

# Web-Enabled DDS

*FTF Beta 1*

---

OMG Document Number: ptc/2013-11-22

Standard document URL: <http://www.omg.org/spec/DDS-WEB>

Machine Consumable Files:

Normative:

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_rest1.xsd](http://www.omg.org/spec/DDS-WEB/20131122/webdds_rest1.xsd)

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_soap1\\_types.xsd](http://www.omg.org/spec/DDS-WEB/20131122/webdds_soap1_types.xsd)

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_soap1.wsdl](http://www.omg.org/spec/DDS-WEB/20131122/webdds_soap1.wsdl)

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_soap1\\_notify.wsdl](http://www.omg.org/spec/DDS-WEB/20131122/webdds_soap1_notify.wsdl)

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_pim\\_model.xmi](http://www.omg.org/spec/DDS-WEB/20131122/webdds_pim_model.xmi)

Non-normative:

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_rest1\\_example.xml](http://www.omg.org/spec/DDS-WEB/20131122/webdds_rest1_example.xml)

[http://www.omg.org/spec/DDS-WEB/20131122/webdds\\_pim\\_model.eap](http://www.omg.org/spec/DDS-WEB/20131122/webdds_pim_model.eap)

---

This OMG document replaces the submission document (mars/13-05-21, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to [issues@omg.org](mailto:issues@omg.org) by March 31, 2014.

You may view pending issues for this specification from the OMG revision issues web page

<http://www.omg.org/issues>.

The FTF Recommendation and Report for this specification will be published on September 26, 2014. If you are reading this after that date, please download the available specification from the OMG Specifications web page

<http://www.omg.org/spec/>.

Copyright © 2013, eProxima  
Copyright © 2014, Object Management Group, Inc. (OMG)  
Copyright © 2013, PrismTch, Ltd  
Copyright © 2013, Real-Time Innovations, Inc. (RTI)  
Copyright © 2013, THALES

## 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.

## DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED “AS IS” AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

## RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph © (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph ©(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

## TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

## COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.



## **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](http://www.omg.org/report_issue.htm).)



# Table of Contents

Preface .....	iii
<b>About the Object Management Group .....</b>	<b>iii</b>
OMG .....	iii
OMG Specifications .....	iii
<b>Issues.....</b>	<b>iv</b>
<b>Introduction.....</b>	<b>iv</b>
<b>Overview of this Specification .....</b>	<b>v</b>
Web-Enabled DDS (WebDDS) Object Model .....	v
Platform-Specific Mappings .....	vi
<b>1 Scope .....</b>	<b>1</b>
<b>1.1 General.....</b>	<b>1</b>
<b>1.2 WebDDS Object Model .....</b>	<b>1</b>
<b>1.3 Platform-Specific Mappings .....</b>	<b>2</b>
<b>1.4 Example Scenarios .....</b>	<b>2</b>
<b>2 Conformance .....</b>	<b>3</b>
<b>3 Normative References.....</b>	<b>3</b>
<b>4 Terms and Definitions .....</b>	<b>4</b>
<b>5 Symbols .....</b>	<b>6</b>
<b>6 Additional Information .....</b>	<b>7</b>
<b>6.1 Changes to Adopted OMG Specifications [optional] .....</b>	<b>7</b>
<b>6.2 Acknowledgements.....</b>	<b>7</b>
<b>7 WebDDS Object Model .....</b>	<b>9</b>
<b>7.1 General.....</b>	<b>9</b>
<b>7.2 Model Overview .....</b>	<b>11</b>
<b>7.3 Users and Access Control .....</b>	<b>12</b>
7.3.1 Class WebDDS::WSDDS .....	15
7.3.2 Class WebDDS::AccessController .....	17
7.3.3 Class WebDDS::User (conceptual) .....	19
7.3.4 Class WebDDS::Application.....	20
<b>7.4 DDS Proxy classes.....</b>	<b>21</b>
7.4.1 ReturnStatus .....	22
7.4.2 Access control and permissions .....	23
7.4.3 Class WebDDS::Application (details).....	24
7.4.4 Class WebDDS::DomainParticipant .....	30
7.4.5 Class WebDDS::Publisher .....	42
7.4.6 Class WebDDS::Subscriber .....	46
7.4.7 Class WebDDS::DataWriter .....	49
7.4.8 Class WebDDS::DataReader .....	53
7.4.9 Class WebDDS::WaitSet .....	57
<b>8 Web-Enabled DDS Platform-Specific Mappings .....</b>	<b>59</b>
<b>8.1 General.....</b>	<b>59</b>
<b>8.2 Formats and Representations for the REST and SIMPLE-WSDL-SOAP platforms .....</b>	<b>59</b>
8.2.1 QoS Representations.....	59
8.2.2 Type Representations.....	59
8.2.3 Data Representations.....	59

8.2.4	WebDDS Entity Representations.....	60
<b>8.3</b>	<b>REST Platform .....</b>	<b>61</b>
8.3.1	Mapping of WebDDS PIM to Resources.....	62
8.3.2	Mapping rules from WebDDS PIM operations to REST methods .....	62
8.3.3	Complete mapping of WebDDS PIM operations to REST methods .....	64
8.3.4	Object representations used by the REST platform .....	68
8.3.5	HTTP Headers used by the REST platform .....	70
<b>8.4</b>	<b>Simplified SOAP Platform.....</b>	<b>71</b>
<b>8.5</b>	<b>Transport-Level Protocol Considerations .....</b>	<b>80</b>
<b>Annex A - References.....</b>		<b>81</b>



## **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); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

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

#### **OMG Specifications**

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. A listing of all OMG Specifications is available from the OMG website at:

<http://www.omg.org/spec/index.htm>

Specifications are organized by the following categories:

#### **Business Modeling Specifications**

#### **Middleware Specifications**

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

#### **IDL/Language Mapping Specifications**

#### **Modeling and Metadata Specifications**

- **UML, MOF, CWM, XMI**
- **UML Profile**

## Modernization Specifications

### Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- CORBAServices
- CORBAFacilities

### OMG Domain Specifications

### CORBA Embedded Intelligence Specifications

### CORBA Security Specifications

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

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

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

## Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to [http://www.omg.org/report\\_issue.htm](http://www.omg.org/report_issue.htm).

## Introduction

The OMG DDS specification [1] defines an API that applications can use to publish and subscribe to data in a "Global Data Space." The API defined by the DDS specification must be implemented by means of local interfaces in each of the supported programming languages. This approach limits the use of DDS to applications that (a) can link into the application the vendor-provided DDS implementation libraries, and (b) use one of the programming languages supported by the DDS implementations. In practice, these limitations mean that no standard APIs or mechanisms to access the DDS Global Data Space from applications running inside a web browser (e.g. JavaScript applications), or from remote "client" libraries that leverage commonly used scripting languages, such as PHP, Perl, Ruby, or Python.

Another important usage scenario is that of disconnected or stateless clients. These are typically implemented as single-command, short-lived processes, for example shell commands or web-server CGI scripts. Under this usage scenario, a user starts a client application to execute a very simple action, such as publishing data on a Topic or receiving the latest data on a Topic. The client executes the action, returns the output (typically to the `stdout`), and then exits. This is a common scenario when integrating with web-server applications that use CGI scripts to execute individual actions.

Disconnected or stateless clients are problematic because of the dynamic, one-to-many nature of publish-subscribe applications and the fact that DDS does not require the presence of a broker or centralized server. In order to publish or subscribe to data, a DDS application must join a domain, discover other participants and other publishers and subscribers, exchange the information, and then

remain present long enough for the reliability protocol to ensure that all subscribers receive the information. This makes the implementation of a short-lived process challenging. How long should the process wait to discover all subscribers or publishers? The approach is also inefficient. Each time the process starts, it must create new DDS entities that then must be discovered by the rest of the system—only to be destroyed shortly afterwards.

With the increasing adoption of DDS for the integration of large distributed systems, it is desirable to define standard ways whereby web-based applications can: access DDS; publish and subscribe to data into the DDS Global Data Space; and benefit from the performance, scalability, and quality of service offered by DDS implementations. In addition, it is desirable for this approach to support efficient access to the Global Data Space by disconnected or stateless clients. Note that all these things were possible before this specification, but the approaches were non-standard.

## Overview of this Specification

This specification includes (1) a platform-independent Abstract Interaction Model of how web-clients should access a DDS System and (2) a set of mappings to specific web platforms that realize the PIM in terms of standard web technologies and protocols. These allow a web client to participate in the DDS global data space in a way that is portable across implementations.

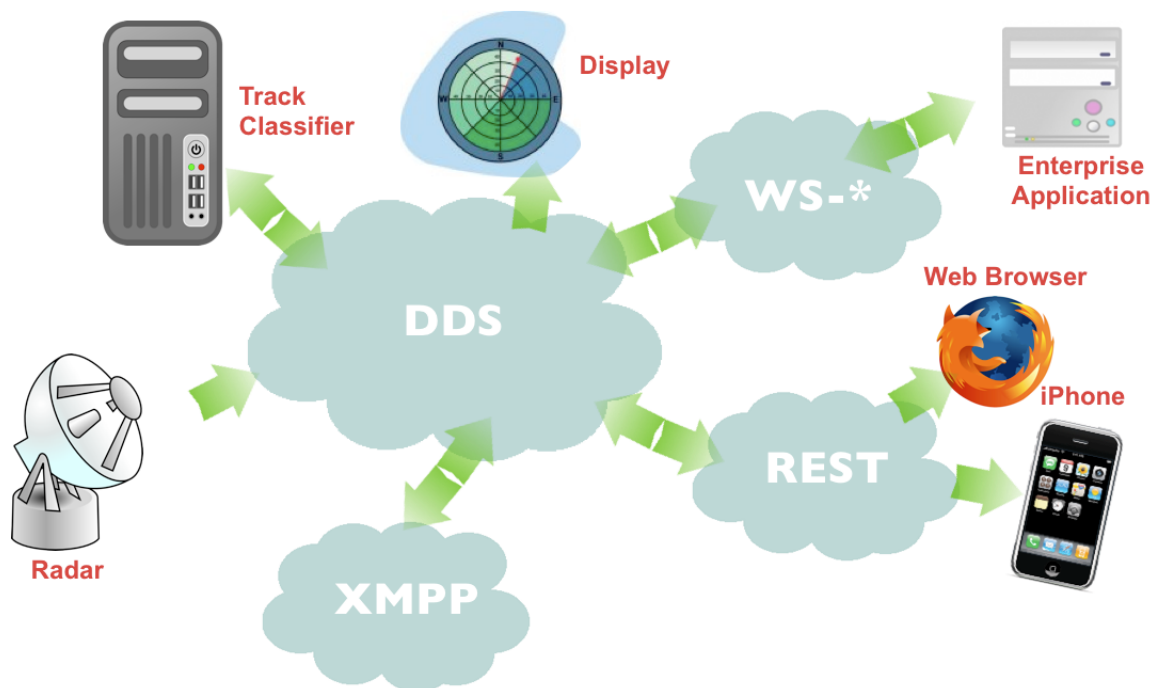


Figure 1—Conceptual topology

## Web-Enabled DDS (WebDDS) Object Model

The “WebDDS Object Model” is the object exposed to the web-enabled DDS clients. Logically one can think of this as logical equivalent to the “DDS Object Model”. This specification does not simply reuse the standard “DDS Object Model” for three reasons:

1. The DDS Object Model is intended for use with a local programming API. For this reason, the DDS Object Model contains many objects and methods with strongly typed parameters, as well as a direct callback interface by means of listener objects that the application registers with the middleware. Such an API is not suitable for web clients that typically prefer more “resource-oriented interfaces” and also expect a simplified interface with no callbacks and

where all parameters are encoded in text.

2. Web client connectivity is inherently intermittent. By the very nature of the HTTP protocol, clients are continually being connected and disconnected from the server. Therefore, the Web Enabled DDS Object must overcome this by introducing a “session,” whose life can span multiple physical connections.
3. Web clients can access a Web-Enabled DDS service from any location, and therefore it is desirable to have an access control model that authenticates each user/principal, controls whether the principal can access the DDS Global Data Space, and controls which operations each principal can perform (e.g. which DDS Topics it can read and write).

### **Platform-Specific Mappings**

Web clients accessing data and services over the web typically use a mix of architectural approaches, technologies, and protocols including: RESTful [2], WSDL/SOAP Web Services [3] [5], HTTP[13], RSS, ATOM, and/or XMPP [4]. Each of these approaches presents advantages and disadvantages; selecting which to use is often driven by business requirements. For example:

1. REST is the most universally deployed architectural approach on the Web and is used for most of the “cloud” services such as those offered by Amazon and Google. It is also simple and friendly to web browsers, which use bookmarks and links to get to the data directly. However, it lacks a well-established formal language with which to define interfaces<sup>1</sup>.
2. Despite being less widely deployed, Web Services have a language (WSDL) that can be used to formally define interfaces and are supported by the major providers of Enterprise Service Bus (ESB) infrastructure. However, they are less friendly to web browsers and cannot be easily called from JavaScript.
3. RSS and Atom are popular protocols for retrieving data. RSS is more established but, unlike Atom, it only defines how to receive existing data, not how to post new data.
4. XMPP is a simple and popular protocol based on HTTP and XML that was originally developed for Internet chat applications but that is now becoming popular as a general protocol for peer-to-peer application communication.

Because of the existence of multiple popular web technologies and protocols (referred to in this specification as “web platforms”), each with its own strengths and limitations, the Web-Enabled DDS specification is not tied to or associated with a single web platform. Instead, it maps the WebDDS Object Model into several web platforms. The intent of this approach is for all platform mappings to be equivalent and interoperable.

---

<sup>1</sup> The Web Applications Description Language (WADL) (see <http://www.w3.org/Submission/wadl/>) has been proposed as a “member submission” to the W3C, but as of 2012 the W3C has stated it “has no plans to take up work based on this submission” (see <http://www.w3.org/Submission/2009/03/Comment>)

# 1 Scope

## 1.1 General

The goal of this specification is to define the means for applications using standard web protocols to participate as first-class citizens as publishers and subscribers of data in the DDS Global Data space. This participation is realised by exposing a WebDDS Object Model and making it accessible as a web service, a REST resources, or some other standard web protocol. Exposing access via these web-friendly protocols allow applications built on various technology stacks (e.g. JavaScript, Python, PHP, Perl , etc.) to communicate with native DDS applications.

This specification consists of the following sections:

## 1.2 WebDDS Object Model

This specification defines a platform-independent WebDDS Object Model using UML. The model defines the objects, interfaces, and operations to be implemented by the service. This specification furthermore defines how this model relates to the DDS Object Model as defined in the OMG Data-Distribution Service Specification [DDS] (we shall refer to this object model as the “Standard DDS Object Model”). In other words, it defines the effects that each operation performed on the WebDDS Object Model has, if any, on the related Standard DDS Object Model Entities.

The WebDDS Object Model both extends and simplifies the Standard DDS Object Model. It extends the Standard DDS Object Model in order to include an *access control model* and a *user & application management model* to support disconnected clients. It is also a simplification of the Standard DDS Object Model in order to reduce the number of objects and operations and make it more suitable for web clients.

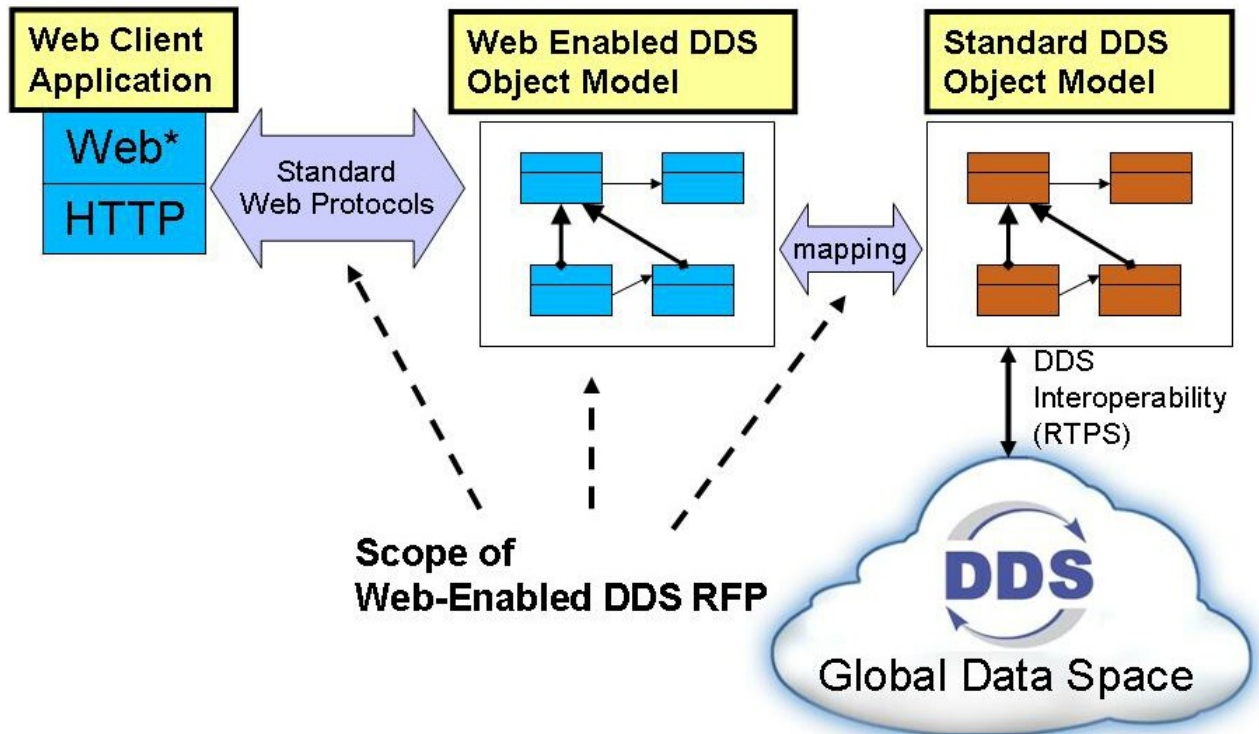


Figure 2 — Scope of this specification

### 1.3 Platform-Specific Mappings

This specification also provides a collection of mappings describing how the Abstract Interaction Model can be accessed and manipulated using common Web technologies, such as RESTful services and SOAP Web Services.

- The **RESTful Platform** mapping defines how to access the Object Model using a RESTful interface [REST].
- The **SOAP Platform** mapping defines how to access the Object Model using WSDL interfaces and SOAP messages [WSDL].

### 1.4 Example Scenarios

Possible (non-normative) scenarios are:

- As a subscriber, a web page can register a subscription to a particular topic in order to display it in a web browser.
- As a publisher, a Web application could be used to publish information using a web browser.
- Based on the given examples, one could design more complex applications by mixing

publishers and subscribers in a single application.

## 2 Conformance

This specification defines the following conformance profiles:

**Table 1 Conformance Profiles**

<i>Profile</i>	<i>Mandatory or Optional</i>	<i>Conformance Sections</i>
REST	Mandatory	Web-DDS Object Model (Clause 7) and REST Platform (Sub clause 8.3)
SIMPLE-REST	Optional	Web-DDS Object Model (Clause 7) and Simplified REST Platform (Sub clause 8.3.5)
SIMPLE-WSDL-SOAP	Optional	Web-DDS Object Model (Clause 7) and WSDL SOAP Platform (Sub clause 8.4)

Conforming implementations *must* implement the WebDDS Object Model as mapped to the RESTful Platform.

Conforming implementations *may* implement one or more additional platform-specific mappings as described in Clause 8 of this specification. Each of these individually shall represent an optional compliance point for this specification.

## 3 Normative References

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

- **[DDS]** *Data Distribution Service for Real-Time Systems Specification*, version 1.2  
<http://www.omg.org/spec/DDS/1.2>
- **[DDS-CCM]** [8] *DDS for light-weight CCM specification (DDS4CCM)* version 1.0.  
<http://www.omg.org/spec/dds4ccm/1.1/>
- **[DDS-XTypes]** *DDS Extensible Types Specification (DDS-XTYPES)* version 1.0.  
<http://www.omg.org/spec/DDS-XTypes/1.0/>
- **[HTTP]** *Hypertext Transfer Protocol*, version 1.1 (IETF RFC 2616);  
<http://tools.ietf.org/rfc/rfc2616.txt>.
- **[HTTP-Auth]** “HTTP Authentication: Basic and Digest Access Authentication” IETF RFC 2617  
<http://tools.ietf.org/html/rfc2617>

- [WebSockets] *The WebSocket Protocol*, version 1.1 (IETF RFC 2616); <http://tools.ietf.org/rfc/rfc6455.txt>.
- [SOAP] *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*; <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.
- [WSDL] *The Web Services Description Language (WSDL)*, version 1.1; <http://www.w3.org/TR/wsdl>.
- [XML] *Extensible Markup Language (XML)*, version 1.1, Fifth Edition (W3C recommendation, November 2008). <http://www.w3.org/TR/REC-xml/>

## 4 Terms and Definitions

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

### **ATOM platform**

For the purposes of this specification, the name ATOM refers to a pair of related standards: (1) the Atom Syndication Format, which is an XML language used for web feeds published by the IETF as RFC 4287 and (2) the Atom Publishing Protocol, which is a simple HTTP-based protocol for creating and updating web resources published by the IETF as RFC 5023.

### **Data-Centric Publish-Subscribe (DCPS)**

The mandatory portion of the DDS specification used to provide the functionality required for an application to publish and subscribe to the values of data objects.

### **Data Distribution Service (DDS)**

An OMG distributed data communications specification that allows Quality of Service policies to be specified for data timeliness and reliability. It is independent of implementation languages.

### **Data representation**

A Data Representation is a serialization format for storing and/or transmitting the state of structured objects. This term is used per [DDS-XTypes].

### **DDS world**

A “DDS world” consists of a collection of peers communicating over the Data Distribution Service and the collection of data observable by those peers. *See also “web world.”*

### **RESTful platform**

As described in a dissertation by Roy Fielding, REST is an “architectural style” that exploits the existing technology and protocols of the Web, including HTTP (Hypertext Transfer Protocol) and



XML. REST is simpler to use than the well-known SOAP (Simple Object Access Protocol) approach, which requires writing or using a provided server program (to serve data) and a client program (to request data).

### **RSS platform**

RSS stands for Really Simple Syndication. RSS is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document (which is called a “feed” or “channel”) includes full or summarized text plus metadata such as publishing dates and authorship. Feeds benefit publishers by letting them syndicate content automatically. They benefit readers who want to subscribe to timely updates from favored websites or to aggregate feeds from many sites into one place. RSS feeds can be read using software called an “RSS reader,” “feed reader,” or “aggregator,” which can be web-based, desktop-based, or mobile-device-based. A standardized XML file format allows the information to be published once and viewed by many different programs.

### **Web client**

Generic term used to refer to an application that is accessing the Web-Enabled DDS over standard web protocols, including but not limited to plain HTTP, SOAP over HTTP, RSS over HTTP, etc.

### **Web-enabled**

Generic term used to indicate that a particular technology is accessible by Web Clients by means of standard web protocols, including but not limited to plain HTTP, SOAP over HTTP, RSS over HTTP, etc.

### **Web socket**

WebSocket is a web technology providing bi-directional communications over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.

### **Web world**

A “web world” consists of a collection of client applications communicating with one another using web protocols, such as SOAP or REST, in conformance with this specification. These clients communicate with one another and with the DDS world (*see “DDS world”*) through a web-enabled DDS gateway.

### **WSDL platform**

As described by the W3C, WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to

communicate; however, the only bindings standardized by the W3C describe how to use WSDL in conjunction with SOAP, HTTP GET/POST, and MIME. For the purposes of this specification, the term “WSDL platform” shall refer to the set of standard specifications defined by the WS-I Basic Profile 1.1 specification.

### **XMPP platform**

The Extensible Messaging and Presence Protocol (XMPP) is an open technology for asynchronous communication that powers a wide range of applications including instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data. The base specifications of the Extensible Messaging and Presence Protocol (XMPP) formalize the core protocols developed within the Jabber open-source community in 1999. They are published as IETF RFCs 3920 and 3921.

## **5 Symbols**

This specification uses the following symbols and abbreviations:

**Table 2 Symbols and Abbreviations**

DDS	Data Distribution Service
IDL	Interface Description Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
Atom	Atom Publishing Protocol
RSS	Really Simple Syndication
REST	Representational State Transfer
HTTP	HyperText Transfer Protocol
CCM	CORBA Component Model
QoS	Quality of Service
RFP	Request For Proposal
WSDL	Web Services Description Language

## **6 Additional Information**

### **6.1 Changes to Adopted OMG Specifications [optional]**

This specification does not extend or modify any existing OMG specifications.

### **6.2 Acknowledgements**

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

- eProsima
- PrismTech
- Real-Time Innovations, Inc. (RTI)
- Thales

The following additional companies support this specification:

- General Dynamics
- Twin Oaks Computing, Inc.

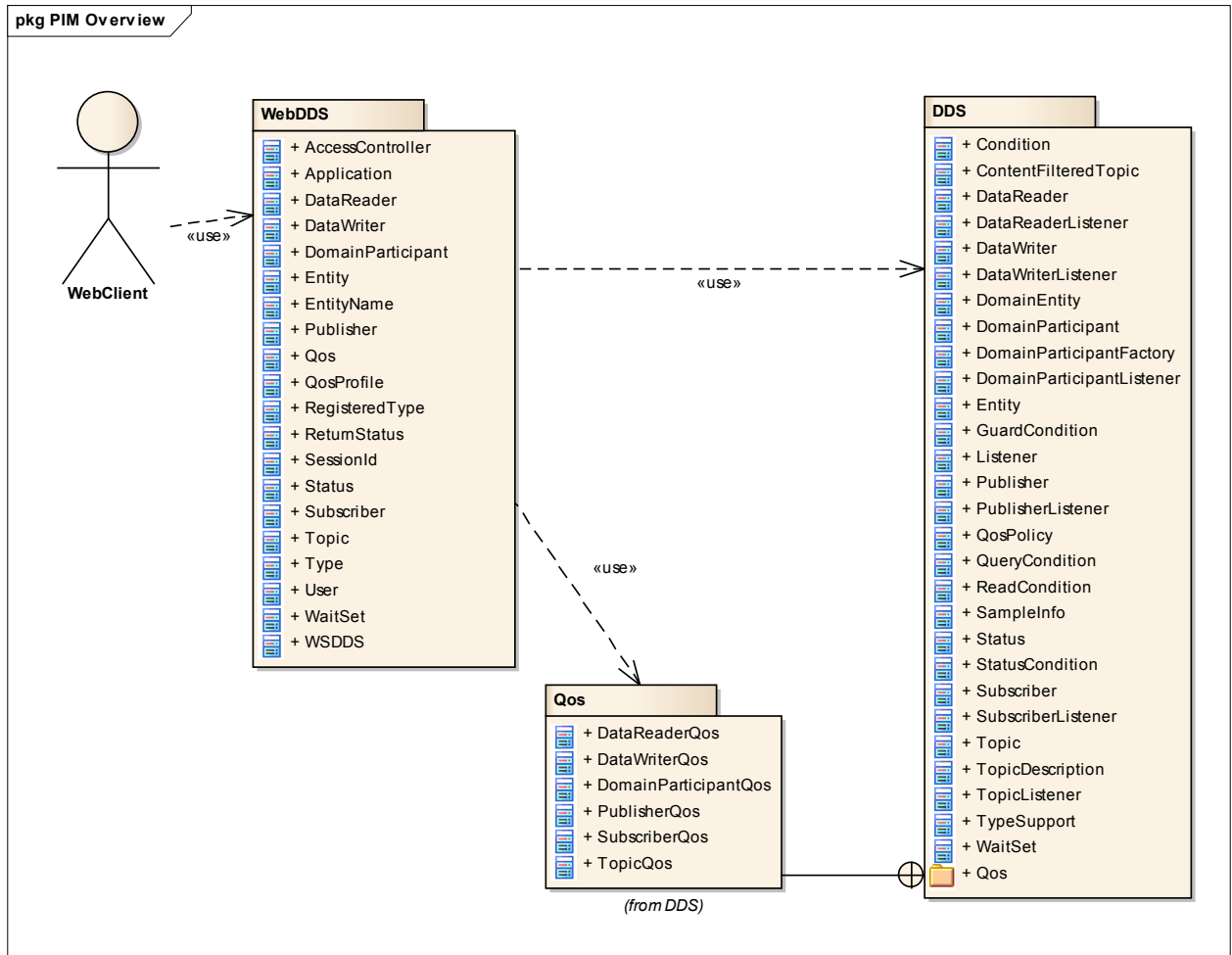


## 7 WebDDS Object Model

### 7.1 General

The WebDDS Object model acts as a façade to the Standard DDS Object model, exposing a simplified model to the Web Client applications. The model is simplified by (1) reducing the number of objects and data-types, by (2) reducing the number of operations, and by (3) making their use more regular. These simplifications make it better suited to being mapped to web architectures such as REST. In addition, the WebDDS Object model adds several new objects necessary to manage the clients, their durable connections to the WebDDS service, and the access rights they have.

All the operations described in the WebDDS PIM pertain the interaction of a client application with a single instantiation of the WebDDS service, identified by the HTTP (or HTTPS) URL used to reach the WebDDS service. The scope of all the operations is therefore limited to its actions on that one WebDDS service instance. Notwithstanding that, client applications may interact with each other despite connecting to different WebDDS service instances. These interactions would happen as a consequence of the WebDDS service instances creating and performing operations on DDS DomainParticipant entities, which exchange information in accordance to the DDS specification.

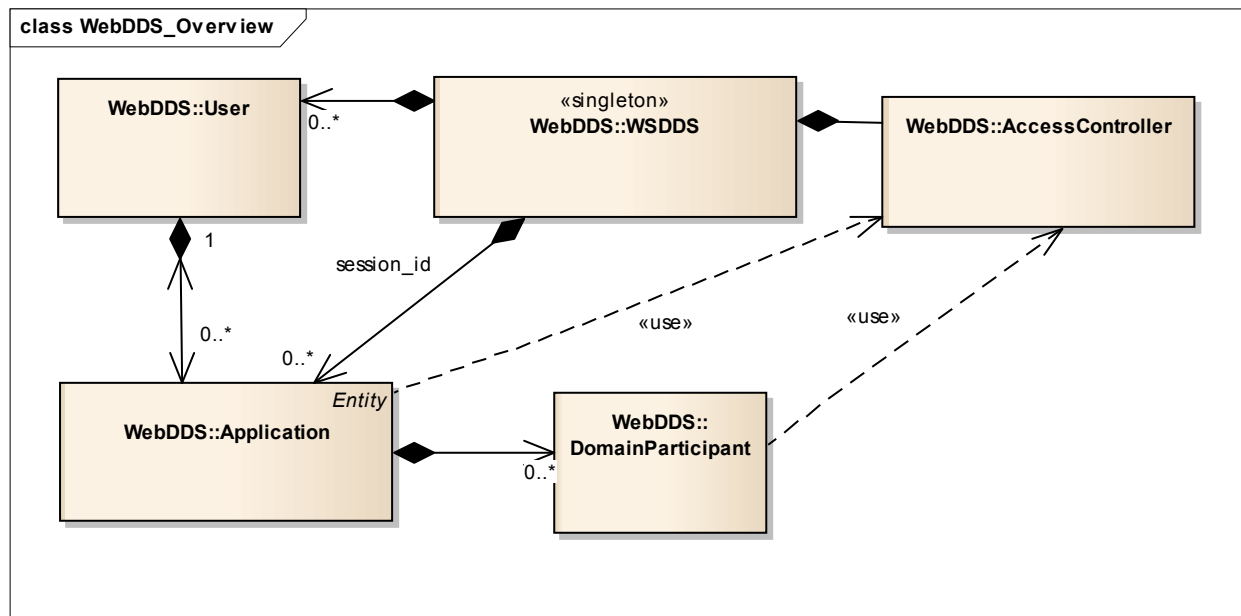


**Figure 3— WebDDS Object Model Overview**

*The WebDDS Object Model is contained in the package WebDDS and acts as a façade to the Standard DDS Object Model (from the DDS specification, contained in the DDS package).*

The remaining of this clause defines the WebDDS object model in detail and the interaction with entities in the Standard DDS Object Model.

## 7.2 Model Overview



**Figure 4—WebDDS Object Model Overview**

At the highest-level Web-Enabled DDS Object Model consists of 5 classes: The WSDDS singleton, User, Application, AccessController, and DomainParticipant.

The WSDDS singleton is the entry point for the service and functions as a root factory and container for all the Objects managed by the Web-Enabled DDS Service.

The User class models the user or principal that executes the client application. Each Application object is associated with a single User and gets its access rights from those assigned to the User.

The Application class models a software application that uses WebDDS service in order to publish and subscribe data on one or more DDS Domains. An Application can be associated with zero or more DomainParticipant objects.

The AccessController is responsible for making decisions regarding the resources and operations a particular Application is allowed to perform. It contains rules that associate Users with privileges which determine which DDS domain an application executing on behalf of a user may join, the DDS Topics it can read and write, etc.

The WebDDS DomainParticipant is a proxy for the DDS DomainParticipant and models the association with a DDS domain and the capability of the Application to publish and subscribe to Topics on that domain.

### 7.3 Users and Access Control

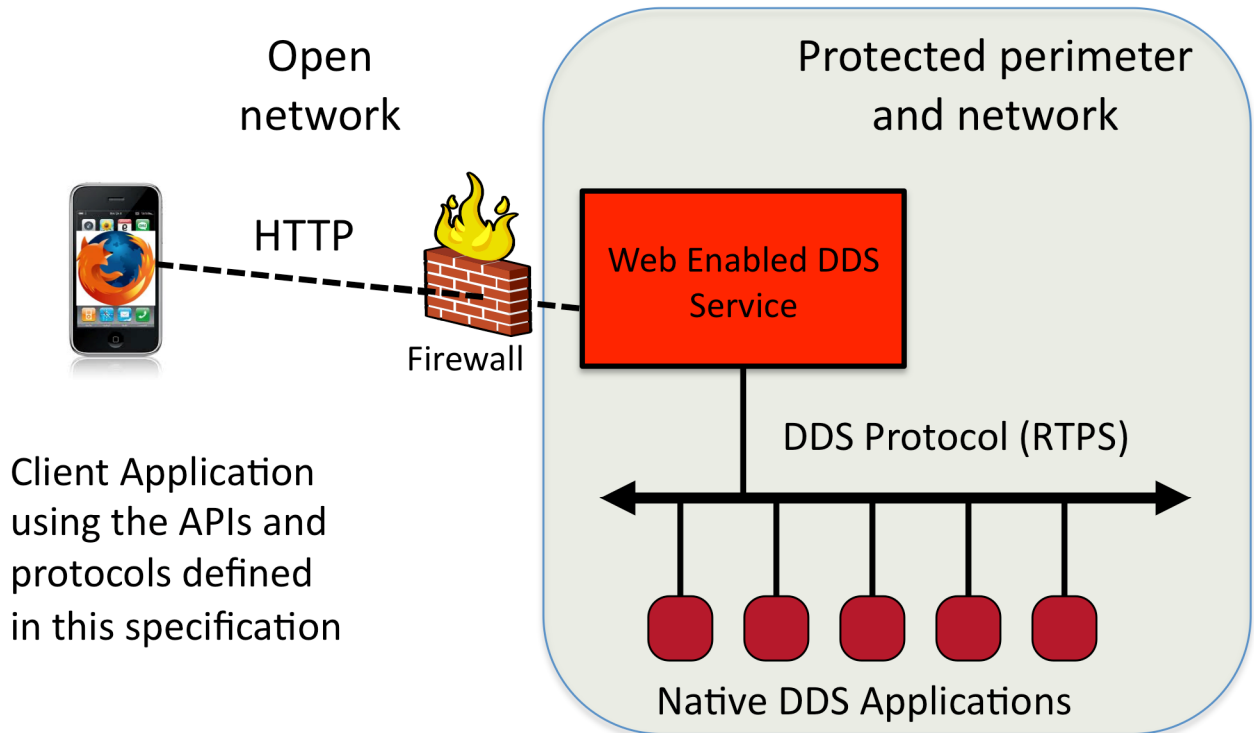
Many DDS applications are deployed within isolated or protected networks. In these situations security and access control can be managed outside the DDS infrastructure.

If DDS applications need to communicate directly over an open or unsecured network, then the DDS protocol itself needs to be secured. The DDS Security specification [21] addresses how to secure applications that use directly the DDS API and communicate using the DDS Interoperability wire protocol (DDS-RTPS).

The situation using Web-Enabled DDS is different. In this situation the security concerns are limited to the remote access from a client application to the Web Enabled DDS Service, which acts as a gateway to the DDS network (see Figure 5). This web-client-to-gateway communication uses standard web protocols (e.g. HTTP) and therefore the security mechanism must be well aligned with these protocols. Securing access from web clients to the Web Enabled DDS Service is orthogonal to securing the communications that use the DDS Interoperability wire protocol. For example the native DDS applications may all reside within a closed network protected by perimeter security (firewalls, NATs, and other network-level access control mechanisms), or the native DDS applications may use the security mechanisms eventually specified by the Secure DDS specification.

For this reason Web-Enabled DDS must provide its own security mechanism that defines how a web client is authenticated to the Web Enabled DDS Service, the access rights an authenticated client has to the entities within the Web Enabled DDS Service and how the HTTP communication is secured.

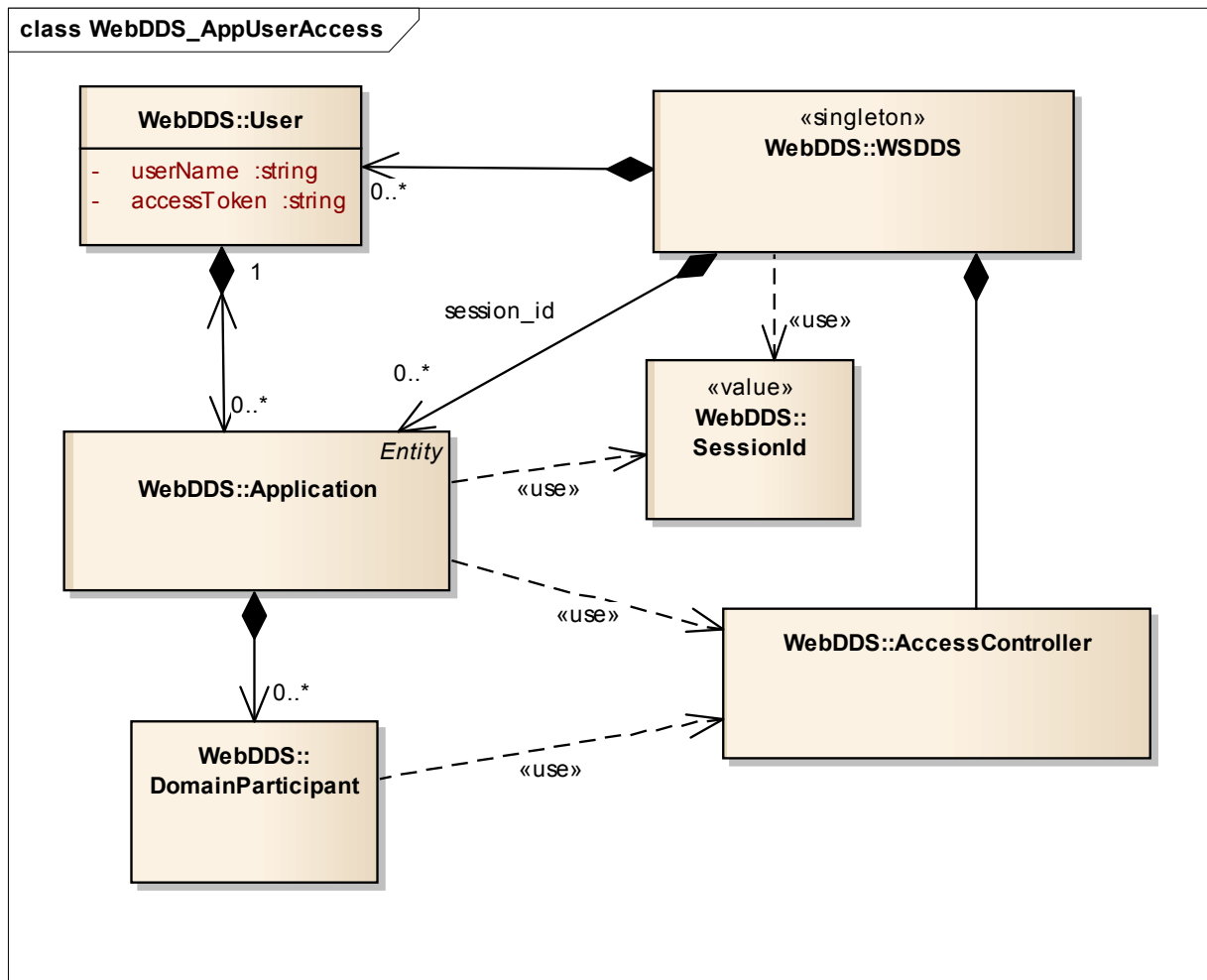




**Figure 5—Web Enabled DDS Service operating as gateway to Protected DDS domain**

The remainder of this clause provides a summary of the workflow. A more detailed description of each interface and its operation is provided in the sections that follow.

The WSDDS singleton is the entry point to the service. Client applications invoke the “login” operation on the WSDDS singleton in order to connect, authenticate and gain access to the service. The login operation receives the user credentials and the name of the application as an argument and the WSDDS uses that information to authenticate the user using the `AccessController` object. Upon successful authentication the WSDDS service instantiates an `Application` bound to that User and returns an “authenticatedSessionToken” value to the client, which is used to represent the authenticated user and connection.



**Figure 6—WebDDS classes involved in client-application, user, and application management**

To understand the purpose of the `Application` class, it is useful to go back to the traditional DDS Object Model. The DDS Object Model is intended for use with a local programming API. Using the DDS programming API applications create DDS entities, which inevitably get destroyed no later than when the application finishes. Therefore, DDS Entities’ lifetimes are contained within the application’s own. However, within a web-based distributed system this last assumption it is not always true. Web client applications may ask web servers to instantiate server-side entities and store their state on the server side. Web clients will potentially be disconnected and reconnected from the server. Such server-based entities may therefore have a lifetime that goes beyond a single client-server session. For this reason the management of server-based entities that survive temporary disconnects is an important issue addressed by this specification. This is precisely the purpose of the `Application` class.

The purpose of the `SessionId` type is to remember an authenticated client across subsequent invocations to operations on the service. This is needed so that each operation does not require re-authentication. By nature of the HTTP (or HTTPS) protocol, the web client connectivity to the

server is intermittent. Even under normal operating conditions successive client operations could close and re-establish the underlying TCP connections. For this reason the WebDDS maintains a Session concept that abstracts the *duration in which web clients are considered authenticated and bound to their created Applications*. During this period, client and server can be considered connected.

### 7.3.1 Class WebDDS::WSDDS

The WSDDS singleton directly manages three kinds of objects: Users, Applications, and AccessController. It serves as the entry point for the web client application. It provides operations that allow a client application connect to the service, authenticates the user (by delegating to the AccessController class), and creates Application objects.

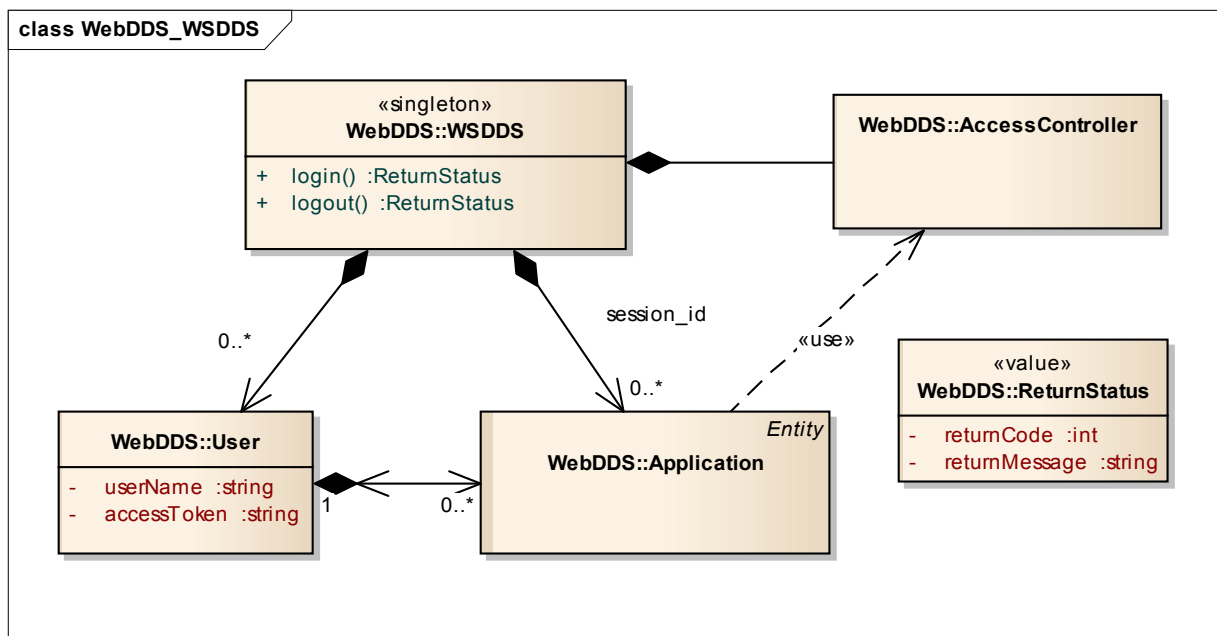


Figure 7—WSDDS class and operations

#### 7.3.1.1 Operation: login

##### Inputs

- `userName` (string): An identifier for the user to authenticate.
- `accessToken` (string): The credential on the basis of which the named user is to be authenticated. It may be simply a password or a more sophisticated token, such as the ones generated by an OATH2 Authorization Server.
- `applicationName` (string): An identifier for the Application (see 7.3.4 and 7.4.1) to create or attach to.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.
- `authenticatedSessionToken` (`AuthenticatedSessionToken`): If the operation succeeded, an identifier for the new session that has begun. If the operation failed, a sentinel value not corresponding to any valid session.

This operation authenticates a user based on a provided user name and access token and, if authentication is successful, it starts a new session, otherwise it returns an error.

The operation locates an existing `User` object with the specified `userName`. If such object is not found, it creates a new one.

The operation locates an existing `Application` object associated with the `User` object that has the specified `applicationName`. If such object is not found it creates a new one.

The `Application` object is bound to the `authenticatedSessionToken`.

This operation may be called multiple times with the same combination of `userName` and `applicationName` without any intervening calls to `logout`. Each such successful execution of this operation shall result in the invalidation of the prior `authenticatedSessionToken`, the creation of a new one and the binding of the new `authenticatedSessionToken` to the `Application`. As a consequence for a single `userName` at any given time there can only be one client application with an active session associated with the `Application`. A successful login operation by the same `userName` for the same `applicationName` executed from a different computer or connection with inactivate the existing session.

**Note:** The service limits each `userName` to have a single session for each `applicationName` in order to provide a sandbox for where each client application can perform operations without being affected by what others do. This does not introduce a limitation on what a client application can do because it would be possible for client applications to alternate the use of different user names, create the same underlying DDS application, and in this manner read and write the same DDS Topics.

### 7.3.1.2 Operation: logout

#### Inputs

- `authenticatedSessionToken` (`AuthenticatedSessionToken`): The session to terminate. This `authenticatedSessionToken` must have been obtained from a previous invocation of `Login`.
- `terminateApplication` (`boolean`): Indicates whether the client wants to also terminate the `Application` associated with the `authenticatedSessionToken`. If the application is not terminated, then the `User` can re-attach to it by providing the same

`applicationName` to the `Login` operation. If it is terminated, all the DDS objects associated with the application will also be deleted.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation disconnects the user from the `Application`. After it has completed, the `authenticatedSessionToken` that was passed to it becomes invalid and shall not be used to identify future sessions.

Disconnecting a `User` does not always imply the removal of the `Application`. If the `Application` is not removed, the related objects managed by the application remain in existence.

### 7.3.2 Class `WebDDS::AccessController`

The `AccessController` class is used to validate that the client application has the necessary privileges to join the DDS domain and perform the operations it requests.

This class is never used directly by the client applications. It is a class used internally by the Web-Enabled DDS service to make decisions as to allow or deny the requests originating from the client applications. For this reason the API or even the explicit existence of this class is not mandated by this specification. Implementers of this specification may choose to fold the functionality offered by this class into other classes or parts of their system and it will not be visible to the client applications.

Following is the normative behavior of the Web-Enabled DDS service related to access control:

- Provide a way to configure which users (identified by a name) are allowed to use the service
- Authenticate the client application (based on the credentials passed on the login operation) to ensure it represents the user identified by the `userName`.
- Provide a way to configure which DDS domains (identified by the numeric DDS `domainId`) each user (identified by the `userName`) is allowed to join.
- Reject any attempts of a client application to join a `domain` unless the associated `User` has been authenticated and given permission to join the `domain` in the service configuration
- Provide a way to configure which DDS Topics (identified by the `Topic` name) each `User` can publish on each DDS `domain`.
- Reject any attempts of a client application to publish to a `Topic` unless the associated `User` has been authenticated and given permission to publish that `topicName` on that `domainId` by the service configuration
- Provide a way to configure which DDS Topics (identified by the `Topic` name) each `User` can subscribe to on each DDS `domain`

- Reject any attempts of a client application to subscribe to a `Topic` unless the associated `User` has been authenticated and given permission to publish that `topicName` on that `domainId` by the service configuration

The specific means to configure the service (via file, tool, etc.) are left outside this specification, as they do not affect interoperability with client applications.

To understand the authentication and access control decisions that the `AccessController` shall make, it is modeled as having the operations: `authenticate_user` and `check_permissions`. The behavior of the Web-Enabled DDS service is explained in terms of these logical operations being invoked at the point where access control decisions should be made. As mentioned above this is only used for descriptive purposes. The implementation of the class is not normative, only the externally-observable behavior concerning the authentication and access control decisions are normative.

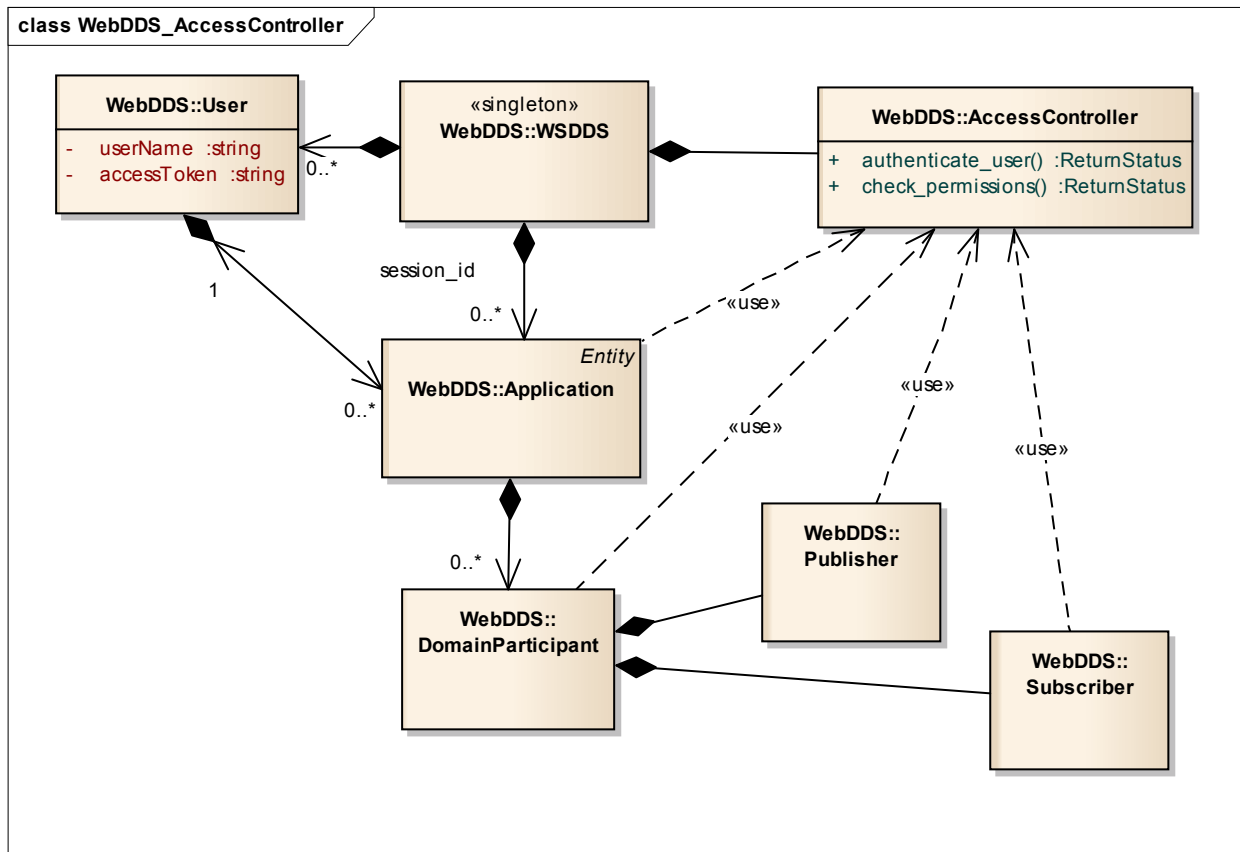


Figure 8—The `AccessController` class (conceptual only)

### 7.3.2.1 Operation: `authenticate_user`

#### Inputs

- `userName` (string): An identifier for the user to authenticate.

- `accessToken` (string): The credential on the basis of which the named user is to be authenticated.

#### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation is logically invoked when a client-application connects to the Web-Enabled DDS service. See 7.3.1.1

#### 7.3.2.2 Operation: `check_permissions`

##### Inputs

- `userName` (string): An identifier for the user performing the operation.
- `operationDescription` (string): A description of the kind of operation that is being performed.
- `operationDetails` (string sequence). Representation of the object on which the operation is being performed and any relevant parameters.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation is logically invoked each time the service must decide whether a particular operation is to be allowed. This specification explicitly documents which operations must perform this check on the description of the operation itself.

### 7.3.3 Class `WebDDS::User` (conceptual)

The `User` class models the user or principal that executes the client application. Each client application shall execute on behalf of a single `User`. The `login` operation on the `WSDDDS` class authenticates the client application and binds it to a single `User`. The Web-Enabled DDS service associates permissions or access rights to each `User` (see 7.3.2).

A `User` may create one or more `Application` objects. The `Application` objects represent specific client applications.

The `User` class does not have any operations and is therefore not used directly by the client applications. For this reason implementations may choose to realize it in a variety of ways, or combine its functionality with other classes. This specification does not mandate a specific implementation for this class, as long as its observable behavior matches what is described in this specification.

### **7.3.4 Class WebDDS::Application**

The `Application` class models a software application that uses WebDDS service in order to publish and subscribe data on one or more DDS Domains. Each `Application` object is bound with a single `User` and gets its access rights from those assigned to the `User`.

An `Application` can be associated with zero or more `DomainParticipant` objects.

The `Application` class operations are described in 7.4.



## 7.4 DDS Proxy classes

The WebDDS object model contains a set of “proxy” classes that collectively define an object model that is logically equivalent to the DDS Object Model. These proxy classes allow client applications to participate as first-class citizens on the DDS network. The client applications instantiate and operate in the proxy objects. Each WebDDS proxy object is backed by one (or more) DDS objects and operations performed on the proxies are delegated to the actual DDS objects.

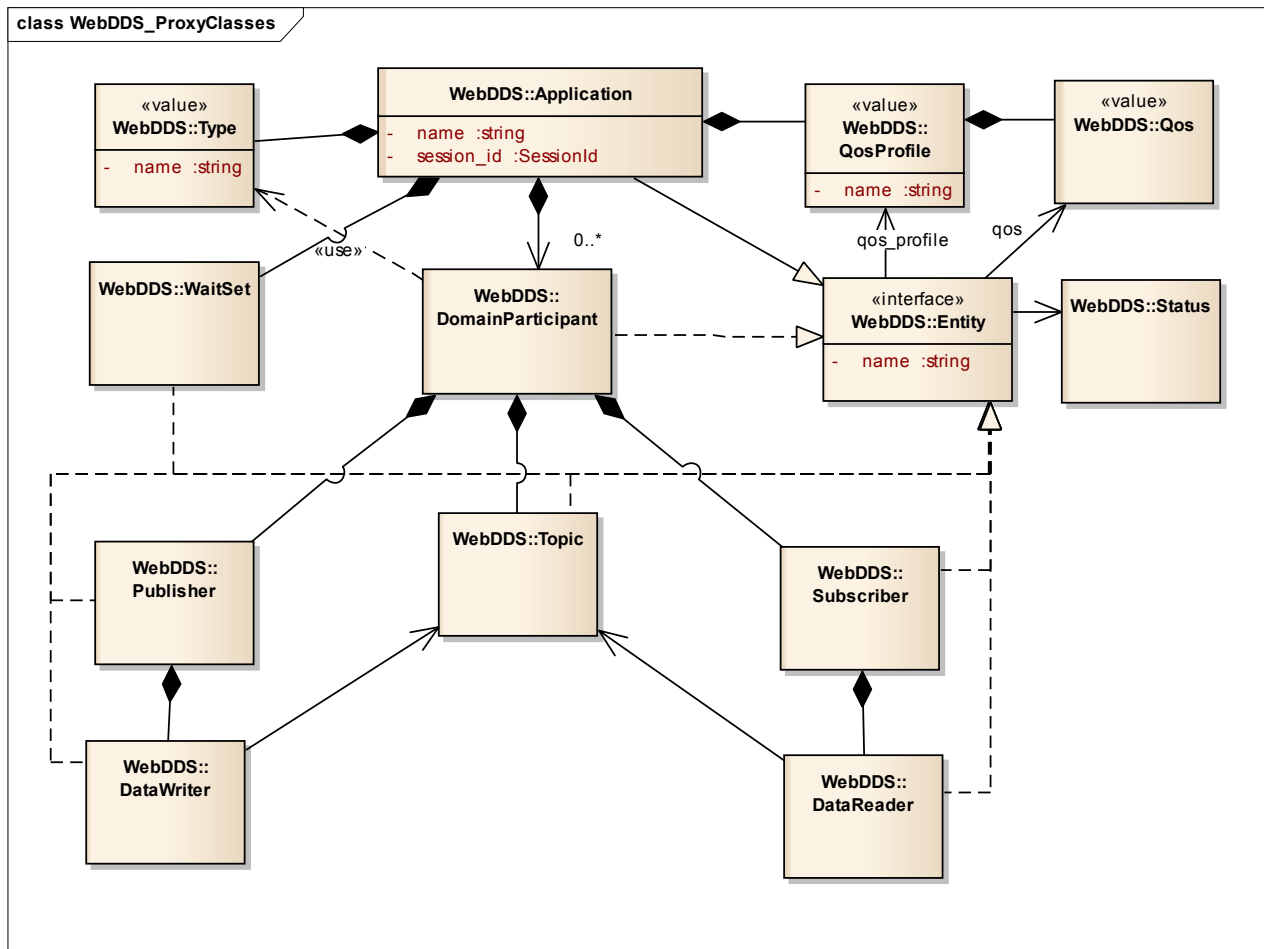


Figure 9—WebDDS classes used to proxy the DDS Objects

The proxy classes are designed as a simplification of the DDS Object Model to (a) reduce the number of classes and operations and (b) make the operation names and semantics more uniform across the different classes so that it can be more easily accessed as resources in a REST-style architecture. The high-level relationship between the proxy classes in the WebDDS Object Model and the corresponding classes in the DDS Object Model is illustrated in Figure 10.

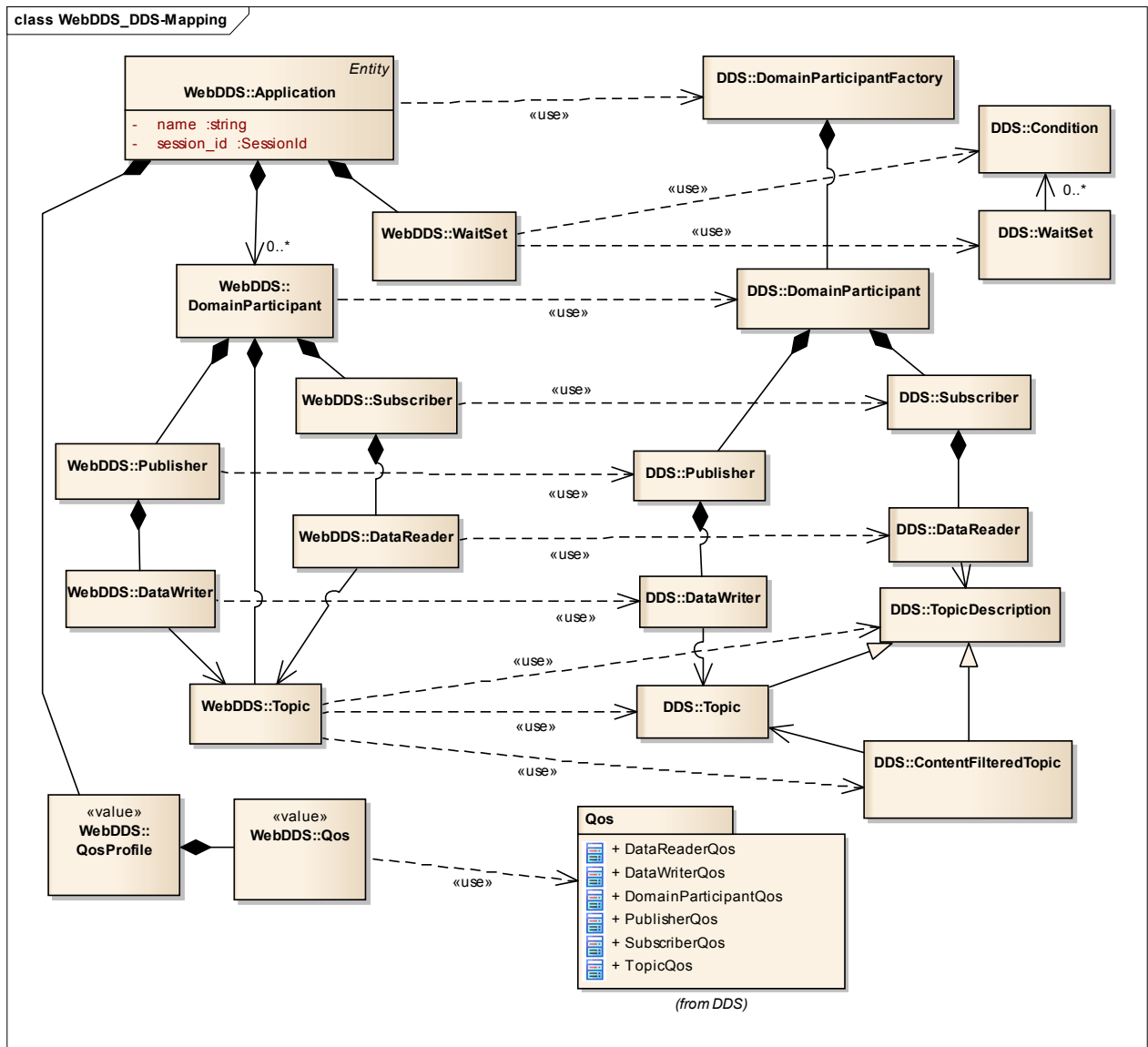


Figure 10—Mapping of WebDDS Object Model classes to DDS Object Model classes

### 7.4.1 ReturnStatus

The operations on the WebDDS objects return a ReturnStatus containing an integer ReturnCode specifying whether the operation succeeded and in the case of failure the reason for the failure. The set of possible ReturnCode values is shown in Table 3 below.

Table 3 ReturnCode values

Value	Meaning
OK	The operation succeeded

DDS_ERROR	An operation on one of the underlying DDS objects returned an error.
OBJECT_ALREADY_EXISTS	Request to create an already existing object
INVALID_INPUT	The parameters passed to the operation were incorrect/invalid
INVALID_OBJECT	Operation specified a non-existing or invalid Object
ACCESS_DENIED	The username and access token provided are invalid
INVALID_SESSION_ID	The authenticatedSessionToken is invalid
PERMISSIONS_ERROR	The operation is not permitted by the access control rules that apply to the user
GENERIC_SERVICE_ERROR	Unspecified error.

Specific PSMs may map the `GENERIC_SERVICE_ERROR` value into more specific `ReturnCode` values providing finer details for the PSM.

The description of the PIM operations specifies the `ReturnStatus` of each operation. As a shortcut the following terminology is used:

- The operation “returns OK.” This is a shortcut to specify that the operation shall return a `ReturnStatus` with the `returnCode` attribute set to the value `OK`.
- The operation “returns the XXX error.” This is a shortcut to specify that the operation shall return a `ReturnStatus` with the `returnCode` attribute set to the `XXX` error and the `returnMessage` attribute set to a textual description of the error.

## 7.4.2 Access control and permissions

The implementation of many of the operations must call `check_permissions` operation on the `WebDDS::AccessControl` class to determine whether the operation is allowed.

The actual representation and parameters to this operation need not be defined by this specification because this operation is invoked internally by the service and does not affect the observable client API and behavior. The only observable behavior by the client happens in its call to other operations of the `WebDDS` API, which may fail as a result of lack of permissions.

This specification states where the `check_permissions` operation shall be invoked in order to specify the operations that can potentially require access permissions. Strictly speaking the specifics of this call are not normative. Implementations may execute them in different order or combine them as long as the observable behavior is as specified.

### 7.4.3 Class WebDDS::Application (details)

This class represents a running client application and serves as the root factory for all the other objects instantiated by the Web-Enabled DDS service.

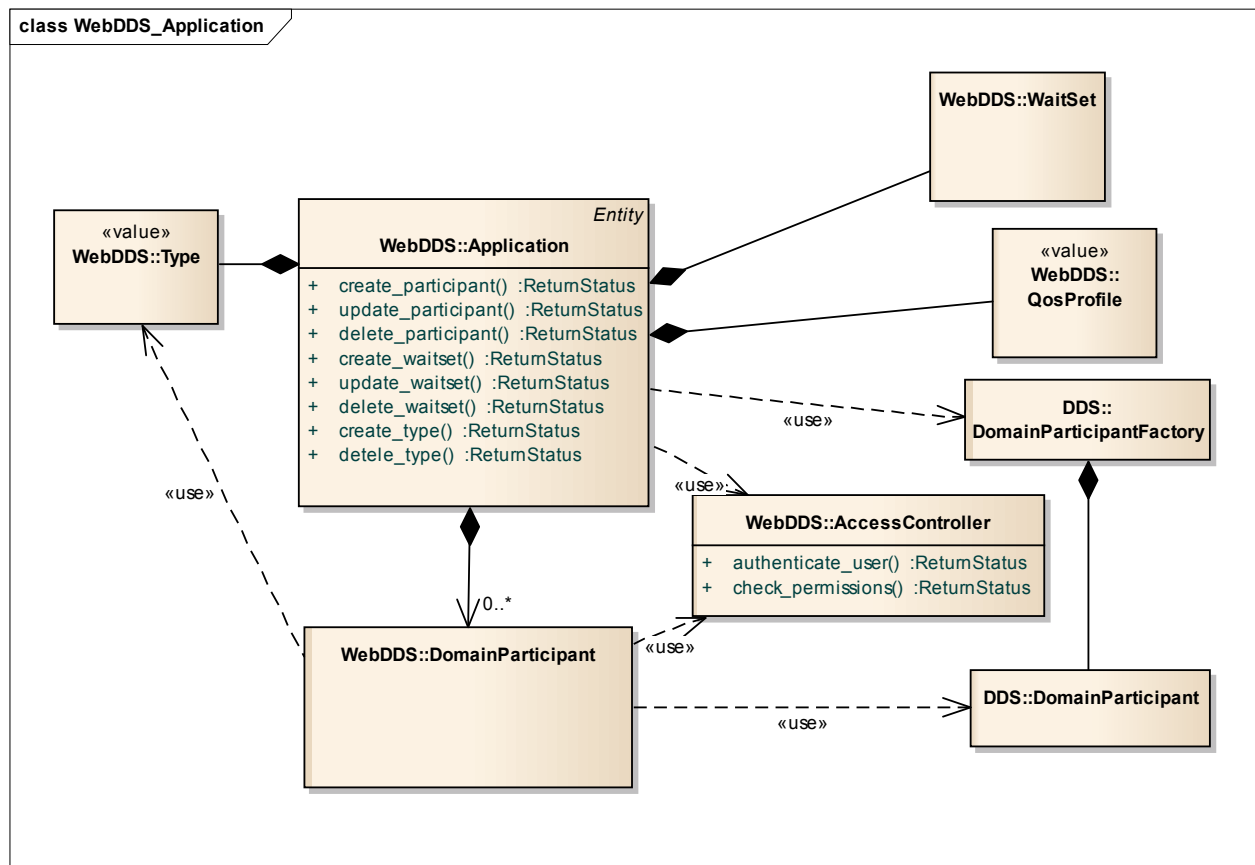


Figure 11—Application class with factory operations

#### 7.4.3.1 Operation: create\_participant

##### Inputs

- `participantObjectRepresentation` (string) a representation of the WebDDS Participant object including its name and optionally Qos and contained entities. The format of the representation shall be defined by each PSM. The name of the participant shall be unique within the scope of the Application object.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::DomainParticipant` of the specified `participantName` within the `WebDDS::Application`. If the `WebDDS::DomainParticipant` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to join the DDS domain identified by the `domainId` with the requested QoS. If the verification fails, the operation returns the `PERMISSIONS_ERROR` error.

If the ***participantObjectRepresentation*** specifies a set of contained entities, the `check_permissions` operation is invoked to verify that the user is allowed by the access control policies to create those entities with their specified QoS. If the verification fails for any of them, the `WebDDS::DomainParticipant` is not created and the operation returns the `PERMISSIONS_ERROR` error.

If the permissions checks succeed, the operation creates a `WebDDS::DomainParticipant` which in turn creates a `DDS::DomainParticipant` on the requested ***domainId*** using the specified QoS. It then creates all the `WebDDS` entities specified as part of the ***participantObjectRepresentation*** and their corresponding DDS Entities.

Each of the DDS Entities is created disabled. If the creation of any DDS Entity fails, then all the created objects are destroyed and the operation returns the `DDS_ERROR` error.

If all the creations are successful, the `DDS::DomainParticipant` and all contained entities are enabled and the operation returns OK.

#### 7.4.3.2 Operation: update\_participant

##### Inputs

- `participantObjectRepresentation` (string) a representation of the `WebDDS Participant` object including its name and optionally QoS and contained entities. The format of the representation shall be defined by each PSM. The name of the participant shall correspond to a previously created participant within the `Application` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It locates a `WebDDS::DomainParticipant` associated with the user with the specified `participantName`. If the participant does not exist, it returns the `INVALID_OBJECT` error.

If the ***participantObjectRepresentation*** specifies a QoS or QoSProfile the

`check_permissions` operation to verify that the user is allowed by the access control policies to change the `DDS::DomainParticipant` QoS. If the verification is successful, it updates the QoS of the `DomainParticipant`. Otherwise it returns the `PERMISSIONS_ERROR` error.

If the ***participantObjectRepresentation*** specifies a set of contained entities, then the operation checks if these contained entities already exist within the `WebDDS::DomainParticipant`.

- For each contained entity that already exists if the ***participantObjectRepresentation*** specifies a QoS the operation shall call the `check_permissions` operation to verify that the user is allowed by the access control policies to change the QoS of that entity.
- For each contained entity that does not exist the operation shall call the `check_permissions` operation to verify that the user is allowed by the access control policies to create that entity and set its QoS as specified.
- The above two steps are repeated recursively as a contained entity (such as a Publisher) may itself contain other entities (such as the DataWriters)

The operation checks if any of the entities contained within the `WebDDS::DomainParticipant` are not present in the `participantObjectRepresentation`. For any such entities, the operation calls `check_permissions` operation to verify that the user is allowed by the access control policies to delete that entity.

If any of the calls to `check_permissions` fails, any actions performed by this operation are undone and the operation returns the `PERMISSIONS_ERROR` error.

If all the calls to `check_permissions` succeed, the operation performs the appropriate actions in terms of

- (a) Creating the `WebDDS` objects specified in the `participantObjectRepresentation`. This creates any associated `DDS` Objects.
- (b) Changing the QoS of the `DDS` Objects associated with previously existing objects
- (c) Deleting the `WebDDS` in the `WebDDS::DomainParticipant` which do not appear in the `participantObjectRepresentation`.

If any of the above creation, deletion, or QoS-setting operations fails, any actions performed by this operation are undone and the operation returns the `DDS_ERROR` error.

If all the creation or QoS-setting operations succeed, the operation returns `OK`.

### 7.4.3.3 Operation: `delete_participant`

#### Inputs

- `participantName` (string): The name of the participant.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

Deletes an existing `WebDDS::DomainParticipant`. This operation performs the following logical steps:

It locates a `WebDDS::DomainParticipant` associated with the `User` with the specified ***participantName***. If the participant does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the `DDS::DomainParticipant` QoS. If the check fails, it returns `PERMISSIONS_ERROR`.

If the verification is successful, it deletes the `DDS::DomainParticipant` associated with the `WebDDS::DomainParticipant`. If this deletion fails it returns the `DDS_ERROR` error.

It deletes the `WebDDS::DomainParticipant`. If this fails, it returns `GENERIC_SERVICE_ERROR`, otherwise the operation returns OK.

### 7.4.3.4 Operation: `create_type`

## Inputs

- `typeLibraryRepresentation` (string): The XML representation of a collection of modules, declarations, and data-types (see [DDS-XTYPES]).
- `typeName` (string): The name that specifies a unique data-type within the `typeLibraryRepresentation` (see [DDS-XTYPES]).

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Type` of the specified `typeName` within the Application. If the `WebDDS::Type` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It creates a `WebDDS::Type` that holds the ***typeLibraryRepresentation*** and ***typeName***. If the creation of the `Type` fails, it returns the `INVALID_INPUT` error.

It instantiates the `DDS::DynamicType` associated with the specified data-type as defined in the [DDS-XTYPES] specification and associates it with the `WebDDS::Type`. If this fails, it returns the `DDS_ERROR` error and deletes the `WebDDS::Type` if it had been just created by the operation.

Otherwise it returns OK.

#### 7.4.3.5 Operation: delete\_type

##### Inputs

- `typeName` (string): The name of the data type. This name must be unique within the scope of the `Application` object.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It locates a pre-existing `WebDDS::Type` of the specified `typeName` within the `Application`. If the `Type` is not found, it returns the `INVALID_OBJECT` error.

It deletes the located `WebDDS::Type` and the associated `DDS::DynamicType` object.

#### 7.4.3.6 Operation: get\_type

##### Inputs

- `typeName` (string): The name of a type specified when it was created

##### Outputs

- `typeRepresentation` (string): The XML representation of the data-type, which includes the name of the type (see [DDS-XTYPES]).
- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Type` of the specified `typeName` within the `Application`. If the `WebDDS::Type` does not exist, it returns the `INVALID_OBJECT` error.

It returns the ***typeRepresentation*** of the data-type associated with the `WebDDS::Type` and returns OK.

#### 7.4.3.7 Operation: create\_waitset

##### Inputs

- `waitsetName` (string): The name of the `WaitSet`. This name must be unique within the scope of the `Application` object.



- `waitsetRepresentation` (string): A string representation of the `WebDDS::WaitSet` which includes the name of the `WaitSet` and the list of conditions associated with it. The format of the representation shall be defined by each PSM.

#### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::WaitSet` of the specified `waitsetName` within the Application. If the `WaitSet` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It creates a `WebDDS::WaitSet` which in turn creates a `DDS::WaitSet`.

It creates the necessary DDS conditions, as specified in the `waitsetRepresentation` and attaches them to the `DDS::WaitSet`.

If any of the DDS related operations fails, it returns the `DDS_ERROR` error. Otherwise it returns `OK`.

#### 7.4.3.8 Operation: `update_waitset`

##### Inputs

- `waitsetRepresentation` (string): An XML representation of the `WebDDS::WaitSet` including the name and new conditions to associate with the `WaitSet`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::WaitSet` of the specified `waitsetName` within the Application. If the `WebDDS::WaitSet` already exists, it returns the `INVALID_OBJECT` error.

It updates the conditions associated with the `WebDDS::WaitSet` which in turn updates the corresponding `DDS::WaitSet`.

If the update operation fails due to a badly formatted `waitsetRepresentation`, it returns `INVALID_OBJECT` error. If it fails due to an error returned by the DDS operation, it returns the `DDS_ERROR` error. Otherwise it returns `OK`.

#### 7.4.3.9 Operation: delete\_waitset

##### Inputs

- `waitsetName` (string): The name of the participant. This name must be unique within the scope of the `Application` object.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It locates a pre-existing `WebDDS::WaitSet` of the specified `waitsetName` within the `Application`. If the `WebDDS::WaitSet` is not found, it returns the `INVALID_OBJECT` error.

It deletes the located `WebDDS::WaitSet` which in turn deletes the related `DDS::WaitSet`. If the delete fails due to an error returned by the DDS delete operation, it returns the `DDS_ERROR`. Otherwise it returns `OK`.

#### 7.4.4 Class `WebDDS::DomainParticipant`

This class is a proxy for a DDS `DomainParticipant` and serves as the factory for the `WebDDS Topic`, `Publisher`, and `Subscriber` objects.

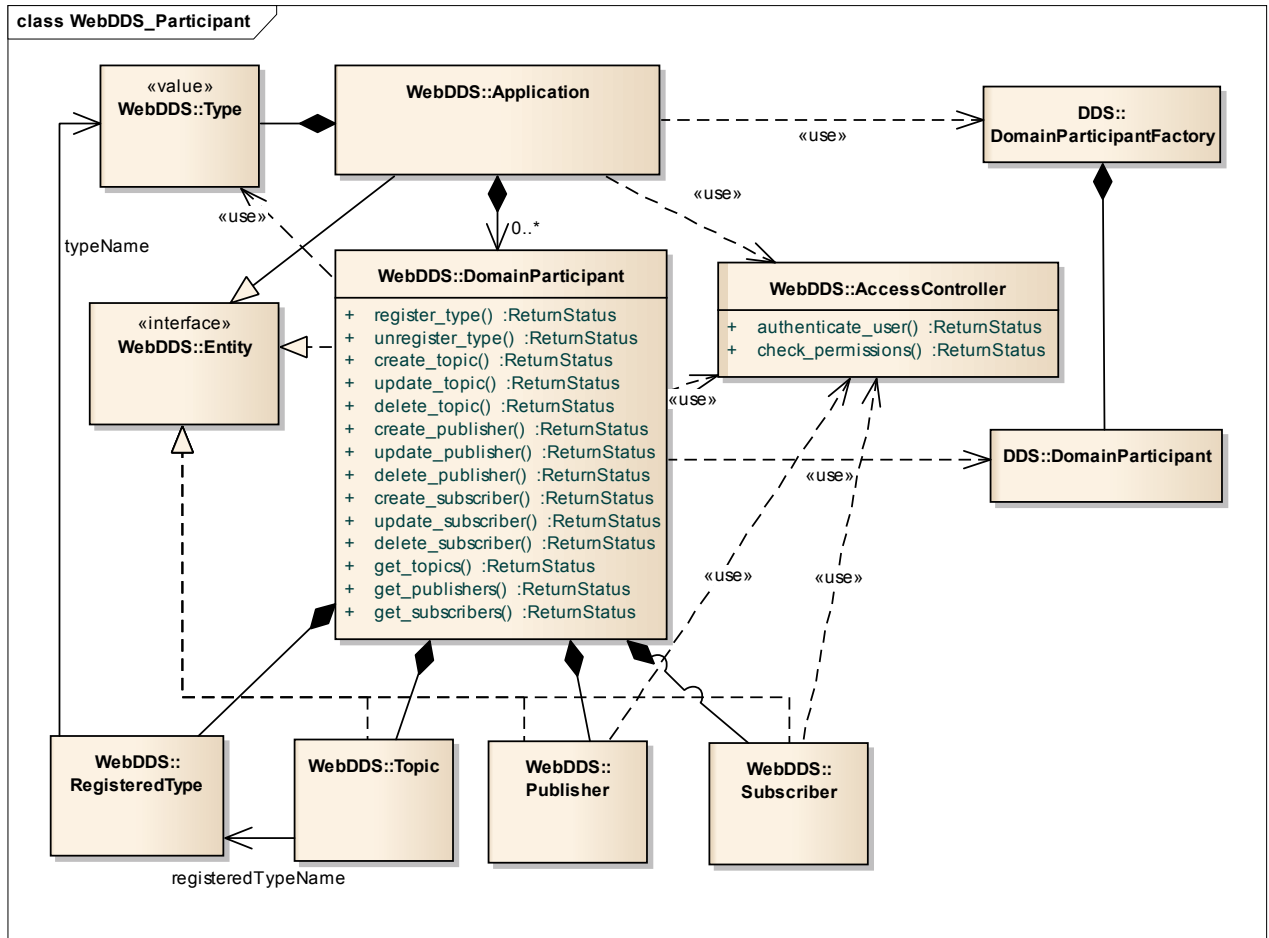


Figure 12—Participant class with operations

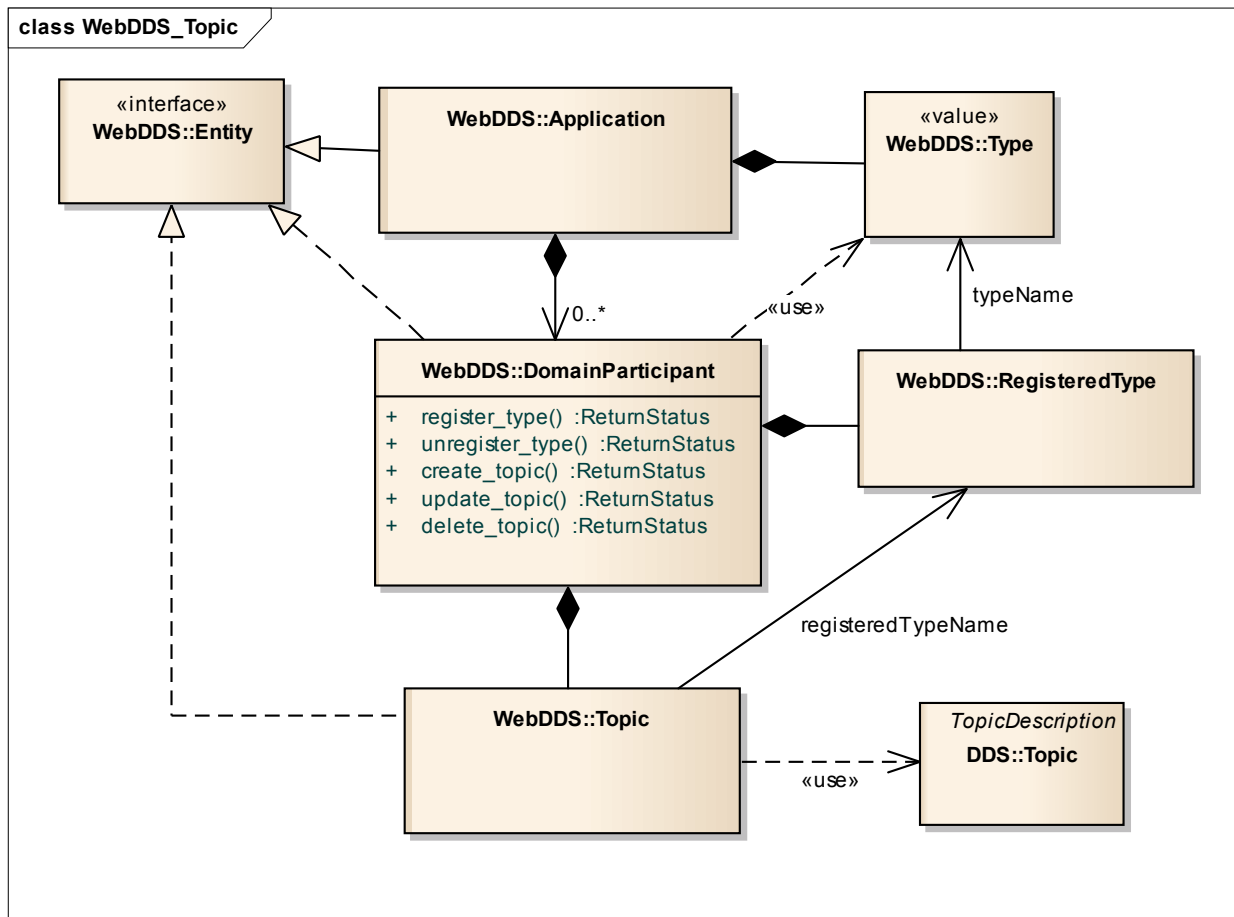


Figure 13—Participant class operations related to types and topics

#### 7.4.4.1 Operation: register\_type

##### Inputs

- `registeredTypeName` (string): The name the `DDS::DomainParticipant` should use to refer to this type.
- `relatedTypeName` (string): The name of the type as specified on a previous successful call to `Application::create_type`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Type` of the specified `typeName` within the `WebDDS Application`. If the `Type` does not exist, it returns the `INVALID_OBJECT` error.

It checks if the associated `DDS::DomainParticipant` already has a type registered under the name `registeredTypeName` and if this is the case, it returns the `OBJECT_ALREADY_EXISTS` error.

It uses the `DynamicType` facility defined by the DDS-XTYPES specification to create a `DDS TypeSupport` for the type represented by *registeredTypeName* and registers it with the associated `DDS::DomainParticipant`. If either operation fails, it returns the `DDS_ERROR` error.

If the operation succeeds, it returns OK.

#### 7.4.4.2 Operation: `unregister_type`

##### Inputs

- `registeredTypeName` (string): The `registeredTypeName` of a previously registered type as specified on a previous successful call to `register_type`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation locates a `WebDDS::Type` within the `WebDDS::DomainParticipant` with the specified *registeredTypeName*. If the `WebDDS::Type` does not exist, it returns the `INVALID_OBJECT` error.

The operation undoes the actions taken by the `register_type` operation removing the `WebDDS::Type` matching the *registeredTypeName*. Future calls that refer to a `WebDDS::Type` registered under the name *registeredTypeName* on that Participant shall return an error. The exception is a new call to the `register_type` operation.

Calling the operation `unregister_type` performs no operations on the underlying `DDS::DomainParticipant`.

If the operation succeeds it returns OK

#### 7.4.4.3 Operation: `create_topic`

##### Inputs

- `topicObjectRepresentation` (string) a representation of the `WebDDS Topic` object including the Topic name (*topicName*), name of the registered type it is associated with, and optionally a QoS. The format of the representation shall be defined by each PSM. The *topicName* of the Topic shall be unique within the scope of the Participant object.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Topic` of the specified name within the `DDS::DomainParticipant`. If the Topic already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It calls the `check_permissions` operation to verify that the User is allowed by the access control policies to create a DDS Topic of the specified name on DDS domain associated with the `WebDDS::Application`. If this fails, it returns the `PERMISSIONS_ERROR` error.

It checks that the `WebDDS::DomainParticipant` has a type registered with the specified *registeredTypeName*. If the `DomainParticipant` does not have a type registered under the name *registeredTypeName*, it returns the `INVALID_OBJECT` error.

It creates a `WebDDS::Topic`. If this fails it returns `GENERIC_SERVICE_ERROR`

It uses the associated `DDS::DomainParticipant` to create a `DDS::Topic` with the associated *topicName* and type *registeredTypeName*. If the `DDS::DomainParticipant` does not have a type registered under the name *registeredTypeName* or if the call to `create_topic` fails for any other reason, the operation return the `DDS_ERROR` error.

If the operation succeeds, it returns OK.

### 7.4.4.4 Operation: update\_topic

#### Inputs

- `topicObjectRepresentation` (string) a representation of the `WebDDS Topic` object including the Topic name (*topicName*) and optionally a qos or `QosProfile`. The format of the representation shall be defined by each PSM. The *topicName* of the Topic shall be unique within the scope of the `DomainParticipant` object.

#### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It locates a `WebDDS::Topic` within the `WebDDS::DomainParticipant` with the specified *topicName*. If the Topic does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to change the `DDS::Topic` QoS. If the verification is successful, it updates the

QoS of the `Topic`. Otherwise it returns the `PERMISSIONS_ERROR` error.

It changes the QoS of the `DDS::Topic`. If the operation fails, it returns the `DDS_ERROR` error. Otherwise it returns OK.

#### 7.4.4.5 Operation: `delete_topic`

##### Inputs

- `topicName` (string): The name of the `Topic`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It locates a `WebDDS::Topic` within the `WebDDS::DomainParticipant` with the specified *topicName*. If the `Topic` does not exist, it returns the `INVALID_OBJECT` error.

The operation calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the `DDS Topic` that `topicName`. If the verification is not successful, it returns the `PERMISSIONS_ERROR` error.

It locates and deletes the `DDS::Topic` with name *topicName* within the `DDS::DomainParticipant` associated with the `WebDDS::DomainParticipant`. If the `DDS::Topic` cannot be located or the operation fails, it returns the `DDS_ERROR` error. Otherwise it returns OK.

#### 7.4.4.6 Operation: `get_topics`

##### Inputs

- `topicNameExpression` (string): An expression on the name of the `Topic`.
- `registeredTypeNameExpression` (string): An expression on the type of the `Topic`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.
- `topicRepresentationList` (string): An XML representation of the list of `Topics` whose name matches the `topicNameExpression`. The format of the representation shall be defined by each PSM.

This operation returns the list of topic names whose name matches the `topicNameExpression` and type matches the `registeredTypeNameExpression`. If

the operation fails, it returns `GENERIC_SERVICE_ERROR`, otherwise it returns `OK`.

Expression syntax and matching for the *topicNameExpression* and *typeNameExpression* shall use the syntax and rules of the POSIX `fnmatch()` function as specified in POSIX 1003.2-1992, section B.6 [19].

#### 7.4.4.7 Operation: `create_publisher`

##### Inputs

- `publisherObjectRepresentation` (string) a representation of the `WebDDS::Publisher` object including its *publisherName* and optionally QoS and contained entities. The format of the representation shall be defined by each PSM. The name of the `Publisher` shall be unique within the scope of the `Participant` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation creates a `WebDDS::Publisher` and the associated `DDS::Publisher` with the desired QoS policies and contained entities.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Publisher` of the specified `publisherName` within the `WebDDS::DomainParticipant`. If the `WebDDS::Publisher` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

If the *publisherObjectRepresentation* specifies a set of contained entities, the `check_permissions` operation is invoked to verify that the user is allowed by the access control policies to create those entities with their specified QoS. If the verification fails for any of them, the `WebDDS::Publisher` is not created and the operation returns the `PERMISSIONS_ERROR` error.

If the permissions checks succeed, the operation creates a `WebDDS::Publisher` which in turn creates a `DDS::Publisher` using the specified QoS. It then creates all the `WebDDS` entities specified as part of the *publisherObjectRepresentation* and their corresponding `DDS` Entities.

Each of the `DDS` Entities is created disabled. If the creation of any `DDS Entity` fails, then all the created objects are destroyed and the operation returns the `DDS_ERROR` error.

If all the creations are successful, the `DDS::Publisher` and all contained entities are enabled and the operation returns `OK`.



#### 7.4.4.8 Operation: update\_publisher

##### Inputs

- `publisherObjectRepresentation` (string) a representation of the WebDDS `Publisher` object including its ***publisherName*** and optionally QoS and contained entities. The format of the representation shall be defined by each PSM. The name of the `Publisher` shall correspond to a previously-created `Publisher` within the `DomainParticipant` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation updates the QoS and contained entities of an existing `Publisher`.

This operation performs the following logical steps:

It locates a `WebDDS::Publisher` within the `WebDDS::DomainParticipant` with the specified ***publisherName***. If the `WebDDS::Publisher` does not exist, it returns the `INVALID_OBJECT` error.

If the `publisherObjectRepresentation` specifies a QoS or `QoSProfile`, the `check_permissions` operation to verify that the user is allowed by the access control policies to change the `DDS::Publisher` QoS. If the verification is successful, it updates the QoS of the `DDS::Publisher`. Otherwise it returns the `PERMISSIONS_ERROR` error.

If the `publisherObjectRepresentation` specifies a set of contained entities (`DataWriter` objects,) then the operation checks if these contained entities already exist.

- For each contained entity that already exists if the ***publisherObjectRepresentation*** specifies a QoS the operation calls `check_permissions` operation to verify that the user is allowed by the access control policies to change the QoS of that entity.
- For each contained entity that does not exist the operation calls `check_permissions` operation to verify that the user is allowed by the access control policies to create that entity and set its QoS as specified.

The operation checks if any of the entities contained within the `WebDDS::Publisher` are not present in the ***publisherObjectRepresentation***. For any such entities, the operation calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete that entity.

If any of the calls to `check_permissions` fails, any actions performed by this operation are undone and the operation returns the `PERMISSIONS_ERROR` error.

If all the calls to `check_permissions` succeed, the operation performs the appropriate actions

in terms of

- a) Creating the WebDDS objects specified in the `publisherObjectRepresentation`. This creates any associated DDS Objects.
- b) Changing the QoS of the DDS Objects associated with previously existing objects.
- c) Deleting the WebDDS entities in the `WebDDS::Publisher` which do not appear in the `participantObjectRepresentation` and their peer objects on the associated `DDS::Publisher`.

If any of the above creation, deletion, or QoS-setting operations fails, any actions performed by this operation are undone and the operation returns the `DDS_ERROR` error.

If all the creation or QoS-setting operations succeed, the operation returns OK.

#### 7.4.4.9 Operation: `delete_publisher`

##### Inputs

- `publisherName` (string): The name of the Publisher.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

Deletes an existing `WebDDS::Publisher` and the associated `DDS::Publisher`. This operation performs the following logical steps:

It locates a `WebDDS::Publisher` within the `WebDDS::DomainParticipant` with the specified ***publisherName***. If the `WebDDS::Publisher` does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the entities contained within the `WebDDS::Publisher`. If the check fails, it returns `PERMISSIONS_ERROR`.

If the verification is successful, it deletes the `DDS::Publisher` associated with the `WebDDS::Publisher`. If this deletion fails it returns the `DDS_ERROR` error.

It deletes the `WebDDS::Publisher`. If this fails, it returns `GENERIC_SERVICE_ERROR`, otherwise the operation returns OK.

#### 7.4.4.10 Operation: `get_publishers`

##### Inputs

- `publisherNameExpression` (string): An expression on the name of the Publisher.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.
- `publisherRepresentationList` (string): An XML representation of the list of `Publisher` objects whose name matches the `publisherNameExpression`. The format of the representation shall be defined by each PSM.

This operation returns a representation of the list of all the `WebDDS::Publisher` objects belonging to the `WebDDS::DomainParticipant` whose name matches the ***publisherNameExpression***. If the operation fails, it returns `GENERIC_SERVICE_ERROR`, otherwise it returns `OK`.

Expression syntax and matching for the ***publisherNameExpression*** shall use the syntax and rules of the POSIX `fnmatch()` function as specified in POSIX 1003.2-1992, section B.6 [19].

### 7.4.4.11 Operation: `create_subscriber`

#### Inputs

- `subscriberObjectRepresentation` (string) a representation of the `WebDDS` `Publisher` object including its ***name*** and optionally `Qos` and contained entities. The format of the representation shall be defined by each PSM. The name of the `Subscriber` shall be unique within the scope of the `Participant` object.

#### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation creates a `WebDDS::Subscriber` and the associated `DDS::Subscriber` with the desired `QoS` policies and contained entities.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::Subscriber` of the specified ***publisherName*** within the `WebDDS::DomainParticipant`. If the `WebDDS::Subscriber` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

If the ***subscriberObjectRepresentation*** specifies a set of contained entities, the `check_permissions` operation is invoked to verify that the user is allowed by the access control policies to create those entities with their specified `QoS`. If the verification fails for any of them, the `WebDDS::Subscriber` is not created and the operation returns the `PERMISSIONS_ERROR` error.

If the permissions checks succeed, the operation creates a `WebDDS::Subscriber` which in turns creates a `DDS::Subscriber` using the specified `QoS`. It then creates all the `WebDDS`

entities specified as part of the *subscriberObjectRepresentation* and their corresponding DDS Entities.

Each of the DDS Entities is created disabled. If the creation of any DDS Entity fails then all the created objects are destroyed and the operation returns the `DDS_ERROR` error.

If all the creations are successful the `DDS::Subscriber` and all contained entities are enabled and the operation returns `OK`.

#### 7.4.4.12 Operation: update\_subscriber

##### Inputs

- `subscriberObjectRepresentation` (string) a representation of the `WebDDS Subscriber` object including its *subscriberName* and optionally QoS and contained entities. The format of the representation shall be defined by each PSM. The name of the `Subscriber` shall correspond to a previously-created `WebDDS::Subscriber` within the `WebDDS::Participant` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation updates the QoS and contained entities of an existing `WebDDS::Subscriber`.

This operation performs the following logical steps:

It locates a `WebDDS::Subscriber` within the `WebDDS::DomainParticipant` with the specified *subscriberName*. If the `WebDDS::Subscriber` does not exist, it returns the `INVALID_OBJECT` error.

If the *subscriberObjectRepresentation* specifies a QoS or `QoSProfile`, the `check_permissions` operation is invoked to verify that the user is allowed by the access control policies to change the `DDS::Subscriber` QoS. If the verification is successful, it updates the QoS of the `DDS::Subscriber`. Otherwise it returns the `PERMISSIONS_ERROR` error.

If the `subscriberObjectRepresentation` specifies a set of contained entities (`DataReader` objects,) then the operation checks if these contained entities already exist.

- For each contained entity that already exists if the *subscriberObjectRepresentation* specifies a QoS the operation calls `check_permissions` operation to verify that the user is allowed by the access control policies to change the QoS of that entity.
- For each contained entity that does not exist the operation calls `check_permissions` operation to verify that the user is allowed by the access control policies to create that entity and set its QoS as specified.

The operation checks if any of the entities contained within the `WebDDS::Subscriber` are not

present in the *subscriberObjectRepresentation*. For any such entities, the operation calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete that entity.

If any of the calls to `check_permissions` fails, any actions performed by this operation are undone and the operation returns the `PERMISSIONS_ERROR` error.

If all the calls to `check_permissions` succeed, the operation performs the appropriate actions in terms of

- a) Creating the WebDDS objects specified in the *subscriberObjectRepresentation*. This creates any associated DDS Objects.
- b) Changing the QoS of the DDS Objects associated with previously existing objects.
- c) Deleting the WebDDS entities in the `WebDDS::Subscriber` which do not appear in the *subscriberObjectRepresentation* and their peer objects on the associated `DDS::Subscriber`.

If any of the above creation, deletion, or QoS-setting operations fails, any actions performed by this operation are undone and the operation returns the `DDS_ERROR` error.

If all the creation or QoS-setting operations succeed, the operation returns OK.

#### 7.4.4.13 Operation: `delete_subscriber`

##### Inputs

- `subscriberName` (string): The name of the `Subscriber`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

Deletes an existing `WebDDS::Subscriber`. This operation performs the following logical steps:

It locates a `WebDDS::Subscriber` within the `WebDDS::DomainParticipant` with the specified *subscriberName*. If the `WebDDS::Subscriber` does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the entities contained within the `WebDDS::Subscriber`. If the check fails, it returns `PERMISSIONS_ERROR`.

If the verification is successful, it deletes the `DDS::Subscriber` associated with the `WebDDS::Subscriber`. If this deletion fails it returns the `DDS_ERROR` error.

It deletes the `WebDDS::Subscriber`. If this fails, it returns `GENERIC_SERVICE_ERROR`, otherwise the operation returns OK.

#### 7.4.4.14 Operation: `get_subscribers`

##### Inputs

- `subscriberNameExpression` (string): An expression on the name of the `Subscriber`.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.
- `subscriberRepresentationList` (string): An XML representation of the list of `Subscriber` objects whose name matches the `subscriberNameExpression`. The format of the representation shall be defined by each PSM.

This operation returns a representation of the list of all the `WebDDS::Subscriber` objects belonging to the `WebDDS::DomainParticipant` whose name matches the ***subscriberNameExpression***. If the operation fails, it returns `GENERIC_SERVICE_ERROR`, otherwise it returns `OK`.

Expression syntax and matching for the ***subscriberNameExpression*** shall use the syntax and rules of the POSIX `fnmatch()` function as specified in POSIX 1003.2-1992, section B.6 [19].

#### 7.4.5 Class `WebDDS::Publisher`

This class is a proxy for a `DDS::Publisher` and serves as the factory for the `WebDDS::DataWriter` objects.

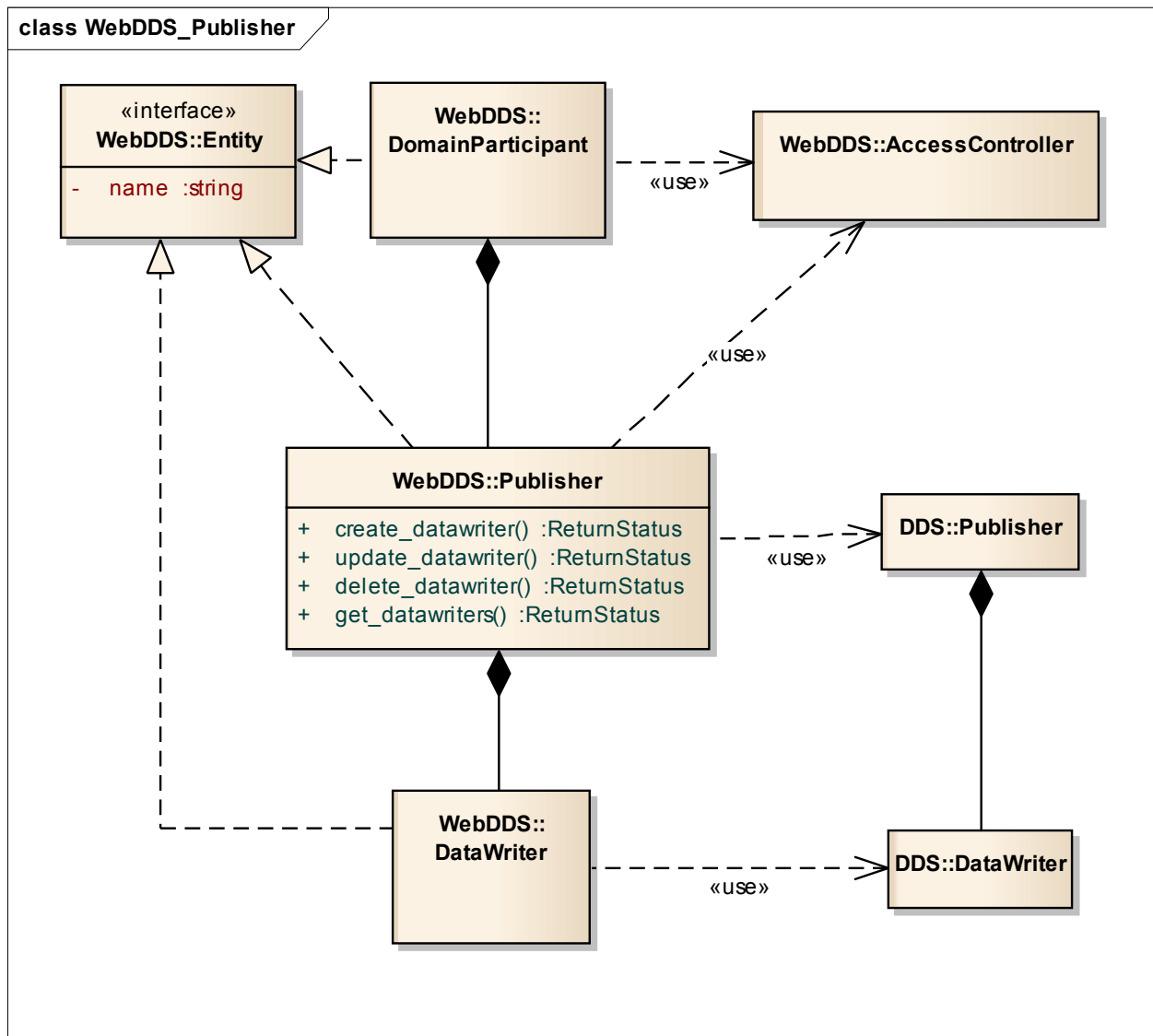


Figure 14—Publisher class with operations

#### 7.4.5.1 Operation: create\_datawriter

##### Inputs

- `datawriterObjectRepresentation` (string) a representation of the `WebDDS::DataWriter` object including its *datawriterName* and optionally Qos. The format of the representation shall be defined for each PSM. The name of the `DataWriter` shall be unique within the scope of the `WebDDS::Publisher` object.

## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation creates a `WebDDS::DataWriter` and the associated `DDS::DataWriter` with the desired QoS policies.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::DataWriter` with the specified *`datawriterName`* within the `WebDDS::Publisher`. If the `WebDDS::DataWriter` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It invokes the `check_permissions` operation to verify that the user is allowed by the access control policies to create the `DDS::DataWriter` entity with the specified QoS. If the check fails, the `WebDDS::DataWriter` is not created and the operation returns the `PERMISSIONS_ERROR` error.

If the permissions check succeeds, the operation creates a `WebDDS::DataWriter` which in turns creates a `DDS::DataWriter` using the specified QoS. The created `DDS::DataWriter` belongs to the `DDS::Publisher` associated with the `WebDDS::Publisher`.

The `DDS::DataWriter` is created disabled. If the creation fails, then all the created objects are destroyed and the operation returns the `DDS_ERROR` error.

If all the creations are successful, the `DDS::DataWriter` is enabled and the operation returns `OK`.

### 7.4.5.2 Operation: `update_datawriter`

#### Inputs

- `datawriterObjectRepresentation` (string) a representation of the `WebDDS::DataWriter` object including its *`datawriterName`* and optionally QoS. The format of the representation shall be defined by each PSM. The name of the `DataWriter` shall correspond to a previously-created `DataWriter` within the `Publisher` object.

#### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation updates the QoS of an existing `DDS::DataWriter`.

This operation performs the following logical steps:

It locates a `WebDDS::DataWriter` within the `WebDDS::Publisher` with the specified *`datawriterName`*. If the `WebDDS::DataWriter` does not exist, it returns an error.



It uses the `check_permissions` to verify that the `WebDDS::User` has the permissions required to change the QoS of the associated `DDS::DataWriter` to the new desired value.

It changes the QoS of the associated `DDS::DataWriter`. If the specified QoS policies are not compatible (in the DDS point of view), the operation will return `DDS_ERROR` and the `DataWriter` will be left with its original QoS.

If all the aforementioned actions and checks are successful, the operation returns OK.

#### 7.4.5.3 Operation: `delete_datawriter`

##### Inputs

- `datawriterName` (string): The name of the `DataWriter`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation deletes an existing `WebDDS::DataWriter` and the associated `DDS::DataWriter`. This operation performs the following logical steps:

It locates a `WebDDS::DataWriter` associated with the `WebDDS::Publisher` with the specified *`datawriterName`*. If the `WebDDS::Publisher` does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the `DDS::DataWriter` associated with the `WebDDS::DataWriter`. If the check fails, it returns `PERMISSIONS_ERROR`.

If the verification is successful, it deletes the `DDS::DataWriter` associated with the `WebDDS::DataWriter`. If this deletion fails, it returns the `DDS_ERROR` error.

It deletes the `WebDDS::DataWriter`. If this fails, it returns `GENERIC_SERVICE_ERROR`, otherwise the operation returns OK.

#### 7.4.5.4 Operation: `get_datawriters`

##### Inputs

- `datawriterNameExpression` (string): An expression on the name of the `DataWriter`.

##### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

- `datawriterRepresentationList` (string): An XML representation of the list of `DataWriter` objects whose name matches the `datawriterNameExpression`. The format of the representation shall be defined by each PSM.

This operation returns a representation of the list of all the `WebDDS::DataWriter` objects belonging to the `WebDDS::Publisher` whose name matches the *datawriterNameExpression*. If the operation fails, it returns `GENERIC_SERVICE_ERROR`, otherwise it returns `OK`.

Expression syntax and matching for the *datawriterNameExpression* shall use the syntax and rules of the POSIX `fnmatch()` function as specified in POSIX 1003.2-1992, section B.6 [19].

### 7.4.6 Class `WebDDS::Subscriber`

This class is a proxy for a `DDS::Subscriber` and serves as the factory for the `WebDDS::DataReader` objects.

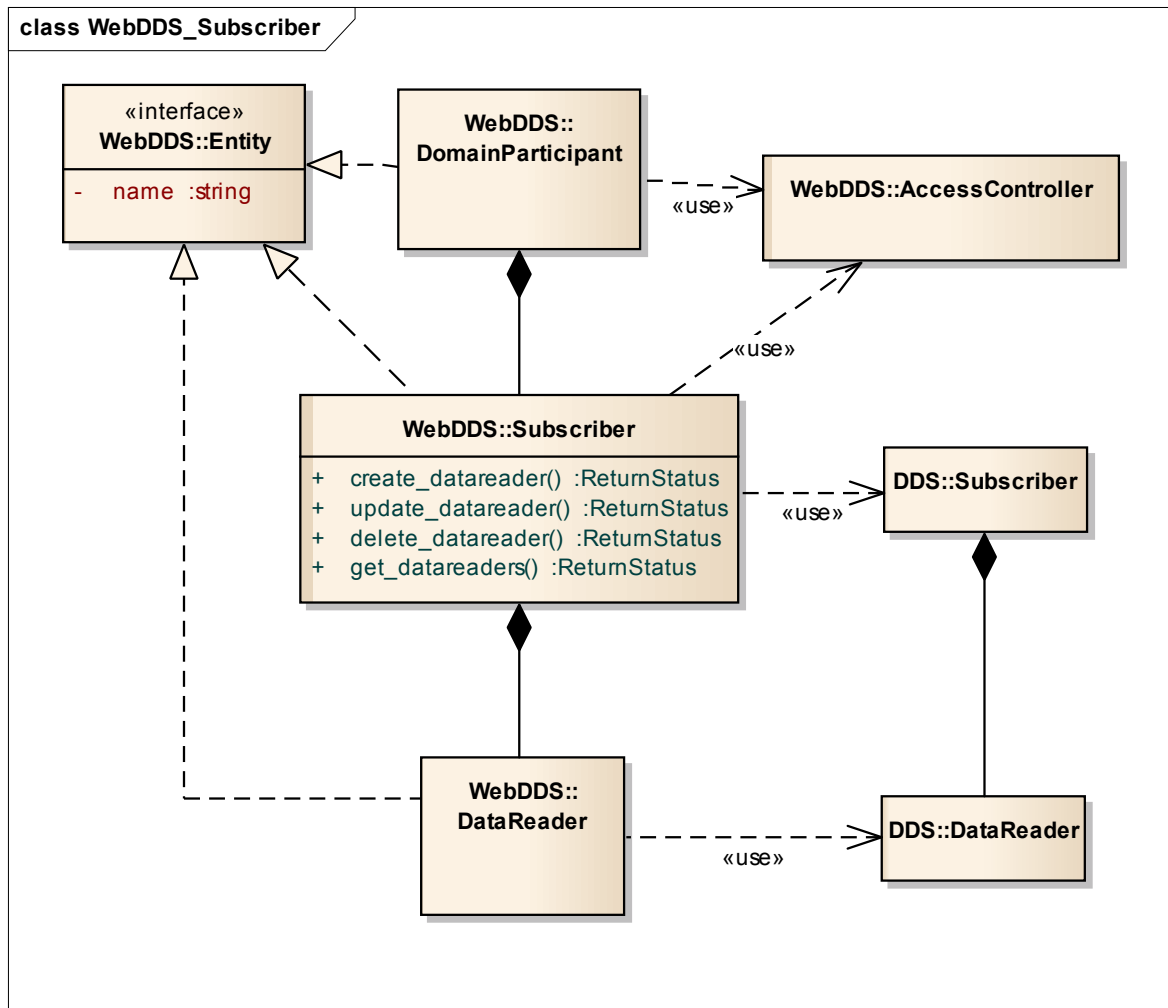


Figure 15—Subscriber class with operations

#### 7.4.6.1 Operation: create\_datareader

##### Inputs

- `datareaderObjectRepresentation` (string) a representation of the `WebDDS::DataReader` object including its ***datareaderName*** and optionally `QoS`, `ContentFilter`, and `Conditions`. The format of the representation shall be defined by each PSM. The name of the `DataReader` shall be unique within the scope of the `WebDDS::Subscriber` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation creates a `WebDDS::DataReader` and the associated `DDS::DataReader` with the desired QoS policies.

This operation performs the following logical steps:

It checks if there is already a pre-existing `WebDDS::DataReader` with the specified ***datareaderName*** within the `WebDDS::Subscriber`. If the `WebDDS::DataReader` already exists, it returns the `OBJECT_ALREADY_EXISTS` error.

It invokes the `check_permissions` operation to verify that the user is allowed by the access control policies to create the `DDS::DataReader` entity with the specified QoS. If the check fails, the `WebDDS::DataReader` is not created and the operation returns the `PERMISSIONS_ERROR` error.

It extracts the name of the `Topic` from the ***datareaderObjectRepresentation*** and checks to determine whether there is an existing `WebDDS::Topic` with that name. If none is found, it returns the `INVALID_INPUT` error.

If the ***datareaderObjectRepresentation*** contains a content filter, then `DDS::DomainParticipant` is used to create a `DDS::ContentFilteredTopic` that uses `DDS::Topic` associated with the `WebDDS::Topic` that was found and the filter expression and parameters found within the ***datareaderObjectRepresentation***. If the creation fails, it returns the `DDS_ERROR` error.

It creates a `WebDDS::DataReader` which in turn creates a `DDS::DataReader` using the `DDS::Topic` (or `DDS::ContentFilteredTopic`) and the specified QoS. The created `DDS::DataReader` belongs to the `DDS::Subscriber` associated with the `WebDDS::Subscriber`.

If the ***datareaderObjectRepresentation*** contains a status condition, then the `DDS::DataReader` `set_status_condition` is called to match the specified condition.

If the ***datareaderObjectRepresentation*** contains a read conditions and/or query conditions, they are created via appropriate calls to the `DDS::DataReader` `create_read_condition`

and/or `create_query_condition`.

The `DDS::DataReader` is created disabled. If the creation fails, all the created objects are destroyed and the operation returns the `DDS_ERROR` error.

If all the creations are successful, the `DDS::DataReader` is enabled and the operation returns OK.

#### 7.4.6.2 Operation: `update_datareader`

##### Inputs

- `datareaderObjectRepresentation` (string) a representation of the `WebDDS::DataReader` object including its *`datareaderName`* and optionally QoS and Conditions. The format of the representation shall be defined by each PSM. The name of the `DataReader` shall correspond to a previously created `DataReader` within the `Subscriber` object.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation updates the QoS of an existing `DDS::DataReader`.

This operation performs the following logical steps:

It locates a `WebDDS::DataReader` within the `WebDDS::Subscriber` with the specified *`datareaderName`*. If the `WebDDS::DataReader` does not exist, it returns an error.

It uses the `check_permissions` to verify that the `WebDDS::User` has the permissions required to change the QoS of the associated `DDS::DataReader` to the new desired value.

It changes the QoS of the associated `DDS::DataReader`. If the specified QoS policies are not compatible (in the DDS point of view), the operation will return `DDS_ERROR` and the `DataReader` will be left with its original QoS.

If all the aforementioned actions and checks are successful, the operation returns OK.

#### 7.4.6.3 Operation: `delete_datareader`

##### Inputs

- `datareaderName` (string): The name of the `DataReader`.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation deletes an existing `WebDDS::DataReader` and the associated `DDS::DataReader`. This operation performs the following logical steps:

It locates a `WebDDS::DataWriter` associated with the `WebDDS::Subscriber` with the specified *datareaderName*. If the `WebDDS::Subscriber` does not exist, it returns the `INVALID_OBJECT` error.

It calls the `check_permissions` operation to verify that the user is allowed by the access control policies to delete the `DDS::DataReader` associated with the `WebDDS::DataReader`. If the check fails, it returns `PERMISSIONS_ERROR`.

If the verification is successful, it deletes the `DDS::DataReader` associated with the `WebDDS::DataReader` as well as any contained objects such read or query conditions. If this deletion fails, it returns the `DDS_ERROR` error.

It deletes the `WebDDS::DataReader`. If this fails, it returns `GENERIC_SERVICE_ERROR`, otherwise the operation returns `OK`.

#### 7.4.6.4 Operation: `get_datareaders`

##### Inputs

- `datareaderNameExpression` (string): An expression on the name of the `DataReader`.

##### Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.
- `datareaderRepresentationList` (string): An XML representation of the list of `DataReader` objects whose name matches the `datareaderNameExpression`. The format of the representation shall be defined by each PSM.

This operation returns a representation of the list of all the `WebDDS::DataReader` objects belonging to the `WebDDS::Subscriber` whose name matches the *datareaderNameExpression*. If the operation fails, it returns `GENERIC_SERVICE_ERROR`, otherwise it returns `OK`.

Expression syntax and matching for the *datareaderNameExpression* shall use the syntax and rules of the POSIX `fnmatch()` function as specified in POSIX 1003.2-1992, section B.6 [19].

#### 7.4.7 Class `WebDDS::DataWriter`

This class is a proxy for a `DDS::DataWriter` and provides the means to write data. The class provides operations to manage the data-instances written. For example register, unregister, and dispose data-instances with the semantics defined by the DDS specification.

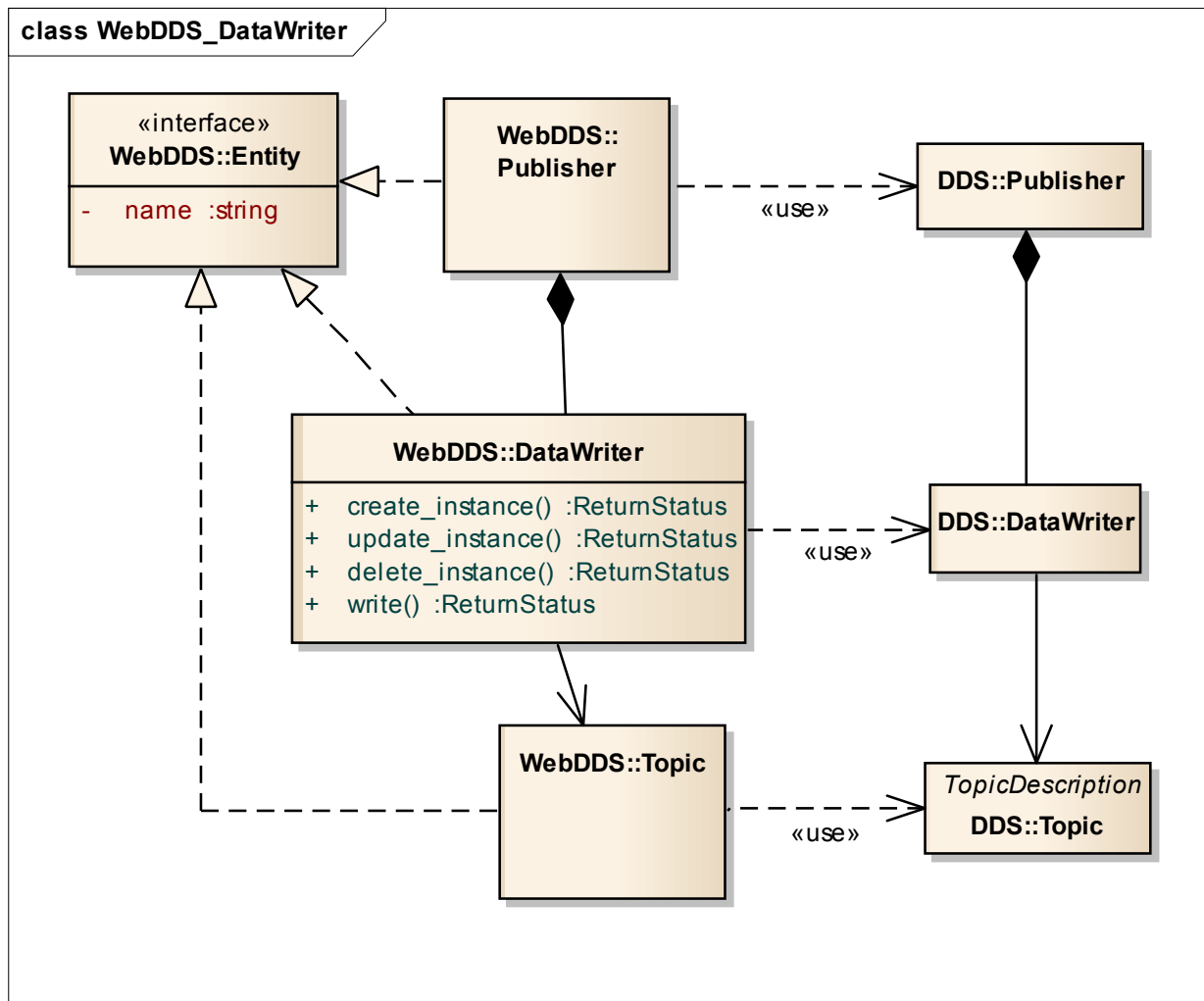


Figure 16—DataWriter class with operations

#### 7.4.7.1 Operation: create\_instance

##### Inputs

- `sampleData` (string): A data-sample represented using the XML format as specified by the DDS-XTYPES. Only the fields of the data that are defined as key in the associated data type are relevant to this operation.

##### Outputs

- `instanceHandleRepresentation` (string): An opaque handle that can be used to refer to the registered instance.
- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It constructs a data-object of the appropriate type for the data-writer from the `dataSample`. Only the fields that are marked as “key” within the data-type are considered for this. If the construction results in an error, it returns the `INVALID_INPUT` error.

It calls the `register_instance` operation on the `DDS::DataWriter` associated with the `WebDDS::DataWriter`. If this operation fails, it returns the `DDS_ERROR` error. Otherwise it returns OK and fills the ***instanceHandleRepresentation*** with a representation of the `DDS InstanceHandle_t` returned by the `register_instance` operation.

#### 7.4.7.2 Operation: update\_instance

##### Inputs

- `writeSampleInfo` (string): An optional XML representation of the `WebDDS::WriteSampleInfo`. The `writeSampleInfo` contains information on the data-sample, the specific representation shall be defined by each PSM and it may contain a timestamp, the `instanceHandle` returned by a previous call to `create_instance` or `update_instance`, and other information as specified by each PSM.
- `sampleData` (string): A representation of the data-sample. The format of the representation shall be defined by each PSM.

##### Outputs

- `instanceHandleRepresentation` (string): An opaque handle that can be used to refer to the registered instance.
- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It constructs a data-object of the appropriate type for the data-writer from the `dataSample`. If the construction results in an error, it returns the `INVALID_INPUT` error.

Depending on whether the `writeSampleInfo` input parameter is present it shall call either the `write` or the `write_w_timestamp` operation on the `DDS::DataWriter` associated with the `WebDDS::DataWriter`. If not present, it calls `write` and if present, it calls `write_w_timestamp` using the timestamp specified within the `writeSampleInfo`.

If the calls to `write` or `write_w_timestamp` return an error, the operation shall return the `DDS_ERROR` error. Otherwise it shall return OK and fill the ***instanceHandleRepresentation*** with a representation of the `DDS InstanceHandle_t` returned by the `DDS` operation.

### 7.4.7.3 Operation: delete\_instance

#### Inputs

`writeSampleInfo` (string): An optional XML representation of the `WebDDS::WriteSampleInfo`. The `writeSampleInfo` contains information on the data-sample, such as a timestamp, the `instanceHandle` returned by a previous call to `create_instance` or `update_instance`, it also contains whether the instance should be unregistered or disposed according to the definitions in the DDS specification [1].

- `sampleData` (string): A representation of the data. Only the fields of the data that are defined as key in the associated data type are relevant to this operation. The format of the representation shall be defined by each PSM. This parameter is optional if the `writeSampleInfo` parameter is present.

#### Outputs

- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation shall perform the following logical steps:

If the `writeSampleInfo` input parameter is present, the operation shall construct the DDS `InstanceHandle_t` from the `writeSampleInfo`. If this construction fails, it shall return the `INVALID_INPUT` error.

If the `writeSampleInfo` input parameter is not present, the operation shall construct a data-object of the appropriate type for the data-writer from the `sampleData`. If the construction results in an error, it shall return the `INVALID_INPUT` error.

Depending on whether the `writeSampleInfo` input parameter is present, the operation shall call either the `dispose` or the `dispose_w_timestamp` operation on the `DDS::DataWriter` associated with the `WebDDS::DataWriter`. If not present, it shall call `dispose` and if present it shall call `dispose_w_timestamp` using the timestamp specified within the `writeSampleInfo`.

If the calls to `dispose` or `dispose_w_timestamp` return an error, the operation shall return the `DDS_ERROR` error. Otherwise it shall return `OK`.

### 7.4.7.4 Operation: write

#### Inputs

- `sampleData` (string): A data-sample. The format of the representation shall be defined by each PSM.



## Outputs

- `returnStatus` (`ReturnStatus`): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation performs the following logical steps:

It constructs a data-object of the appropriate type for the data-writer from the `dataSample`. If the construction results in an error, it returns the `INVALID_INPUT` error.

It calls the `write` operation on the `DDS::DataWriter` associated with the `WebDDS::DataWriter`.

If the call to `write` returns an error, the operation returns the `DDS_ERROR` error. Otherwise it returns `OK`.

### 7.4.8 Class `WebDDS::DataReader`

This class is a proxy for a `DDS::DataReader` and provides the means to read data from DDS. The class provides operations that allow reading all data, as well as reading the data that matches certain criteria with regards to its contents or instance state. In addition the operation gives the user the option to leave the data in the Service's `DDS::DataReader` (i.e., use the `DDS::DataReader` `read` operation so the same data can be accessed again), or else remove it from the service's `DDS::DataReader` cache (i.e., use the `DDS::DataReader` `take` operation).

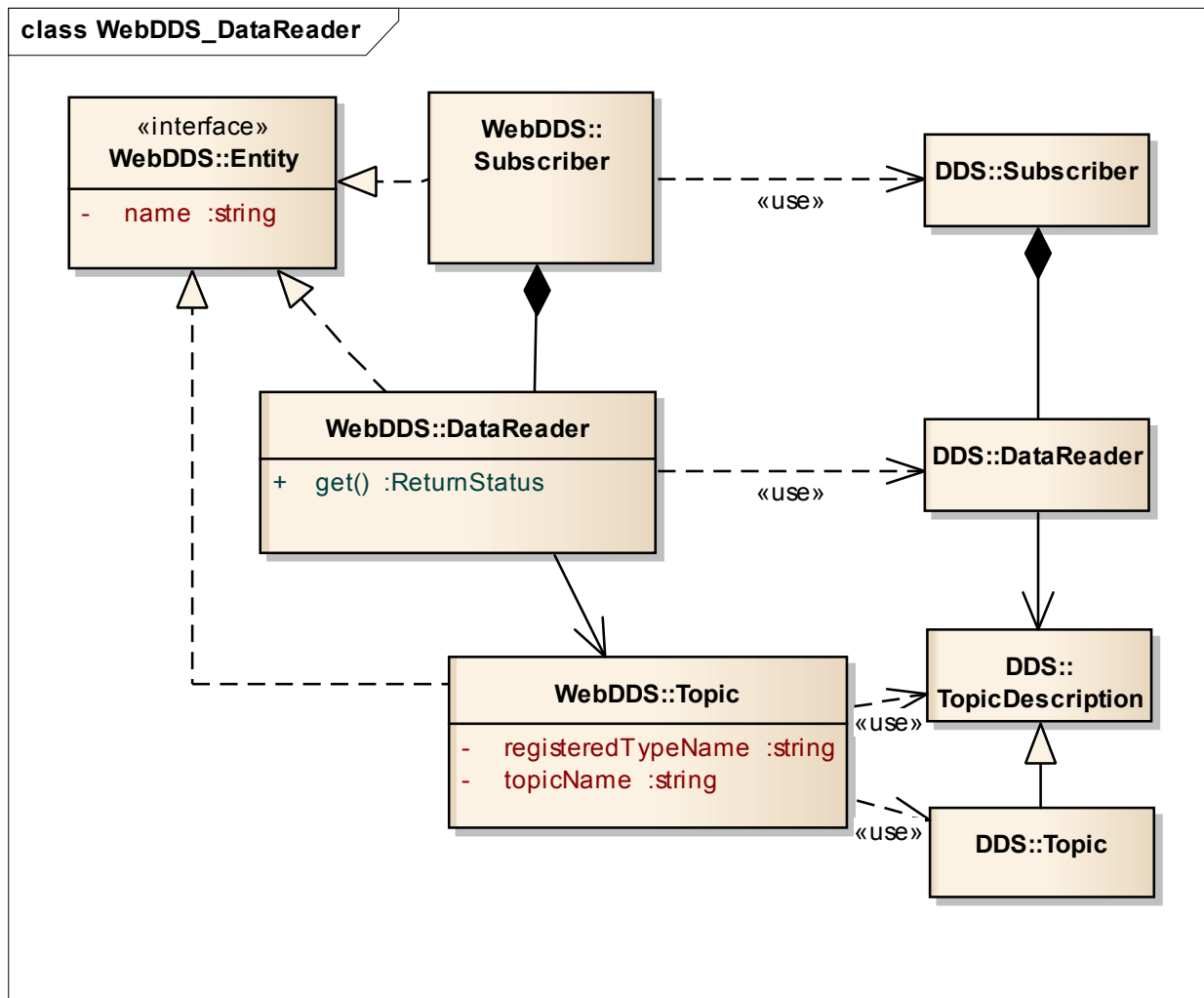


Figure 17—DataReader class with operations

### 7.4.8.1 Operation: get

#### Inputs

- `sampleSelector` (string): An optional filter used to select which samples to access from the `DataReader`, the syntax used for the *sampleSelector* is described in 7.4.8.1.1.
- `removeFromReaderCache` (boolean): Optional parameter indicating whether the samples should be removed from the reader cache (equivalent to the `DDS::DataReader take` operation) or left in the cache (equivalent to the `DDS::DataReader read` operation). If unspecified, it defaults to `TRUE` meaning samples are removed from the reader cache.
- `minSamples` (int32): Optional parameter indicating the minimum number of samples to retrieve. If unspecified, it defaults to one.

- `maxSamples` (int32): Optional parameter indicating the maximum number of samples to retrieve. If unspecified, it defaults to unlimited.
- `maxWait` (float): Optional parameter indicating the willingness of the caller to wait until the specified `minSamples` are available. The operation shall block until either `minSamples` samples are received or the `maxWait` is exceeded. The representation, including the units used for this parameter shall be specified by each PSM. The parameter shall default to zero.

## Outputs

- `sampleSequence` (string): The available data samples along with their respective metadata (corresponding to the DDS `SampleInfo`). The format of the representation shall be defined by each PSM. Each sample in the sequence shall contain the following information:
  - `sampleData` (string): contains a representation of the data accessed from the `DDS::DataReader`.
  - `readSampleInfo` (string): contains are representation of the `DDS::SampleInfo_t` accessed fom the `DDS::DataReader` that is associated with the `sampleData`.
- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

The `get` operation shall allow the client application to retrieve data received by the `DDS::DataReader` associated with the `WebDDS::DataReader`. The operation offers various parameters to control the data retrieved and whether it is left in `DDS::DataReader` cache or removed from it. If the operation requests data to be removed from the `DDS::DataReader` cache, it invokes a “take” operation on the underlying `DDS::DataReader`. If it requests that the data is left, it invokes the “read” operation on the underlying `DDS::DataReader`.

Note that the underlying `DDS::DataReader` offers many operations to allow access the `DDS::DataReader` data in various ways, one at a time, in sequence, selected by the instance, by the value of the various state flags (`sampleState`, `instanceState`, `viewState`), by content (via `DDS::QueryConditions`), etc.

This PIM exposes access to all this functionality using only the “get” operation combined with the parameters to the call. Some (non-normative) examples follow:

- To read a single sample leaving it in the `DataReader` cache you can use ***removeFromReaderCache=false*** and ***maxSamples=1***
- To take all the samples for the data instance with a specified instance handle “MyHandle” include the expression ***instanceHandle=“MyValue”*** within the ***sampleSelector***.

- To take all the samples with *instanceState* “NOT\_ALIVE\_DISPOSED” include the expression *instanceState* =” NOT\_ALIVE\_DISPOSED” within the *sampleSelector*.

The syntax used for the *sampleSelector* is described in 7.4.8.1.1.

This operation performs the following logical steps:

It parses the *sampleSelector* to determine if it is a DDS *FilterExpression* or a *MetadataExpression*. If there is a parse error, it returns the *INVALID\_INPUT* error.

There are three possible cases depending on whether the *sampleSelector* is empty, it contains a *FilterExpression*, or it contains a *MetadataExpression*.

**Case 1:** If the *sampleSelector* is empty, then the operation calls the *read* or the *take* operation on the *DDS::DataReader* associated with the *WebDDS::DataReader*. If the parameter *removeFromReaderCache* is *true*, then it calls *take*. Otherwise it calls *read*. The *minSamples*, *maxSamples*, and *maxWait* parameters control the number of samples that must be obtained from the *DataReader* prior to returning from the function. These parameters do not have one-to-one direct correspondence with parameters to the *DDS::DataReader* *read* and *take* operations. Rather they indicate what the *WebDDS::DataReader* wrapper logic must do. For example, if the call to the underlying *DDS::DataReader* operation does not return the requested *minSamples*, then the *WebDDS::DataReader* shall keep retrying the *read/take* operation on the underlying *DDS::DataReader* and accumulate the results until either the requested *minSamples* have been obtained or the *maxWait* time has been exceeded.

**Case 2:** If the *sampleSelector* is a *FilterExpression*, then the operation uses the *FilterExpression* to construct a *DDS QueryCondition* and uses the operation *read\_w\_condition* or *take\_w\_condition* to access the samples from the *DDS::DataReader*. Aside from this the logic is the same described in Case 1.

**Case 3:** If the *sampleSelector* is a *FilterExpression*, then the operation uses the *MetadataExpression* to construct the *sample\_states*, *view\_states*, and *instance\_state* parameters to use in the calls to the *DDS::DataReader* *read* or *take* operation. Aside from this the logic is the same described in Case 1.

If any of the operations on the *DDS::DataReader* return an error, the operation returns the *DDS\_ERROR* error.

The operation fills the *sampleSequence* output parameter with a representation of the samples obtained by the aforementioned process and *readerSampleInfo* with a representation of the corresponding *DDS::SampleInfo\_t* for those samples and returns *OK*. The timeout is detected by an *OK* return with a *sampleSequence* containing fewer samples than the ones specified by the *minSamples* parameter.

#### 7.4.8.1.1 Sample Selector Syntax

The *sampleSelector* re-uses the same syntax defined for the DDS SQL `FilterExpression` (see Annex A of the DDS specification titled “Annex A: Syntax for DCPS Queries and Filters”) [20], except that the syntax is extended to allow additional selection criteria. The extended syntax is defined using BNF grammar below:

```
SampleSelector ::= FilterExpression
                | MetadataExpression
                | FilterExpression 'AND' MetadataExpression
                | FilterExpression 'OR' MetadataExpression
                .
FilterExpression ::= <<Defined in Annex A of the DDS Spec >>
MetadataExpression ::= MetadataExpression 'OR' MetadataExpression
                    | MetadataExpression 'AND' MetadataExpression
                    | InstanceHandleExpr
                    | InstanceStateExpr
                    | SampleStateExpr
                    | ViewStateExpr
                    .
InstanceHandleExpr ::= instanceHandle '=' STRING
                    .
InstanceStateExpr ::= instanceState '=' InstanceStateValue
                   .
SampleStateExpr ::= sampleState '=' SampleStateValue
                 .
ViewStateExpr ::= viewState '=' ViewStateValue
               .
InstanceStateValue ::= 'ALIVE'
                   | 'NOT_ALIVE_DISPOSED'
                   | 'NOT_ALIVE_NO_WRITERS'
                   .
SampleStateValue ::= 'READ'
                   | 'NOT_READ'
                   .
ViewStateValue ::= 'NEW'
                 | 'NOT_NEW'
                 .
```

#### 7.4.9 Class `WebDDS::WaitSet`

This class is a proxy for a DDS `WaitSet` and provides the means for a client to wait for specific conditions such as the arrival of data on certain Topics.

##### 7.4.9.1 Operation: `wait`

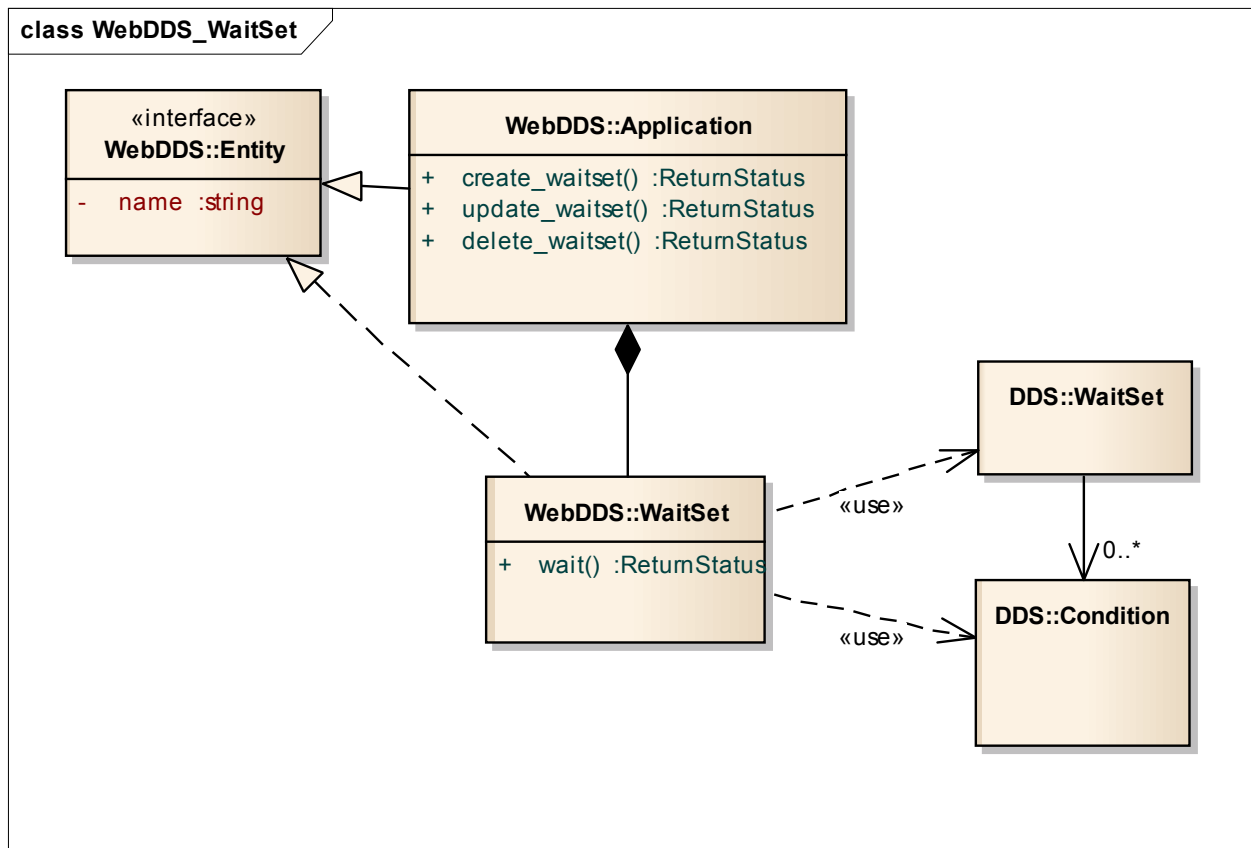


Figure 18—WaitSet class with operations

### Inputs

- `timeout` (float): A timeout in seconds.

### Outputs

- `conditionNameList` (ConditionList): The list of conditions that are active. The format of the representation shall be defined by each PSM.
- `returnStatus` (ReturnStatus): A numeric code indicating success or failure of the operation and a textual description in case of failure.

This operation allows the client application to block waiting for a set of conditions to become active, or else for a timeout to occur. The operation shall return immediately if any of the conditions associated with the `WaitSet` are active at the time the operation is called. If no conditions are active, it shall wait until either a condition becomes active or else a timeout occurs.

## 8 Web-Enabled DDS Platform-Specific Mappings

### 8.1 General

The Web-Enabled DDS specification maps the Object Model to the following two web platforms: REST and SIMPLE-WSDL-SOAP

- The REST platform maps the WebDDS Object Model into REST resources and operations on those resources.
- The SIMPLE-WSDL-SOAP has equivalent functionality and purpose to the SIMPLE-REST platform, except it is mapped to a WSDL/SOAP platform

### 8.2 Formats and Representations for the REST and SIMPLE-WSDL-SOAP platforms

The REST and SIMPLE-WSDL-SOAP platforms share some common XML-based formats and representations for the WebDDS Objects. These are described below.

#### 8.2.1 QoS Representations

The following representations of the WebDDS::Qos and the WebDDS::QosProfile objects are used by one or more of the platforms.

##### 8.2.1.1 XML QoS and QosProfile Representation

Qos and Qos Profiles may be represented in XML as described in the XML QoS Profiles defined by [DDS-CCM] [9][8].

#### 8.2.2 Type Representations

The following representations of the WebDDS::Type objects are used by one or more of the platforms.

##### 8.2.2.1 XML Type Representation

Data Types may be represented in XML as described in the XML Type Representation defined by [DDS-XTYPES].

#### 8.2.3 Data Representations

The following representations of the WebDDS::Data objects are used by one or more of the platforms.

##### 8.2.3.1 XML Data Representation

Data may be represented in XML as described in the XML Data Representation defined by [DDS-

XTYPES].

## 8.2.4 WebDDS Entity Representations

The following representations of the WebDDS Entity objects, that is objects of classes that implement the `WebDDS::Entity` interface are used by one or more of the platforms.

### 8.2.4.1 XML Entity Representation

Objects of the classes defined in the WebDDS Object Model that implement the `WebDDS::Entity` interface may be represented in XML. Unless defined differently for a specific PSM the XML representation of these objects uses the XML Data Representation defined by DDS-XTYPES applied to the objects defined in the following IDL:

```
1.     @Mutable
2.     struct named_object {
3.         string name;
4.     };
5.
6.     @Mutable
7.     struct entity : named_object {
8.         @Optional Qos qos;
9.         @Optional string qos_profile;
10.    };
11.
12.    @Mutable
13.    struct topic : entity {
14.        @Optional string registered_type_name;
15.    };
16.
17.    @Mutable
18.    struct data_writer : entity {
19.        string topic_name;
20.    };
21.    typedef sequence<data_writer> datawriter_seq;
22.
23.    @Mutable
24.    struct publisher : entity {
25.        @Optional datawriter_seq data_writers;
26.    };
27.    typedef sequence<publisher> publisher_seq;
28.
29.    @Mutable
30.    struct condition : named_object {
31.        string expression;
32.    };
33.    typedef sequence<condition> condition_seq;
34.
```



```

35.     @Mutable
36.     struct data_reader : entity {
37.         string topic_name;
38.         @Optional condition status_condition;
39.         @Optional condition_seq read_conditions;
40.         @Optional condition_seq query_conditions;
41.     };
42.     typedef sequence<data_reader> datareader_seq;
43.
44.     @Mutable
45.     struct subscriber : entity {
46.         @Optional datareader_seq data_readers;
47.     };
48.     typedef sequence<subscriber> subscriber_seq;
49.
50.     @Mutable
51.     struct wait_set : named_object {
52.         sequence<string> condition_name;
53.     };
54.
55.     @Mutable
56.     struct participant : entity {
57.         @long domain_id;
58.         @Optional publisher_seq publishers;
59.         @Optional subscriber_seq subscribers;
60.     };
61.     typedef sequence<participant> participant_seq;
62.
63.     @Mutable
64.     struct user_id : named_object {
65.         @Optional string access_token;
66.     };
67.
68.     @Mutable
69.     struct application : named_object {
70.         @Optional user_id user;
71.         @Optional participant_seq participants;
72.     }

```

### 8.3 REST Platform

REST can be seen as a request-reply architecture where a client access and modifies a representation the state of the server using standardized operations as POST, PUT, GET, and DELETE on a set of resources addressed by means of URIs.

This specification can be implemented on a REST/HTTP and REST/HTTPS platform by mapping the objects in the WebDDS PIM into resources and their operation into one of the allowed REST

operations POST, PUT, GET, and DELETE.

### 8.3.1 Mapping of WebDDS PIM to Resources

Each Object in the WebDDS PIM is mapped into a resource with the URI shown in the table below. All URI have the prefix “/dds/rest1”. For brevity the prefix is omitted from the URIs in the table below.

**Table 4 Resource URIs for the REST platform**

Object Type	Resource URI All resources have the prefix “/dds/rest1”
Application	/applications/<appname>
QosProfile	/applications/<appname>/qos_profiles/<profile_name>
Type	/applications/<appname>/types/<typename>
WaitSet	/applications/<appname>/waitsets/<waitsetname>
Participant	/applications/<appname>/participants/<partname>
RegisteredType	/applications/<appname>/participants/<partname>/registered_types/<reg_type_name>
Topic	/applications/<appname>/participants/<partname>/topics/<topicname>
Publisher	/applications/<appname>/participants/<partname>/publishers/<pubname>
Subscriber	/applications/<appname>/participants/<partname>/subscribers/<subname>
DataWriter	/applications/<appname>/participants/<partname>/publishers/<pubname>/data_writers/<dwname>
DataReader	/applications/<appname>/participants/<partname>/subscribers/<subname>/data_readers/<drname>

### 8.3.2 Mapping rules from WebDDS PIM operations to REST methods

The operations on the WebDDS objects are mapped according to the following rules:

- Create operations are mapped into the “POST” method.
- Delete operations are mapped into the “DELETE” HTTP method unless they take parameters in which case they map to a POST.
- Update operations are mapped into the “PUT” HTTP method.
- Get operations are mapped to the “GET” HTTP method.

- Operations that do not fit into the above categories are mapped into the POST method.

The parameters to the operations in the WebDDS PIM object are mapped according to the following rules:

- Create (POST) operations receive the parameters in the HTML body. The parameter is the XML representation of the object being created as defined in 8.2.4.
- Delete operations mapped to DELETE operate just on the URI. DELETE receives no parameters in the body.
- Delete operations mapped to POST receive the parameters in the HTML body. It receives the XML representation of the object being deleted minimally containing the name or any fields that form a unique identifier.
- Update (PUT) operations receive the parameters in the HTML body. The parameter is an XML representation of the object. It is the same format as when the object was created
- Get (GET) operations receive the parameters as part of the URI. The parameters follow the resource name, separated by a “?” character. Each parameter is represented using the format <parameter-Name>=<parameterValue>. Successive parameters are separated by the “&” sign. Non-allowed characters are encoded using percent-encoding as is customary in URIs.

The WebDDS::ReturnStatus returned by all PIM operations is mapped to the HTTP response Status line (IETF RFC 2616 [13], Section 6.1) and it shall not appear in the body of the HTTP response.

The remaining outputs from the PIM operations are returned in the HTTP response body as specified in 8.3.3.

- The string returnMessage attribute of the WebDDS::ReturnStatus shall be mapped to the Reason-Phrase in the HTTP Status Line.
- The integer returnCode attribute of the WebDDS::ReturnStatus shall be mapped to the HTTP status code in accordance with the following rules:
  - ReturnCode OK shall be mapped differently depending of the PIM operation:
    - The PIM login operations shall map it to HTTP status **201 Created**
    - The PIM logout operations shall map it to HTTP status **204 No Content**
    - The PIM create operations shall map it to HTTP **204 No Content**
    - The PIM delete operations shall map it to HTTP status **204 No Content**
    - The PIM get operations shall map it to HTTP status **200 OK**
    - The PIM update operations shall map it to HTTP status **204 No Content**
  - ReturnCode OBJECT\_ALREADY\_EXISTS is mapped to HTTP status **409 Conflict**
  - ReturnCode INVALID\_INPUT shall be mapped to HTTP status **422 Unprocessable Entity** (see IETF RFC 4918 [22]).
  - ReturnCode INVALID\_OBJECT shall be mapped to HTTP status **404 Not Found**
  - ReturnCode ACCESS\_DENIED shall be mapped to HTTP status **401 Unauthorized**
  - ReturnCode INVALID\_SESSION\_ID shall be mapped to HTTP status **409 Not Found**
  - ReturnCode PERMISSIONS\_ERROR shall be mapped to HTTP status **403 Forbidden**

- ReturnCode GENERIC\_SERVICE\_ERROR shall be mapped to HTTP status **500 Internal Server Error**

### 8.3.3 Complete mapping of WebDDS PIM operations to REST methods

The complete mapping is shown in the table below.

In addition to the HTTP methods specified in the table, the HEAD HTTP method shall be supported on the same URIs as the GET method. The HEAD method shall behave identically to the GET method except for it shall return no body.

**Table 5 Mapping of PIM operations to REST methods**

Operation	HTTP method	URI	HTTP request and response bodies
WSDDS::login	POST	/applications/	Request body: applicationRepresentation Response body: authenticatedSessionRepresentation
WSDDS::logout	DELETE	/applications/<appname>	Request body: Empty Response body: Empty
Application ::create_participant	POST	/applications/<appname>/participants	Request body: participantObjectRepresentation Response body: Empty
Application ::update_participant	PUT	/applications/<appname>/participants/<partname>	Request body: participantObjectRepresentation Response body: Empty
Application ::delete_participant	DELETE	/applications/<appname>/participants/<partname>	Request body: Empty Response body: Empty
Application ::get_participants	GET	/applications/<appname>/participants	Request body: Empty Response body: List of :participantObjectRepresentation
Application ::create_type	POST	/applications/<appname>/types	Request body: typeObjectRepresentation

			Response body: Empty
Application ::delete_type	DELETE	/applications/<appname>/types/<typename>	Request body: Empty Response body: Empty
Application ::get_types	GET	/applications/<appname>/types	Response body: List of: typeObjectRepresentation
Application ::create_waitset	POST	/applications/<appname>/waitsets	Request body: waitsetObjectRepresentation Response body: Empty
Application ::update_waitset	PUT	/applications/<appname>/waitsets /<waitsetname>	Request body: waitsetObjectRepresentation Response body: Empty
Application ::delete_waitset	DELETE	/applications/<appname>/waitsets /<waitsetname>	Request body: Empty Response body: Empty
Application ::get_waitsets	GET	/applications/<appname>/waitsets	Response body: List of: waitsetObjectRepresentation
Participant ::register_type	POST	/applications/<appname>/participants/<partname>/registered_types/	Request body: registerTypeObjectRepresentation Response body: Empty
Participant ::unregister_type	DELETE	/applications/<appname>/participants/<partname>/registered_types/<registered_type_name>	Request body: Empty Response body: Empty
Participant ::get_registered_types	GET	/applications/<appname>/participants/<partname>/registered_types	Response body: List of: registerTypeObjectRepresentation
Participant ::create_topic	POST	/applications/<appname>/participants/<partname>/topics/	Request body: topicObjectRepresentation Response body: Empty

Participant ::update_topic	PUT	/applications/<appname>/participants/<parname>/topics/<topicname>	Request body: topicObjectRepresentation Response body: Empty
Participant ::delete_topic	DELETE	/applications/<appname>/participants/<parname>/topics/<topicname>	Request body: Empty Response body: Empty
Participant ::get_topics	GET	/applications/<appname>/participants/<parname>/topics	Response body: List of: topicObjectRepresentation
Participant ::create_publisher	POST	/applications/<appname>/participants/<parname>/publishers	Request body: publisherObjectRepresentation Response body: Empty
Participant ::update_publisher	PUT	/applications/<appname>/participants/<parname>/publishers/<publishername>	Request body: publisherObjectRepresentation Response body: Empty
Participant ::delete_publisher	DELETE	/applications/<appname>/participants/<parname>/publishers/<publishername>	Request body: Empty Response body: Empty
Participant ::get_publishers	GET	/applications/<appname>/participants/<parname>/publishers	Response body: List of: publisherObjectRepresentation
Participant ::create_subscriber	POST	/applications/<appname>/participants/<parname>/subscribers	Request body: subscriberObjectRepresentation Response body: Empty
Participant ::update_subscriber	PUT	/applications/<appname>/participants/<parname>/subscribers/<subscribername>	Request body: subscriberObjectRepresentation Response body: Empty
Participant ::delete_subscriber	DELETE	/applications/<appname>/participants/<parname>/subscribers/<subscribername>	Request body: Empty Response body: Empty
Participant	GET	/applications/<appname>	Response body:

::get_subscribers		me>/participants/<partname>/subscribers	List of: subscriberObjectRepresentation
Publisher ::create_datawriter	POST	/applications/<appname>/participants/<partname>/publishers/<publishername>/datawriters	Request body: datawriterObjectRepresentation Response body (for 201 response): entityCompactRepresentation Response body: Empty
Publisher ::update_datawriter	PUT	/applications/<appname>/participants/<partname>/publishers/<publishername>/datawriters/<datawritername>	Request body: datawriterObjectRepresentation Response body: Empty
Publisher ::delete_datawriter	DELETE	/applications/<appname>/participants/<partname>/publishers/<publishername>/datawriters/<datawritername>	Request body: Empty Response body: Empty
Publisher ::get_datawriters	GET	/applications/<appname>/participants/<partname>/publishers/<publishername>/datawriters	Response body: List of: datawriterObjectRepresentation
Subscriber ::create_datareader	POST	/applications/<appname>/participants/<partname>/Subscribers/<Subscribername>/datareaders	Request body: datareaderObjectRepresentation Response body: Empty
Subscriber ::update_datareader	PUT	/applications/<appname>/participants/<partname>/Subscribers/<Subscribername>/datareaders/<datareadername>	Request body: datareaderObjectRepresentation Response body: Empty
Subscriber ::delete_datareader	DELETE	/applications/<appname>/participants/<partname>/subscribers/<Subscribername>/da	Request body: Empty Response body: Empty

		tareaders/<datareader name>	
Subscriber ::get_datareaders	GET	/applications/<appname>/participants/<partname>/subscribers/<subscribername>/datareaders	Response body: List of: datareaderObjectRepresentation
DataWriter ::write	POST	/applications/<appname>/participants/<partname>/publishers/<publishername>/datawriters/<datawritername>	Request body: dataObjectRepresentation Response body: Empty
DataReader ::read	GET	/applications/<appname>/participants/<partname>/subscribers/<subscribername>/datareaders/<datareader name>	Response body: List of: dataSampleRepresentation
Waitset::get	GET	/applications/<appname>/waitsets /<waitsetname>	Response body: List of: conditionNames

### 8.3.4 Object representations used by the REST platform

The representation for all parameters and return values used in the REST platform is describe in the table below.

**Table 6 Object and parameter representations used by the REST platform**

<i>Object Representation</i>	<i>Format for the object representation</i>
participantObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: DomainParticipant
typeRepresentation	File: <i>webdds_rest1.xsd</i> Type: typeLibrary File (from DDS-XTYPES): <a href="http://www.omg.org/spec/DDS-XTypes/20120202/dds-xtypes_type_definition.xsd">http://www.omg.org/spec/DDS-XTypes/20120202/dds-xtypes_type_definition.xsd</a>



	Type: moduleElements
waisetRepresentation	File: <i>webdds_rest1.xsd</i> Type: WaitSet
topicObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: Topic
topicRepresentationList	File: <i>webdds_rest1.xsd</i> Type: Topic
topicRepresentationList	File: <i>webdds_rest1.xsd</i> Type: TopicList
publisherObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: Publisher
publisherRepresentationList	File: <i>webdds_rest1.xsd</i> Type: PublisherList
subscriberObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: Subscriber
subscriberRepresentationList	File: <i>webdds_rest1.xsd</i> Type: SubscriberList
datawriterObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: DataWriter
datawriterObjectRepresentationList	File: <i>webdds_rest1.xsd</i> Type: DataWriterList
datareaderObjectRepresentation	File: <i>webdds_rest1.xsd</i> Type: DataReader
datareaderObjectRepresentationList	File: <i>webdds_rest1.xsd</i> Type: DataReaderList
sampleData	Use the XML Data Representation defined by the DDS-XTYPES specification, clause 7.4.2 XML Data Representation
writeSampleInfo	File: <i>webdds_rest1.xsd</i>

	Type: writeSampleInfo
sampleSequence	File: <i>webdds_rest1.xsd</i> Type: SampleSequence

### 8.3.5 HTTP Headers used by the REST platform

This sub clause specifies the request headers and reply headers whose presence and behavior is relied upon by the REST PSM. The use of other standard HTTP headers is not precluded by this specification, however a compliant implementation is not required to include or interpret those headers.

#### 8.3.5.1 HTTP Request Headers

The table below lists the HTTP requests headers used by the WebDDS REST platform.

**Table 7 HTTP request headers used by the REST platform**

Header	Required /Optional	Description
Accept	Required	Request a particular content type. Valid values: application/webdds+xml
Authorization	Required	The authorization token appropriate for the operation being performed:  For the “login” operation it shall include the userName and accessToken (7.3.1.1)  For the remaining operations it shall include the authenticatedSessionToken returned by the login operation
Content- Length	Required (except for the GET and HEAD operations)	Transfer-length of the message-body
Content-Type	Optional	Valid values: application/webdds+xml
Cache-Control	Required	Valid values: as specified in Section 14.9 of IETF 2616.
OMG-DDS-sessionId	Required	

#### 8.3.5.2 HTTP Response Headers

The table below lists the HTTP response headers used by the WebDDS REST platform.

**Table 8 HTTP response headers used by the REST platform**

Header	Required /Optional	Description
Authentication-Info	Required (for response to login)	Uses to communicate the AuthenticatedSessionToken (7.3.1.1)
Cache-Control	Required	Valid values: as specified in Section 14.9 of IETF 2616.
Content- Length	Required	Transfer-length of the message-body
Content-Type	Required	Valid values: application/webdds+xml
Date	Optional	Valid values: as specified in Section 14.18 of IETF 2616.
Expires	Optional	Valid values: as specified in Section 14.21 of IETF 2616.
Location	Required for successful response to POST operations	URI for the newly created resource
Last-Modified	Required for successful responses to GET and HEAD	The last modification time of the resource that is being accessed

## 8.4 Simplified SOAP Platform

The Simplified SOAP platform uses a simpler Object Model. The advantage of the approach is the simplicity.

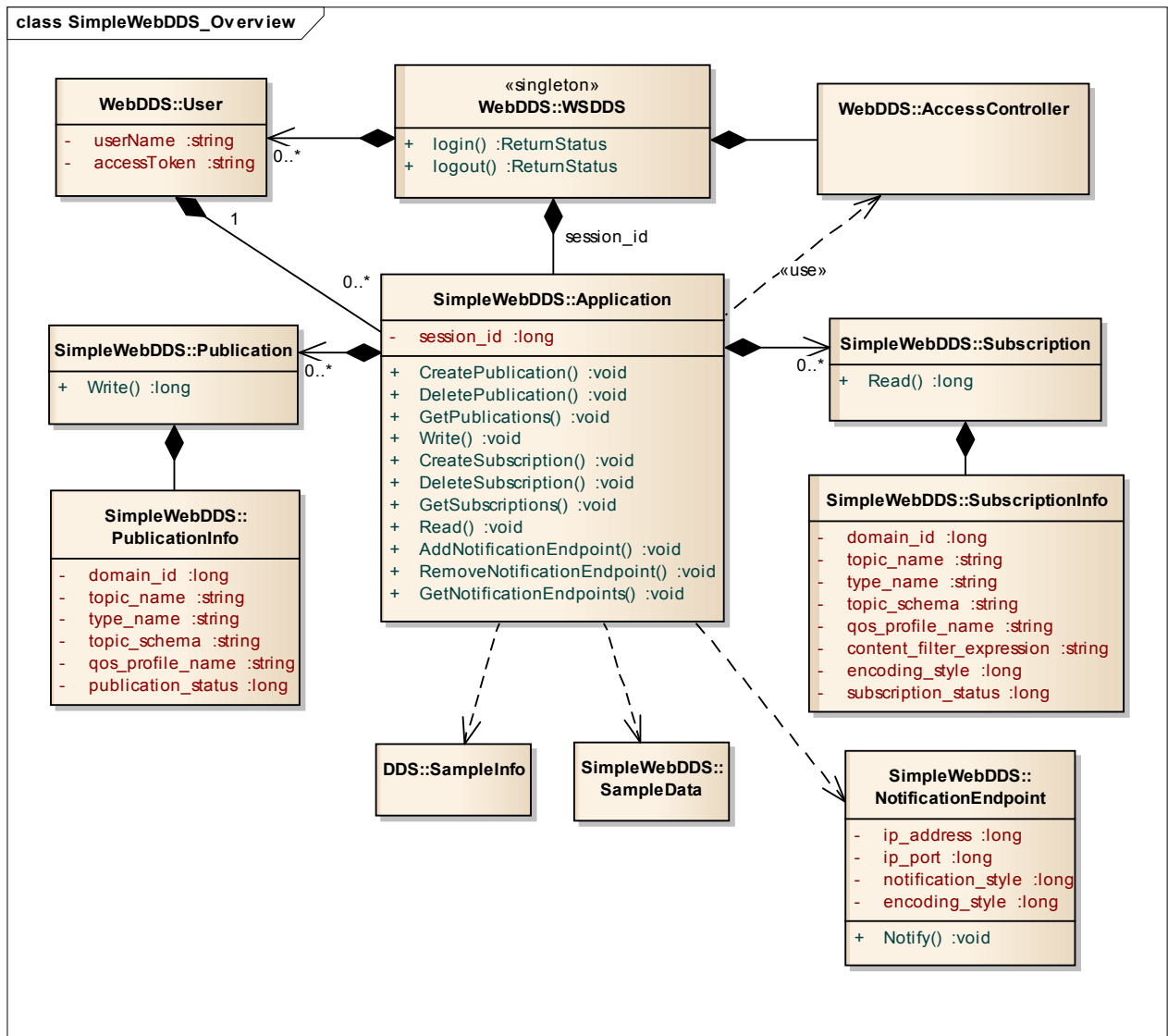


Figure 19—Simplified WebDDS Object Model used by the Simple WSDL-SOAP Platform

The Object Model used by the Simple WSDL-SOAP platform shares the WSDDS, User and AccessController classes with the WebDDS object Model. The remaining classes are specific to this model.

Most of the logic in the simplified object model is carried out by the SimpleWebDDS::Application class which combines the functionality offered by the Application, Participant, Topic, Publisher, Subscriber, DataWriter, and DataReader from the WebDDS object model.

The Simple WSDL-SOAP platform is defined by mapping to SOAP messages for each of the operations specified for the WSDDS and the Application classes of the Simplified Object Model.

This platform relies on WSDL and SOAP and given these are formal languages to define web

interfaces and protocols it can be fully described using just those interfaces.

There are three files that describe this platform: **webdds\_soap1\_types.xsd**, **webdds\_soap1.wsdl**, and **webdds\_soap1\_notify.wsdl**.

The **webdds\_soap1\_types.xsd** defines general data-types that are used in the other WSDL files.

All the client operations (messages) are defined in the WSDL file **webdds\_soap1.wsdl**

The **webdds\_soap1\_notify.wsdl** defines callback operations specified for the `NotificationEndpoint` interface. These operations should be implemented in order to receive notifications of data reception.

The mappings between PIM operations and the SOAP messages is defined in the table below.

**Table 9 Mapping between PIM operations and SOAP messages**

<b>Operation</b>	<b>WSDL Port Type (operation)</b>	<b>Related WSDL/SOAP messages and types</b>	<b>Notes</b>
WSDDS::login	Login	webdds :loginRequest webdds loginResponse	
WSDDS::logout	Logout	webdds:logoutRequest webdds:logoutReply	
Application ::create_participant	N/A	webdds: createPublicationRequest  webdds: createSubscriptionRequest	The PSM specifies the domain participant attributes as part of the parameters that passed to the PIM CreatePublication and CreateSubscription operations.  The PIM actions specified by the operation, specifically the creation of the associated WebDDS::DomainParticipant, shall be executed when the client calls the PSM operations CreatePublication or CreateSubscription and specifies a value for the domainId parameters that has not been specified before.
Application ::update_participant	N/A		The PSM does not support updating participants.
Application	N/A		The PSM does not map this

::delete_participant			<p>operation explicitly into any messages.</p> <p>The PIM actions specified by the operation, including the deletion of the associated WebDDS::DomainParticipant, shall be executed when the last entity belonging to that participant is deleted.</p>
Application ::get_participants	N/A		The PSM does not support this operation.
Application ::create_type	N/A	<p>webdds: createPublicationRequest</p> <p>webdds: createSubscriptionRequest</p>	<p>The PSM specifies the type as part of the parameters passed to the PIM CreatePublication and CreateSubscription operations.</p> <p>The PIM actions specified by the operation, specifically the creation of the WebDDS::Type, shall be executed when the client calls the PSM operations CreatePublication or CreateSubscription and specifies a value for “typeSchema” that has not been used before for that domainId.</p>
Application ::delete_type	N/A		<p>The PSM does not map this operation explicitly into any messages.</p> <p>The PIM actions specified by the operation shall be executed when the last created entity that uses the associated WebDDS::Type is deleted.</p>
Application ::get_types	N/A		The PSM does not support this operation.
Application ::create_waitset	N/A		The PSM does not support this operation.
Application ::update_waitset	N/A		The PSM does not support this operation.

Application ::delete_waitset	N/A		The PSM does not support this operation.
Application ::get_waitsets	N/A		The PSM does not support this operation.
Participant ::register_type	N/A	webdds: createPublicationRequest  webdds: createSubscriptionRequest	The PSM registers types the types as a side-effect of the creation of Data Writes and Data Readers that use the data-type.  The PSM shall implement the PIM actions associated with this operation when the client calls the PSM operations CreatePublication or CreateSubscription and specifies a value for “typeName” that has not been used before for that domainId.
Participant ::unregister_type	N/A		The PSM does not map this operation explicitly into any messages.  The PIM actions specified by the operations shall be executed when the last entity that uses the type is deleted.
Participant ::get_registered_types	N/A		The PSM does not support this operation.
Participant ::create_topic	N/A	webdds: createPublicationRequest  webdds: createSubscriptionRequest	The PSM specifies the Topic as part of the parameters passed to the PIM CreatePublication and CreateSubscription operations.  The PSM shall implement the PIM actions associated with this operation, specifically the creation of the WebDDS::Topic, when the client calls the PSM operations CreatePublication or CreateSubscription and specifies a value for “topicName” that has not been

			used before for that domainId.
Participant ::update_topic	N/A		The PSM does not support this operation.
Participant ::delete_topic	N/A		The PSM does not map this operation explicitly into any messages.  The PIM actions specified by the operations shall be executed when the last created entity that uses the associated WebDDS::Topic is deleted.
Participant ::get_topics	N/A		The PSM does not support this operation.
Participant ::create_publisher	N/A	webdds: createPublicationRequest	The PSM creates the WebDDS::Publisher as needed to support the PSM CreatePublication operation.  The details on when the PIM actions associated with this operation are executed and the WebDDS::Publisher created are left to the implementation as they do not affect interoperability.
Participant ::update_publisher	N/A		The PSM does not support this operation.
Participant ::delete_publisher	N/A		The PSM does not map this operation explicitly into any messages.  The PIM actions specified by the operation shall be executed when the last entity that uses the associated WebDDS::Publisher is deleted.
Participant ::get_publishers	N/A		The PSM does not support this operation.
Participant ::create_subscriber	N/A	webdds: createSubscriptionRequest	The PSM creates the WebDDS::Subscriber as needed to support the PSM CreateSubscription operation.



			The details on when the PIM actions associated with this operation are executed and the WebDDS::Subscriber created are left to the implementation as they do not affect interoperability.
Participant ::update_subscriber	N/A		The PSM does not support this operation.
Participant ::delete_subscriber	N/A		The PSM does not map this operation explicitly into any messages.  The PIM actions specified by the operation shall be executed when the last entity that uses the associated WebDDS::Subscriber is deleted.
Participant ::get_subscribers	N/A		The PSM does not support this operation.
Publisher ::create_datawriter	CreatePublic ation	webdds:createPublicati onRequest  webdds:createPublicati onResponse	
Publisher ::update_datawriter	CreatePublic ation	webdds:createPublicati onRequest  webdds:createPublicati onResponse	The operation CreatePublication can be used to modify an already existing WebDDS::DataWriter if it already exists or create it if it does not exist.
Publisher ::delete_datawriter	RemovePubl ication	webdds:removePublicat ionRequest  webdds:removePublicat ionResponse	
Publisher ::get_datawriters	GetPublicati ons	webdds:getPublications Request  webdds:getPublications Response	
Subscriber ::create_datareader	CreateSubsc ription	webdds:createSubscript ionRequest	

		webdds:createSubscriptionResponse	
Subscriber::update_datareader	CreateSubscription	webdds:createSubscriptionRequest webdds:createSubscriptionResponse	The operation CreateSubscription can be used to modify an already existing WebDDS::DataReader if it already exists or create it if it does not exist.
Subscriber::delete_datareader	RemoveSubscription	webdds:removeSubscriptionRequest webdds:removeSubscriptionResponse	
Subscriber::get_datareaders	GetPublications	webdds:getSubscriptionsRequest webdds:getSubscriptionsResponse	
DataWriter::write	Write	webdds:writeRequest webdds:writeResponse	
DataReader::read	Read	webdds:readRequest webdds:readReply	
Waitset::get	N/A		The PSM does not support this operation.
N/A	AddNotificationEndpoint	webdds:addNotificationEndpointRequest webdds:addNotificationEndpointResponse	This PSM operation has no equivalent in the PIM.  The PSM shall retain the notificationEndpointInfo specified as parameter to the operation and use it to invoke the Notify operation as defined in webdds_soap1_notify.wsdl
N/A	RemoveNotificationEndpoint	webdds:removeNotificationEndpointRequest webdds:removeNotificationEndpointResponse	This PSM operation has no equivalent in the PIM.  This operation shall remove the state added as a result to a previous call to AddNotificationEndpoint and stop future notifications to that

			endpoint.
N/A	GetNotificationEndpoints	webdds:getNotificationEndpointsRequest  webdds:getNotificationEndpointsResponse	This PSM operation has no equivalent in the PIM.  This operation shall return a webdds:notificationEndpointInfoSeq containing the notification endpoints that have been added (and not deleted) by prior calls to AddNotificationEndpoint

The operations in the Simplified SOAP PSM all include a returnCode and a returnMessage as part of the operation return. The PSM's returnString maps directly to the returnMessage attribute of the WebDDS::ReturnStatus in the PIM. The PSM's returnCode maps to the returnCode attribute of the WebDDS::ReturnStatus in the PIM according to the table below.

**Table 10 Mapping of Simplified SOAP PSM ReturnCode to PIM ReturnCode**

<i>Simplified SOAP PSM ReturnCode</i>	<i>PIM ReturnCode</i>
OK	OK
DDS_ERROR	DDS_ERROR
SUBSCRIPTION_ALREADY_EXISTS, PUBLICATION_ALREADY_EXISTS	OBJECT_ALREADY_EXISTS
INVALID_INPUT, INVALID_IP_ADDRESS, INVALID_NOTIFICATION_ENDPOINT_PORT_NUMBER, INVALID_NOTIFICATION_ENDPOINT, BAD_TYPE_SCHEMA, BAD_CONTENT_FILTER_EXPRESSION, BAD_DATA_SAMPLE, QOS_PROFILE_NOT_FOUND, INCOMPATIBLE_TOPIC, TOPIC_CREATED_WITH_DIFFERENT_TYPE_SCHEMA, TOPIC_DEFINED_WITH_DIFFERENT_TYPE_SCHEMA, TOPIC_DISCOVERED_WITH_DIFFERENT_TYPE_SCHEMA, AMBIGUOUS_TYPE_DEFINITION_FOUND_ON_DOMAIN, SUBSCRIPTION_ON_INCOMPATIBLE_STATUS, PUBLICATION_ON_INCOMPATIBLE_STATUS, TOPIC_CREATED_WITH_DIFFERENT_TYPE_NAME	INVALID_INPUT
INVALID_SESSION_ID, INVALID_SUBSCRIPTION_ID, INVALID_PUBLICATION_ID, INVALID_NOTIFICATION_ENDPOINT_ID,  EXPIRED_SESSION	INVALID_OBJECT

ACCESS_DENIED	ACCESS_DENIED
INVALID_SESSION_ID	INVALID_SESSION_ID
PERMISSIONS_ERROR, NO_RIGHTS_JOINING_DOMAIN, NO_RIGHTS_CREATING_TOPIC, NO_RIGHTS_CREATING_UNKNOWN_TOPIC, NO_RIGHTS_SUBSCRIBING, NO_RIGHTS_PUBLISHING, NO_RIGHTS_BEING_NOTIFIED	PERMISSIONS_ERROR
SERVER_ERROR, MAX_SESSION_COUNT_REACHED	GENERIC_SERVICE_ERROR

The full WSDL interfaces are defined in *the normative readable files webdds\_soap1\_types.xsd, webdds\_soap1.xml, and webdds\_soap1\_notify.xml.*

## 8.5 Transport-Level Protocol Considerations

The PSMs in this specification all operate over an HTTP (or HTTPS) transport. The `AuthenticationToken` information should be carried in the HTTP headers using the `auth-scheme` and `auth-parameter` attributes compliant with the IETF RFC 2617 (HTTP Authentication: Basic and Digest Access Authentication) [18].

The HTTP `Content-Type` field should be set to `application/webdds+xml`.

Implementations of this specification shall support the PSMs operating over HTTP and HTTPS. When operating over HTTPS the certificate and certificate authority configuration is left outside this specification.

Implementations of the specification shall support operation over HTTP 1.1. As an option implementers may support upgrading the connection to using WebSockets in compliance with IETF 6455 (The WebSocket Protocol) [14].

## Annex A - References

- [1] DDS: Data-Distribution Service for Real-Time Systems version 1.2. <http://www.omg.org/spec/DDS/1.2>
- [2] Roy Fielding. “Representational State Transfer (REST)” [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- [3] Web Services Description Language (WSDL). <http://www.w3.org/TR/wsdl20/>
- [4] Extensible Messaging and Presence Protocol (XMPP): Core. IETF RFC 6120. XMPP <http://xmpp.org/rfcs/rfc6120.html>
- [5] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) <http://www.w3.org/TR/soap12-part1/>
- [6] DDS-RTPS: Data-Distribution Service Interoperability Wire Protocol version 2.1, <http://www.omg.org/spec/DDS-RTPS/2.1/>
- [7] Uniform Resource Identifier (URI): Generic Syntax. IETF RFC 3986. <http://tools.ietf.org/html/rfc3986>
- [8] DDS for light-weight CCM specification (DDS4CCM) version 1.0. <http://www.omg.org/spec/dds4ccm/1.1/>
- [9] DDS XML Schema for QoS Profile. [http://www.omg.org/spec/dds4ccm/20110201/DDS\\_QoSProfile.xsd](http://www.omg.org/spec/dds4ccm/20110201/DDS_QoSProfile.xsd)
- [10] DDS Extensible Types Specification (DDS-XTYPES) version 1.0. <http://www.omg.org/spec/DDS-XTYPES/1.0/>
- [11] DDS-XTypes XML Type Representation, XSD format. [http://www.omg.org/spec/DDS-XTYPES/20120202/dds-xtypes\\_type\\_definition.xsd](http://www.omg.org/spec/DDS-XTYPES/20120202/dds-xtypes_type_definition.xsd)
- [12] Atom Syndication Format (IETF RFC 4287); <http://www.ietf.org/rfc/rfc4287.txt>. Atom Publishing Protocol (IETF RFC 5023); <http://tools.ietf.org/rfc/rfc5023.txt>
- [13] Hypertext Transfer Protocol, version 1.1 (IETF RFC 2616); <http://tools.ietf.org/rfc/rfc2616.txt>
- [14] The WebSocket Protocol, version 1.1 (IETF RFC 6455); <http://tools.ietf.org/rfc/rfc6455.txt>.
- [15] JavaScript Object Notation (IETF RFC 4627); <http://www.ietf.org/rfc/rfc4627.txt>.
- [16] RSS Specification, version 2.0; <http://www.rssboard.org/rss-specification>.
- [17] Extensible Markup Language (XML), version 1.1, Second Edition (W3C recommendation, August 2006).
- [18] HTTP Authentication: Basic and Digest Access Authentication. IETF RFC 2617. <http://tools.ietf.org/html/rfc2617>
- [19] File expression matching syntax for fnmatch() ; POSIX fnmatch API (IEEE 1003.2-1992 section B.6)
- [20] DDS Data-Distribution Service fore Real-Time Systems version 1.2, Annex A: Syntax for DCPS Queries and Filters. <http://www.omg.org/spec/DDS/1.2>
- [21] DDS Security specification, version 1.0. <http://www.omg.org/spec/DDS-SECURITY/>
- [22] IETF RFC 4918 , “HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)” <http://tools.ietf.org/html/rfc4918>

- [23] IETF RFC 2617 “HTTP Authentication: Basic and Digest Access Authentication”  
<http://tools.ietf.org/html/rfc2617>
- [24] IETF RFC 6648 “Deprecating the “X-“ Prefix and Similar Constructs in Application Protocol”  
<http://tools.ietf.org/html/rfc6648>
- [25] IETF RFC 5849 “The OAuth 2.0 Authorization Framework, v2-31” <http://tools.ietf.org/html/draft-ietf-oauth-v2-31>
- [26] OpenID Authentication 2.0 – Final [http://openid.net/specs/openid-authentication-2\\_0.html](http://openid.net/specs/openid-authentication-2_0.html)