification:

- [DDS4CCM] DDS for Lightweight CCM (DDS4CCM), Version 1.1. OMG Document: formal/2012-02-01. Available from: http://www.omg.org/spec/dds4ccm/1.1.
- [DDS-XTYPES] Extensible and Dynamic Topic Types for DDS, Version 1.1. OMG Document: formal/2014-11-03. Available from: http://www.omg.org/spec/DDS-XTypes/1.1.

- [DDS-WEB] Web-Enabled DDS, Version 1.0. OMG Document: formal/2016-03-01. Available from: http://www.omg.org/spec/DDS-WEB/1.0.

- [DDS] Data Distribution Service, Version 1.4. OMG Document: formal/15-04-10. Available from: http://www.omg.org/spec/DDS/1.4.

- [IDL] Interface Definition Language (IDL), Version 4.0. OMG Document: formal/2016-04-02. Available from: http://www.omg.org/spec/IDL/4.0.

4 Terms and Definitions

In this specification:

- A *building block* is a consistent set of XML schemas that together can be used to describe the syntax of XML documents that represent a set of set of DDS resources. Building blocks are atomic, which means that if selected they must be totally supported. Building blocks are described in clause 7, XML Syntax for DDS Resources.

- A *building block set* is a selection of building blocks that determines a specific XSD schema usage. Building block sets are described in clause 8.

5 Symbols

The following acronyms are used in this specification are show in Table 5.1.

Table 5.1 Acronyms

Acronym	Meaning
DDS	Data Distribution Service
ISO	International Organization for Standardization
LwCCM	Lightweight CCM
OMG	Object Management Group
QoS	Quality of Service
UTF	Unicode Transformation Format
XML	Extensible Markup Language
XSD	XML Schema Definition
XTypes	eXtensible and dynamic topic Types (for DDS)

6 Additional Information

6.1 Changes to Adopted OMG Specifications

This specification does not change any adopted OMG specification.

6.2 Acknowledgments

The following companies submitted this specification:

- Real-Time Innovations, Inc.
- Twin Oaks Computing, Inc.
- Jackrabbit Consulting

7 XML Syntax for DDS Resources

7.1 XML Representation Syntax

7.1.1 General Rules

The XML representation of DDS-related resources must follow these syntax rules:

- It shall be a well-formed XML document according to the criteria defined in clause 2.1 of [XML].
- It shall use UTF-8 character encoding for XML elements and values.
- It shall use `<dds>` as the root tag of every document.

7.1.2 XML Schema Definition Files

This specification makes use of XML Schema Definition (XSD) language specified in [XSD-1] and [XSD-2] to represent the syntax of the different building blocks that define DDS resources. In particular, each building block contains two normative XSD files that define their syntax (see sub clause 7.3.1).

7.1.3 XML Chameleon Schema Definition Pattern (non-normative)

7.1.3.1 Motivation

XSD provides namespaces to scope the name of all the different elements, attributes, and types defined in a schema file. This is especially useful in projects that need to combine schema files that are designed by different organizations, as they prevent most of the naming conflicts that may arise in this kind of scenarios.

However, combining schema files with multiple namespaces presents some complications. End users of the XML file need to be aware of all the different namespaces defined by the schemas that are imported, and specify them in the XML document by either using the `xmlns` attribute in each tag or using different qualified names.

To provide a reusable set of building blocks avoid the problems described above, the normative schema files in this specification follow a well-known pattern for designing XSD files: the XML Chameleon Schema Definition pattern.

In this pattern, XML elements, attributes, and types are specified in an XSD file that defines no namespace. Schema files that define no namespace are often referred to as chameleon schemas, because they take the namespace (i.e., the color) of the schema that includes them. As a result, the definitions in the chameleon schema can be easily imported in other XSD files, which helps define XML files that do not require handling different namespaces in different tags.

7.1.3.2 Example

To illustrate this scenario, let us use as an example a DDS application that uses XML files to configure QoS settings, Data Types, and other vendor-specific parameters.

7.1.3.2.1 Using Schema Files with Different Namespaces

The standard QoS parameters are defined in a normative XSD included in [DDS4CCM], and the XML syntax for declaring types is defined in a normative XSD included in [DDS-XTYPES]. Therefore, the schema file that defines the syntax of the XML files that configure the application includes the standard schema files associated with these specifications using the `<xs:import>` mechanism defined in [XSD-1].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example-vendor.org"
           targetNamespace="http://www.example-vendor.org"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
    ...
    <xs:import namespace="http://www.omg.org/dds/"
         schemaLocation="http://www.omg.org/spec/dds4ccm/20110201/DDS_QoSProfile.xsd"/>
    <xs:import namespace="http://www.omg.org/ptc/2011/01/07/XML_Type_Representation"
         schemaLocation="http://www.omg.org/spec/DDS-XTypes/20120202/dds-
xtypes_type_definition.xsd" />
    ...
</xs:schema>
```

The resulting XSD above uses three different namespaces: `http://www.example-vendor.org`, which is the vendor-specific namespace; `http://www.omg.org/dds/`, which is the namespace of the XSD defined in [DDS4CCM]; and `http://www.omg.org/ptc/2011/01/07/XML_Type_Representation`, which is the namespace of the XSD defined in [DDS-XTYPES].

Consequently, the elements need to be identified with the appropriate namespace in the XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application_cfg
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://www.example-vendor.org"
         xsi:schemaLocation="http://www.example-vendor.org vendor_schema.xsd">

    <my_configuration name="ExampleConfiguration">
        <!-- Declare types using the namespace defined in [DDS-XTYPES] -->
        <types xmlns="http://www.omg.org/ptc/2011/01/07/XML_Type_Representation">
            <struct name="ShapeType">
                <member name="x" type="long" />
            </struct>
        </types>

        <!-- Configure entity QoS settings using the namespace defined in
             [DDS4CCM] -->
        <datareader_qos xmlns="http://www.omg.org/dds/">
            <reliability>
                <kind>RELIABLE_RELIABILITY</kind>
            </reliability>
        </dateader_qos>
```

```
            <!-- Vendor specific configuration settings use the default
                 namespace for the document indicated via the xmlns attribute
                 to the root tag <application_cfg> -->
            <vendor_specific_settings>
                ...
            </vendor_specific_settings>
        </my_configuration>
</application_cfg>
```

7.1.3.2.2 Chameleon Schema Files

Following the example defined above, if the XSD files in [DDS4CCM] and [DDS-XTYPES] defined no `targetNamespace`, the different elements they define would automatically take the `http://www.example-vendor.org` namespace upon their inclusion. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example-vendor.org"
           targetNamespace="http://www.example-vendor.org"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
    ...
    <!-- Note that include does not prepend the namespace anymore -->
    <xs:include
schemaLocation="http://www.omg.org/spec/dds4ccm/20110201/DDS_QoSProfile.xsd"/>
    <xs:include schemaLocation="http://www.omg.org/spec/DDS-XTypes/20120202/dds-
xtypes_type_definition.xsd"/>
    ...
</xs:schema>
```

Consequently, an XML file including QoS settings and Types in this new context would only need to declare the vendor-specific namespace using the `xmlns` attribute in the top-level `<application_cfg>` tag. That is:

```
<application_cfg
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://www.example-vendor.org"
         xsi:schemaLocation="http://www.example-vendor.org vendor_schema.xsd">

    <my_configuration name="ExampleConfiguration">
        <!-- No namespace for types and datareader_qos -->
        <types>
            ...
        </types>
        <datareader_qos>
            ...
```

```
        </datareader_qos>
    </my_configuration>
</application_cfg>
```

The specific use of the *Chameleon Schema Definition* pattern for the different building blocks of this specification is described in detail in sub clause 7.3.1.

7.1.4 XML Element Values

The primitive types for XML Element values are specified in Table 7.1.

Table 7.1 Supported XML Element Values

Type	Format	Notes
boolean	Yes: `1` or `true`.	Values are case sensitive.
	No: `0` or `false`.	Values are case sensitive.
enum	A string. Legal values are the ones defined for QoS Policies in the DCPS IDL of DDS specification [DDS].	Must be specified as a string. (Do not use numeric values.)
long	`-2147483648` to `2147483647` or `0x80000000` to `0x7fffffff` or `LENGTH_UNLIMITED`.	A 32-bit signed integer.
unsigned long	`0` to `4294967296` or `0` to `0xffffffff`.	A 32-bit unsigned integer.
string	The string with the reserved XML characters escaped according to the standard rules for element content [XML].	Per the XML rules only < and `&` are required to be escaped within an element content. The characters >, `'`, and `"` may be escaped.

7.1.5 XML Attribute Values

The primitive types for XML Attribute values are specified in Table 7.2.

Table 7.2 Supported XML Element Values

Type	Format	Notes
boolean	Yes: `1` or `true`.	Values are case sensitive.
	No: `0` or `false`.	Values are case sensitive.
enum	A string. Legal values are the ones defined for QoS Policies in the DCPS IDL of DDS specification [DDS].	Must be specified as a string. (Do not use numeric values.)
long	`-2147483648` to `2147483647` or `0x80000000` to `0x7fffffff` or `LENGTH_UNLIMITED`.	A 32-bit signed integer.
unsigned long	`0` to `4294967296` or `0` to `0xffffffff`.	A 32-bit unsigned integer.
string	The string with the reserved XML characters escaped according to the standard rules for element content [XML].	

7.2 XML Representation of Resources Defined in the DDS IDL PSM

The XML representation of resources that correspond to data-types defined in the DDS IDL PSM [DDS] is obtained by performing a 1-to-1 mapping of the corresponding IDL data type.

7.2.1 XML Representation of Enumeration types

IDL Enumerations are represented in XML according to a schema defined as an XSD `simpleType` defined as a restriction of a string that can take values of the enumeration literals.

7.2.1.1 Example (Non-normative)

For example, the **HistoryQosPolicyKind** is defined in the DDS IDL PSM as:

```
enum HistoryQosPolicyKind {
   KEEP_LAST_HISTORY_QOS,
   KEEP_ALL_HISTORY_QOS
};
```

The equivalent representation in XML is defined by the XSD `historyQosPolicyKind` below:

```
<xs:simpleType name="historyKind">
    <xs:restriction base="xs:string">
        <xs:enumeration value="KEEP_LAST_HISTORY_QOS" />
        <xs:enumeration value="KEEP_ALL_HISTORY_QOS" />
    </xs:restriction>
</xs:simpleType>
<xs:element name="kind" type="historyKind" />
```

An example XML resource representation satisfying this syntax would be:

```
<kind>KEEP_ALL_HISTORY_QOS</kind>
```

7.2.2 XML Representation of Primitive Constants

The DDS IDL PSM defines constant values of type **long** and **string**. These are intended as pre-defined values that can be used to initialize members of certain structured types.

These constant definitions appear in the XSD as `simpleType` schemas defined as restrictions of `xs:string` to provide custom syntax allowing an element to have a value that is either given as a number or as the name of the constant.

7.2.2.1 Example (Non-Normative)

For example, the IDL defines the constants:

```
const long LENGTH_UNLIMITED = -1;
const long DURATION_INFINITE_SEC   = 0x7fffffff;
const unsigned long DURATION_INFINITE_NSEC  = 0x7fffffff;
const long DURATION_ZERO_SEC = 0;
const unsigned long DURATION_ZERO_NSEC = 0;
const long TIME_INVALID_SEC = -1;
const unsigned long TIME_INVALID_NSEC = 0xffffffff;
```

The constant **LENGTH_UNLIMITED** is intended to initialize structure members that represent lengths. The constants with the prefix **DURATION_** are intended to initialize the members (**second** and **nanosecond**) of the structure **Duration_t**,

and the constants with the prefix **TIME_** are intended to initialize the members (**second** and **nanosecond**) of the structure **Time_t**.

For example, the above constants are mapped into the schema types:

```
<xs:simpleType name="nonNegativeInteger_UNLIMITED">
    <xs:restriction base="xs:string">
        <xs:pattern value="(LENGTH_UNLIMITED|([0-9])*)?" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="positiveInteger_UNLIMITED">
    <xs:restriction base="xs:string">
        <xs:pattern value="(LENGTH_UNLIMITED|[1-9]([0-9])*)?" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger_Duration_SEC">
    <xs:restriction base="xs:string">
        <xs:pattern
                value="(DURATION_INFINITY|DURATION_INFINITE_SEC|([0-9])*)?" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger_Duration_NSEC">
    <xs:restriction base="xs:string">
        <xs:pattern
                value="(DURATION_INFINITY|DURATION_INFINITE_NSEC|([0-9])*)?" />
    </xs:restriction>
</xs:simpleType>
```

See clause 7.2.6 for a description of how they are used in the schemas for **Duration_t**.

7.2.3 XML Representation of Structure Types

In general, IDL structures are represented in XML according to a schema defined as an XSD `complexType`. The fields in an IDL structure become unordered elements of the `complexType` with the field `name` appearing as the corresponding element name. This mapping is applied recursively for nested structures.

If the DDS specification defines default values for the structure fields, the corresponding XSD element definition shall provide the same default value.

7.2.3.1 Example (Non-normative)

For example, the **HistoryQosPolicy** is defined in the DDS IDL PSM as:

```
struct HistoryQosPolicy {
```

```
    HistoryQosPolicyKind kind;
    long depth;
};
```

The DDS specification [DDS] states that the default value for the **HistoryQosPolicy** is **KEEP_LAST_HISTORY_QOS** and the default for the depth is 1.

The equivalent representation in XML is defined by the XSD `complexType` `historyQosPolicy` below:

```
<xs:complexType name="historyQosPolicy">
    <xs:all>
        <xs:element name="kind" type="dds:historyKind" minOccurs="0"
                default="KEEP_LAST_HISTORY_QOS" />
        <xs:element name="depth" type="xs:positiveInteger" minOccurs="0"
                default="1" />
    </xs:all>
</xs:complexType>
```

An example XML resource representation satisfying this syntax would be:

```
<history>
    <kind>KEEP_LAST_HISTORY_QOS</kind>
    <depth>10</depth>
</history>
```

7.2.4 XML Representation of Sequences

7.2.4.1 General Rules

The general XML representation of IDL sequences is done following a schema defined as an XSD `complexType`. The `complexType` contains zero or more elements named `element`. Nested inside each element is the XSD schema obtained from mapping the IDL type of the element itself.

7.2.4.1.1 Example (Non-normative)

For example, the **QosPolicyCountSeq** is defined in the DDS IDL PSM as:

```
struct QosPolicyCount {
    long policy_id;
    long count;
};
typedef sequence<QosPolicyCount> QosPolicyCountSeq;
```

The equivalent representation in XML is defined by the XSD `complexType` `qosPolicyCountSeq` defined below:

```
<xs:complexType name="qosPolicyCount">
    <xs:all>
        <xs:element name="policy_id" type="xs:Integer" minOccurs="0" />
        <xs:element name="count" type="xs:Integer" minOccurs="0" />
    </xs:all>
```

```
</xs:complexType>

<xs:complexType name="qosPolicyCountSeq">
    <xs:sequence>
        <xs:element name="element" type="qosPolicyCount"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType >
```

An example XML resource representation satisfying this syntax would be:

```
<qos_policy_count_seq>
    <element>
        <policy_id>1</policy_id>
        <count>23</count>
    </element>
    <element>
        <policy_id>4</policy_id>
        <count>44</count>
    </element>
</qos_policy_count_seq>
```

7.2.4.2 Sequences of Octets

As a special case, sequences of octets are represented either as a comma-separated list of the value of each octet represented in decimal or hexadecimal, or alternatively using Base64 binary. The two options are differentiated by the element name. The latter having the suffix `B64`.

7.2.4.2.1 Example (Non-normative)

For example, the IDL type **UserDataQosPolicy** is defined in the DDS IDL PSM as:

```
struct UserDataQosPolicy {
  sequence<octet> value;
};
```

The equivalent representation in XML is defined by the XSD `complexType` `userDataQosPolicy` defined below:[1]

```
<xs:simpleType name="Binary ">
    <xs:restriction base="xs:string">
        <xs:pattern value="(\s)*(([0-2]?[0-9]?[0-9])|(0[xX][0-9a-fA-F]?[0-9a-fA-
F]))(\s)*(,(\s)*(([0-2]?[0-9]?[0-9])|(0[xX][0-9a-fA-F]?[0-9a-fA-F]))(\s)*)*"/>
        <xs:pattern value=""/>
    </xs:restriction>
</xs:simpleType>
```

[1] Note that long regular expressions, such as the value of `<xs:pattern>`, shall be contained in a single line.

```
<xs:complexType name="userDataQosPolicy">
    <xs:choice>
        <xs:element name="value" type="dds:Binary" minOccurs="0" />
        <xs:element name="valueB64" type="xs:base64Binary" minOccurs="0" />
    </xs:choice>
</xs:complexType>
```

An example XML resource representation satisfying this syntax would be:

```
<user_data_qos>
    <value>84, 104, 101, 32, 0x71, 0x75, 0x69, 0x63</value>
<user_data_qos>
```

Or alternatively:

```
<user_data_qos>
    <valueB64>VGhlJTIwcXVpY2sl</valueB64>
<user_data_qos>
```

7.2.5 XML Representation of Arrays

The XML representation of IDL arrays is the same as it would be for IDL sequences of the same element type.

7.2.6 XML Representation of Duration

The IDL structure **Duration_t** is represented in XML following the general rules for structures defined in sub clause 7.2.3, except that the schema provides the option to use the symbolic defined in the IDL to set the values of the intended elements.

The **Duration_t** structure is defined in the DDS IDL PSM as:

```
struct Duration_t {
  long sec;
  unsigned long nanosec;
};
```

The equivalent representation in XML is defined by the XSD `complexType duration`:

```
<xs:complexType name="duration">
    <xs:all>
        <xs:element name="sec" type="dds:nonNegativeInteger_Duration_SEC"
                minOccurs="0" />
        <xs:element name="nanosec" type="dds:nonNegativeInteger_Duration_NSEC"
                minOccurs="0" />
    </xs:all>
</xs:complexType>
```

7.2.6.1 Example (Non-normative)

An example XML resource representation satisfying the syntax defined above would be:

```
<duration>
    <sec>DURATION_INFINITY</sec>
    <nanosec>DURATION_INFINITY_NSEC</nanosec>
</duration>
```

7.3 Building Blocks

7.3.1 Overview

This specification breaks the syntax used to represent DDS resources in XML into the six different building blocks as shown in Figure 7.1:

- Building Block QoS
- Building Block Types
- Building Block Domains
- Building Block DomainParticipants
- Building Block Applications
- Building Block Data Samples

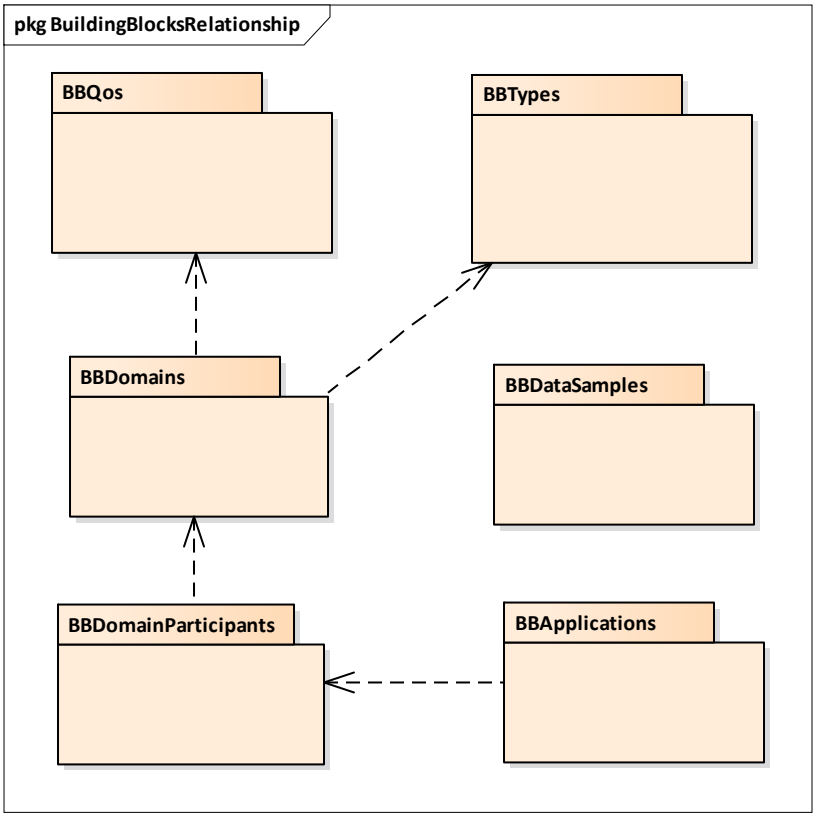

Figure 7.1 Relationship between building blocks

Each of these building blocks is associated with two normative schema files in XSD format that are designed according to the *XML Chameleon Schema Definition* pattern (see sub clause 7.1.3).

- *dds-xml_<building_block_name>_definitions_nonamespace.xsd* contains the type declarations for all the constructs the building block defines. This XSD file specifies neither a `targetNamespace` nor a root element. Therefore, users of this specification may easily integrate this XSD file in their own schema to define custom elements making use of the different building block's constructs without any restriction in terms of tag hierarchy or namespace.

- *dds-xml_<building_block_name>_definitions.xsd* includes the XSD with no `targetNamespace`, defines the top level element for the building block, and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.2 Building Block QoS

7.3.2.1 Purpose

This building block defines the syntax to represent DDS QoS in XML.

7.3.2.2 Dependencies with other Building Blocks

This building block has no dependencies on other building blocks.

7.3.2.3 Syntax

The following XSD files contain the syntax of the resource representations defined by this building block:

- *dds-xml_qos_definitions_nonamespace.xsd* contains the XSD type definition for all the DDS QoS. It defines no `targetNamespace` so that it can be reused by other schemas following the *XML Chameleon Schema Definition* pattern.

- *dds-xml_qos_definitions.xsd* defines the `<qos_library>` top-level element, and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.2.4 Explanations and Semantics

7.3.2.4.1 QoS Libraries and QoS Profiles

QoS Libraries are the top level element of the Building Block QoS. They are collections of QoS profiles, which group a set of related QoS—usually one per entity.

7.3.2.4.1.1 Example (Non-normative)

```
<qos_library name="ReliableProfilesLibrary">
    <qos_profile name="StrictReliableCommunicationProfile">
        <datawriter_qos>
            <history>
                <kind>KEEP_ALL_HISTORY_QOS</kind>
            </history>
            <reliability>
                <kind>RELIABLE_RELIABILITY_QOS</kind>
            </reliability>
        </datawriter_qos>
        <datareader_qos>
            <history>
                <kind>KEEP_ALL_HISTORY_QOS</kind>
            </history>
            <reliability>
                <kind>RELIABLE_RELIABILITY_QOS</kind>
            </reliability>
        </datareader_qos>
    </qos_profile>
</qos_library>
```

7.3.2.4.2 QoS Profile Inheritance

A QoS Profile can inherit from other QoS Profiles using the `base_name` XML attribute.

A QoS profile can only inherit from QoS profiles that have been defined its definition

7.3.2.4.2.1 .Example (Non-normative)

```
<qos_profile name="MyProfile" base_name="BaseProfile">
    ...
</qos_profile>
```

7.3.2.4.3 QoS Profile Topic-name Filters

A QoS Profile may contain several DataWriter, DataReader, and Topic QoS settings that are selected based on the evaluation of a filter expression on the topic name.

The filter expression is specified via the `topic_filter` XML attribute in the QoS definition of the entity QoS.

If the topic filter is unspecified, the filter "*" will be assumed. The QoS with an explicit `topic_filter` attribute definition will be evaluated in order; they take precedence over a QoS without a topic filter expression.

7.3.2.4.3.1 Example (Non-normative)

```
<qos_profile name="StrictReliableCommunicationProfile">
    <datawriter_qos topic_filter="A*">
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
    </datawriter_qos>

    <datawriter_qos topic_filter="B*">
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
        <resource_limits>
            <max_samples>128</max_samples>
            <max_samples_per_instance>128</max_samples_per_instance>
            <initial_samples>128</initial_samples>
            <max_instances>1</max_instances>
            <initial_instances>1</initial_instances>
        </resource_limits>
    </datawriter_qos>
    ...
</qos_profile>
```

7.3.2.4.4 QoS Profiles with a Single QoS

The definition of an individual QoS is a shortcut for defining a QoS profile with a single QoS.

7.3.2.4.4.1 Example (Non-normative)

For example the XML:

```
<datawriter_qos name="KeepAllWriter">
    <history>
        <kind>KEEP_ALL_HISTORY_QOS</kind>
    </history>
</datawriter_qos>
```

Is equivalent to the following XML:

```
<qos_profile name="KeepAllWriter">
    <datawriter_qos>
        <history>
            <kind>KEEP_ALL_HISTORY_QOS</kind>
        </history>
    </datawriter_qos>
</qos_profile>
```

7.3.3 Building Block Types

7.3.3.1 Purpose

This building block gathers the syntax used to represent DDS Types in XML. Additionally, it provides capabilities that are necessary or convenient for the organization and management of types and other XML resource representations.

7.3.3.2 Dependencies with other Building Blocks

This building block has no dependencies on other building blocks.

7.3.3.3 Syntax

The following XSD files contain the syntax of the resource representations defined by this building block:

- *dds-xml_type_definitions_nonamespace.xsd* contains the XSD type definitions for all the DDS types. It defines no `targetNamespace` so that it can be reused by other schemas following the *XML Chameleon Schema Definition* pattern.

- *dds-xml_type_definitions.xsd* defines the `<types>` top-level element, and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.4 Building Block Domains

7.3.4.1 Purpose

This building block defines the syntax used to represent DDS Domains in XML. Domains provide a data space where information can be shared by reading and writing a set of Topics, which are associated to registered data types.

7.3.4.2 Dependencies with other Building Blocks

This building block depends on the Building Block QoS and the Building Block Types.

7.3.4.3 Syntax

The following XSD files contain the syntax of the resources represented by this building block:

- *dds-xml_domain_definitions_nonamespace.xsd* contains the XSD type definition for domains and their contained entities. It defines no `targetNamespace` so that it can be reused by other schemas following the *XML Chameleon Schema Definition* pattern.

- *dds-xml_domain_definitions.xsd* defines the `<domain_library>` root element and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.4.4 Explanations and Semantics

7.3.4.4.1 Defining a Domain

A Domain includes a set of Topics and Registered Types that can be read and written in the Domain.

Register types shall provide a reference to data types that have been previously defined using the `type_ref` XML attribute. The name under which types are registered may be different than original type name.

Topics shall refer to a registered type using the `register_type_ref` XML attribute. Topics may also specify QoS settings inline following the syntax defined in the Building Block QoS. QoS profile inheritance through the `base_name` attribute may be used as defined in clause 7.3.2.4.2.

7.3.4.4.1.1 Example (Non-normative)

```
<domain name="MyDomain" domain_id="10">
    <register_type name="MyFirstRegisterType" type_ref= "MyType" />
    <register_type name="MySecondRegisterType" type_ref="MyType" />
    <topic name="FisrstTopic" register_type_ref="MyFirstRegisterType">
        <topic_qos base_name="BaseQoSProfile" />
    </topic>
    <topic name="SecondTopic" register_type_ref="MySecondRegisterType" />
</domain>
```

7.3.4.4.2 Domain Inheritance

A Domain can inherit from other Domains using the `base_name` XML attribute. A Domain can only inherit from domains that have been defined before.

7.3.4.4.2.1 Example (Non-normative)

```
<domain name="MyDomain" base_name="BaseDomain">
    ...
</domain>
```

7.3.5 Building Block DomainParticipants

7.3.5.1 Purpose

This block defines the syntax to represent DDS DomainParticipants and their contained entities (i.e., Publishers, Subscribers, DataWriters, and DataReaders) in XML.

7.3.5.2 Dependencies with other Building Blocks

This building block depends on the Building Block QoS, the Building Block Types, and the Building Block Domains.

7.3.5.3 Syntax

The following XSD files contain the syntax of the resources represented by this building block:

- *dds-xml_domainparticipant_defintions_nonamespace.xsd* contains the XSD type definition for all the DDS entities. It defines no `targetNamespace` so that it can be reused by other schemas following the *XML Chameleon Schema Definition* pattern.

- *dds-xml_domainparticipant_definitions.xsd* defines the `<domain_participant_library>` root element and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.5.4 Explanations and Semantics

7.3.5.4.1 Domain Participant Libraries, DomainParticipants, and Contained Entities

Domain Participant Libraries are collections of DomainParticipants and contained entities. They are the top level elements of the Building Block DomainParticipants.

DomainParticipants are responsible for the creation and deletion of Publishers and Subscribers, which are likewise responsible for the deletion and creation of DataWriters and DataReaders.

To represent this hierarchical relationship between DDS entities, each entity is declared as a nested XML tag under the declaration of its parent entity.

7.3.5.4.1.1 Example (Non-normative)

```
<domain_participant_library="MyDomainParticipantLibrary">
    <domain_participant name="MyDomain" domain_ref="MyDomain">
        <publisher name="MyPublisher">
            <data_writer name="MyDataWriter" topic_ref="MyTopic"/>
        </publisher>

        <subscriber name="MySubscriber">
            <data_reader name="MyDataReader"/>
        </subscriber>
    </domain_participant>
</domain_participant_library>
```

7.3.5.4.2 Using the Domain Building Block

DomainParticipants may refer to a Domain declared in the context of a Domain Library (see Building Block Domains) using the `domain_ref` XML attribute. This makes the Topics and Register Types defined in the Domain available for all the DataWriters and DataReaders defined in the context of the DomainParticipant.

The Domain Id specified in the parent Domain can be overridden via the `domain_id` XML attribute.

7.3.5.4.2.1 *Example (Non-normative)*

```
<domain_participant name="MyDomain" domain_ref="MyDomain" domain_id="32">
   ...
</domain_participant>
```

7.3.5.4.3 DomainParticipant Inheritance

DomainParticipants may inherit from DomainParticipants defined in the context of a DomainParticipant Library using the `base_name` XML attribute. A DomainParticipant can only inherit from DomainParticipants in DomainParticipant Libraries that have been defined before its own definition.

7.3.5.4.3.1 *Example (Non-normative)*

```
<domain_participant_library name="AParticipantLibrary">
    <domain_participant name="MyParticipant"
            base_name="AnotherParticipantLibrary::TheirParticipant" />
</domain_participant_library>
```

7.3.5.4.4 Inline Entity QoS Settings Definition

Inline QoS setting definition is allowed in the context an entity's definition. Inline QoS settings apply only to the entity in whose context they are being defined.

Inline entities may inherit from an existing QoS Profile using the `base_name` XML attribute.

7.3.5.4.4.1 *Example (Non-normative)*

```
<domain_participant_library name="AParticipantLibrary">
    <domain_participant name="MyParticipant"
            base_name="AnotherParticipantLibrary::TheirParticipant">
        <domain_participant_qos base_name="BaseProfile">
            <entity_factory>
                <autoenable_created_entities>false</autoenable_created_entities>
            </entity_factory>
        </domain_participant_qos>
    </domain_participant>
</domain_participant_library>
```

7.3.6 Building Block Applications

7.3.6.1 Purpose

This block defines the XML syntax to represent DDS applications that participate (or may be participating) in the DDS Global Data Space.

7.3.6.2 Dependencies with other Building Blocks

This building block depends on the Building Block QoS, the Building Block Types, the Building Block Domains, and the Building Block DomainParticipants.

7.3.6.3 Syntax

The following XSD files contain the syntax of the resources represented by this building block:

- *dds-xml_application_definitions_nonamespace.xsd* contains the XSD type definition for applications and their contained entities. It defines no `targetNamespace` so that it can be reused by other schemas following the *XML Chameleon Schema Definition* pattern.

- *dds-xml_application_definititons.xsd* defines the `<application_library>` root element and sets `targetNamespace` to `http://www.omg.org/dds`.

7.3.6.4 Explanations and Semantics

7.3.6.4.1 Applications, DomainParticipants, and Contained Entities

Application Libraries are collections of Applications, which are composed of a set of DomainParticipants and contained entities. They are the top level elements of the Building Block Applications.

7.3.6.4.1.1 Example (Non-normative)

```
<application_library name="ShapesRelatedApplications">
    <application name="SimpleShapesDemoApplication">
        <domain_participant name="MyParticipant"
                domain_ref="ShapesDomainLibrary::ShapesDomain">
            <publisher name="MyPublisher">
                <data_writer name="MySquareWriter" topic_ref="Square"/>
                <data_reader name="MySquareReader" topic_ref="Square"/>
            </publisher>
        </domain_participant>
    </application>
</application_library>
```

7.3.6.4.2 Using DomainParticipants defined in DomainParticipant Libraries

DomainParticipants defined in the context of an Application may inherit from DomainParticipants defined in the context of a DomainParticipant Library using the `base_name` XML attribute as described in the Building Block DomainParticipants.

A DomainParticipant can only inherit from DomainParticipants in DomainParticipant Libraries that have been defined before its own definition.

7.3.6.4.2.1 Example (Non-normative)

```
<application name="SimpleShapesDemoApplication">
    <domain_participant name="MyParticipant"
            base_name="ShapesParticipantLibrary::MyParticipant"/>
</application>
```

7.3.7 Building Block Data Samples

7.3.7.1 Purpose

This block defines XML syntax to represent Data Samples that may be exchanged between different DDS applications.

7.3.7.2 Dependencies with other Building Blocks

This building block has no dependencies on other building blocks.

7.3.7.3 Syntax

7.3.7.3.1 General Rules

Because it is impossible to define a generic XSD file to represent Data Samples of all the possible Data Type combinations in DDS, the syntax to represent Data Samples is based on the XML representation rules specified in sub clauses 7.1, 7.2.1, 7.2.2, 7.2.3, and 7.2.6.

To comply with the previously specified representation in [DDS-XTYPES], this building block defines its own XML Representation for Sequences and Arrays in sub clause 7.3.7.3.2.

Users of this specification who may want to define schema files to specify the syntax of specific Data Samples can follow the *XML Chameleon Schema Definition* pattern.

7.3.7.3.2 XML Representation of Sequences and Arrays

The general XML representation of IDL sequences and arrays is done following a schema defined as an XSD `complexType`. The `complexType` contains zero or more elements named `item`. Nested inside each element is the XSD schema obtained from mapping the IDL type of the element itself.

7.3.7.3.2.1 Example (Non-normative)

For example, for the sequence **CoordinatesSeq** defined in IDL:

```
struct Coordinates {
  long x;
  long y;
};
typedef sequence<Coordinates> CoordinatesSeq;
```

The equivalent representation in XML is defined by the XSD `complexType coordinatesSeq` below:

```
<xs:complexType name="coordinates">
    <xs:all>
        <xs:element name="x" type="xs:Integer" minOccurs="0" />
        <xs:element name="y" type="xs:Integer" minOccurs="0" />
    </xs:all>
</xs:complexType>

<xs:complexType name="coordinatesSeq">
    <xs:sequence>
```

```
        <xs:element name="item" type="coordinates"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType >
```

The XML data sample representation would be the following:

```
<coordinates_seq>
    <item>
        <x>1</x>
        <y>15</y>
    </item>
    <item>
        <x>4</x>
        <y>11</y>
    </item>
</coordinates_seq>
```

7.3.7.4 Data Sample Representation Examples (non-normative)

The following non-normative examples illustrate the application of the aforementioned XML representation rules for different Data Sample kinds.

7.3.7.4.1 Structure Types

```
<aStruct>
    <aLong>0</aLong>
    <aLong>1</aLong>
    <aNestedStruct>
        <aLong>0</aLong>
        <aLong>1</aLong>
    </aNestedStruct>
</aStruct>
```

7.3.7.4.2 Unions

```
<data>
    <aUnion>
        <aLong>0</aLong>
    </aUnion>
</data>
```

7.3.7.4.3 Sequences

```
<aStruct>
    <aSequenceOfDoubles>
        <item>1.1</item>
        <item>1.2</item>
```

```
        </aSequenceOfDoubles>
        <aSequenceOfStructs>
            <item>
                <aFloat>0.0</aFloat>
            </item>
            <item>
               <aFloat>1.0</aFloat>
            </item>
        </aSequenceOfStructs>
    </aStruct>
```

7.3.7.4.4 Arrays

```
    <aStruct>
        <anArrayOfLongs>
            <item>1</item>
            <item>2</item>
        </anArrayOfLongs>
        <anArrayOfStructs>
            <item>
                <aFloat>0.0</aFloat>
            </item>
            <item>
                <aFloat>1.0</aFloat>
            </item>
        </anArrayOfStructs>
    </aStruct>
```

7.3.7.4.5 Primitive Types

```
    <aStruct>
        <aShort>3</aShort>
        <aUShort>2</aUShort>
        <anEnum>ACE</anEnum>
        <aLong>2452</aLong>
        <aULong>3245</aULong>
        <aLongLong>23121</aLongLong>
        <aULongLong>2345212</aULongLong>
        <aFloat>2.3</aFloat>
        <aDouble>3.14</aDouble>
        <aBoolean>false</aBoolean>
        <aString>A string!</aString>
        <anotherString>&#xa1;El r&#xed;o mi&#xf1;o es precioso!</anotherString>
```

```
    <anOctet>0x00</anOctet>
    <aChar>a</aChar>
</aStruct>
```

8 Building Block Sets

This Chapter defines some relevant combinations of building blocks called *building block sets*. A block set is a collection of building blocks.

Block sets provide a convenient mechanism to group related building blocks so that other specifications can reference the complete set as opposed to the individual building blocks.

8.1 DDS System Block Set

This block set offers the ability to describe a complete DDS system.

It contains:

- Building Block QoS
- Building Block Types
- Building Block Domains
- Building Block DomainParticipants
- Building Block Applications