Date: January 2011

ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)

FTF Beta 1

OMG Document Number: **ptc/2011-01-02**

Standard document URL: **http://www.omg.org/spec/DDS-PSM-Cxx/1.0**

Associated Schema files*: **http://www.omg.org/spec/DDS-PSM-Cxx/20101101**

---

* original file: mars/2010-11-02

This OMG document replaces the submission document (mars/2010-12-20, 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 by August 29, 2011.

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

The FTF Recommendation and Report for this specification will be published on November 7, 2011. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

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.

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/technology/agreement.)

Table of Contents

Preface

OMG

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

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

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

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:
http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG 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
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

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

Typographical Conventions

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

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

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

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

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Scope

The purpose of this document is to specify the ISO/IEC C++ PSM for DDS. This new PSM provides a new C++ API for programming DDS which is clear, simple, expressive, safe, efficient, extensible and portable. The ISO/IEC-C++ PSM does not impact on-the-wire interoperability with other language mappings. The PSM API is defined by means of a set of C++ header files.

This PSM includes all DCPS conformance profiles defined in the DDS specification. In addition, it includes platform-specific mappings for:

- The programming interface specified by [DDS-XTypes]
- Accessing QoS profiles such as are specified in [DDS-CCM]

This specification only addresses the DCPS layer of the DDS specification. The optional DLRL layer may be addressed separately in a future specification.This specificatio also introduces a new C++ mapping for the DDS type system as specified in the Extensible and Dynamic Topic Types Specification [REF].

2 Conformance

This specification consists of this document as well as a set of C++ header files, references on the cover page. Both are normative. In the event of a conflict between them, the latter shall prevail.

2.1 Conformance Profiles

Conformance to this specification parallels conformance to the DDS specification itself and consists of the same conformance levels. For example, an implementation may conform to the DDS Minimum Profile with respect to this PSM, meaning that all of the programming interfaces identified by the DDS specification as pertaining to that conformance level must be implemented in this PSM. The one exception to this rule is the Object Model Profile, which defines the Data Local Reconstruction Layer (DLRL); DLRL is outside of the scope of this PSM.

In addition to the conformance level defined in the DDS specification itself, this PSM recognizes and implements the Extensible and Dynamic Types conformance level for DDS defined by the Extensible and Dynamic Topic Types for DDS specification.

This PSM furthermore defines methods to create Entities and to set their QoS based on the XML QoS libraries and profiles defined by the DDS for Lightweight CCM specification. Implementations that support these XML QoS profiles shall implement these operations fully; other implementations shall indicate failure with the DDS-standard UNSUPPORTED error. The Plain Language Binding for C++ defined in this specification represents an optional conformance point. Implementers may support either this Language Binding or the previously defined Plain Language Binding for C++ defined in [DDS-XTypes].

2.2 Programming Interfaces

Conformance to the C++ programming interfaces consists of the following conditions:

- The file names and relative locations of all C++ headers within the “dds” directory are normative. Those headers within “detail” subdirectories are excepted; they are not normative.
- All public symbol names within the ::dds:: namespace and its contained namespaces, including those names introduced into those namespaces by means of typedef declarations, are normative. Those names within “detail” namespaces are excepted; they are not normative.
- The distribution of the normative symbol names among the normative headers is itself normative, such that a source file that includes the header in which a given name is declared will continue to compile when that header is replaced with the corresponding header from a different DDS implementation.

The remainder of the files, declarations, and definitions contained within this specification's C++ programming interfaces constitute a reference implementation and a set of examples. They are not normative.

Conforming implementations shall not define implementation-specific extension programming interfaces within normative namespaces. They may, however, specialize normative templates defined by this specification.

3 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.

- [**C99**] C Programming Language (ISO/IEC 9899:1999)
- [**C++**] C++ Programming Language (ISO/IEC 14882:2003)
- [**DDS**] Data Distribution Service for Real-Time Systems Specification, version 1.2 (OMG document formal/2007-01-01)
- [**DDS-XTypes**] Extensible and Dynamic Topic Types, version 1.0 Beta 1 (OMG document ptc/2010-05-12)
- [**DDS-CCM**] DDS for Lightweight CCM, version 1.0 Beta 1 (OMG document ptc/2009-02-02).

4 Terms and Definitions

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

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 for Real-Time Systems (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 Local Reconstruction Layer

The optional portion of the DDS specification used to provide the functionality required for an application for direct access to data exchanged at the DCPS layer. This later builds upon the DCPS layer.

Platform-Independent Model (PIM)

An abstract definition of a facility, often expressed with the aid of formal or semi-formal modeling languages such as OMG UML that does not depend on any particular implementation technology.

Platform-Specific Model (PSM)

A concrete definition of a facility, typically based on a corresponding PIM, in which all implementation-specific dependencies have been resolved.

5 Symbols

This specification leverages some symbols of common usage whose meaning is reported in the table below:

Symbol	Meaning
<:	The symbol “<:” is the commonly used symbol to denote subtyping. Given two programming language type T and Q, we can say that Q <: T if any occurrence of T can be replaced by Q.
Foo<+T>	When Foo is a class parameterized on the type T, we use the notation Foo<+T> to indicate that Foo is covariant in T. This means that given Q <: T then Foo<Q> <: Foo<T>

Symbol	Meaning
	When no annotation is provided then the class is supposed to be invariant.
Foo<-T>	When Foo is a class parameterized on the type T, we use the notation Foo<-T> to indicate that Foo is contravariant in T. This means that given Q <: T then Foo<T> <: Foo<Q> When no annotation is provided then the class is supposed to be invariant.

6 Additional Information

6.1 Acknowledgements

The following companies submitted this specification:

- PrismTech Corporation, Ltd.
- Real-Time Innovations, Inc. (RTI)

7 ISO/IEC C++ Language DDS PSM (DDS-PSM-CXX)

7.1 Overview

The “ISO/IEC C++ Language DDS PSM” (DDS-PSM-Cxx) was motivated by mainly two reasons. First the IDL-derived C++ API for DDS does not integrate well with the C++ language and it does not leverage some of the features provided by the C++ language today universally supported by C++ compilers. Second, the current IDL-derived PSM suffers from the gap existing between the features available in IDL and those available in a programming language such as C++. Some examples of this gap are as simple as method overloading, yet, there are many other examples that we could make in comparing the expressiveness power of IDL versus that of native C++.

As a result this submission takes a complete fresh look at how a native C++ PSM can be derived from the DDS PIM. In doing so, it tries to balance two forces--derive an API that is as simple and safe as possible while retaining the structure of the PIM. In addition, the specification, while not requiring any C++0x features it will greatly benefits from it both in terms of further simplifying the programming of DDS applications, for instance using the new “auto” keyword” or by simplifying the actual implementation of the API.

7.2 Specification Organization

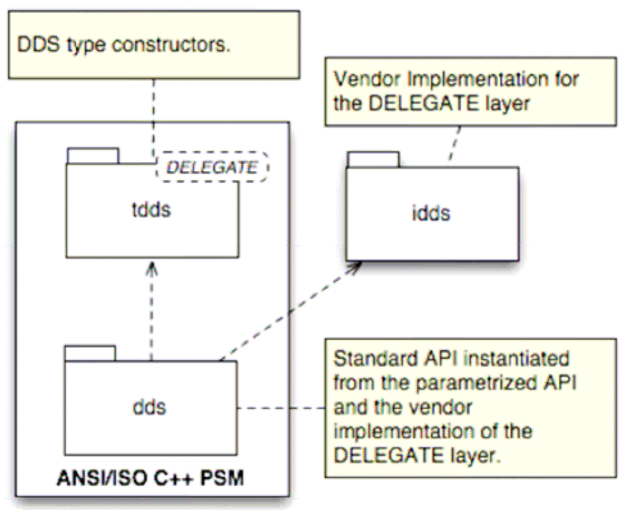


Figure 7.1 – Standard Packages Organization

The DDS-PSM-Cxx API is organized in two packages hierarchies, namely “tdds” and “dds”, as shown in Figure 7.1. All the classes included in the package hierarchy “tdds” are *type constructors*, meaning template classes parameterized with respect to a delegate type to which the implementation of the prescribed behavior is delegated—these delegates are to be provided by vendors as shown in Figure 7.1 and are used to instantiate the standard types as specified in the “dds” package by means of the type constructors. The structure and dependencies of the “tdds” and “dds” package hierarchies are shown in Figure 7.3 and Figure 7.2 respectively. The package organization reveals how various parts of the API are grouped into a package so to limit their dependencies. The DDS-PSM-Cxx organizes the various DDS classes in a set of packages that maximize the coherence among contained classes and minimize the dependencies across packages. The new organization assures that the API minimizes dependencies to the minimum by construction. In addition, the DDS-PSM-Cxx allow applications that use only the publishing or the subscribing functionalities to limit to the minimum required the files being included, thus speeding up compilation times.

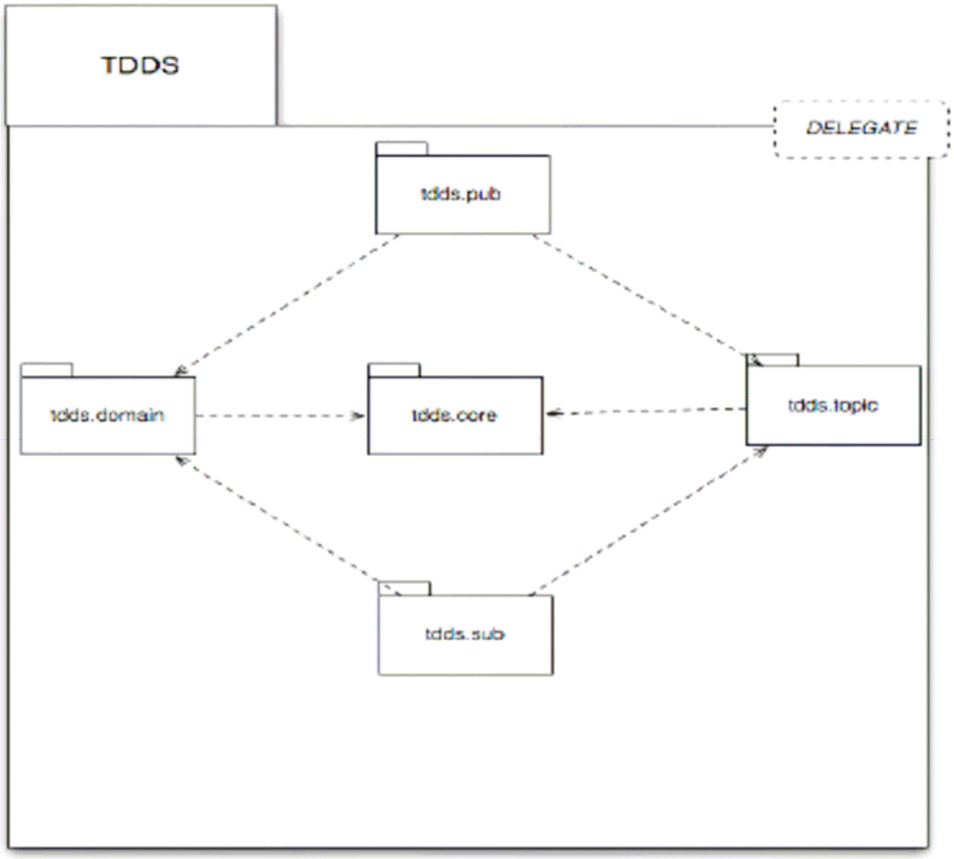


Figure 7.2 – Instantiated Package Structure

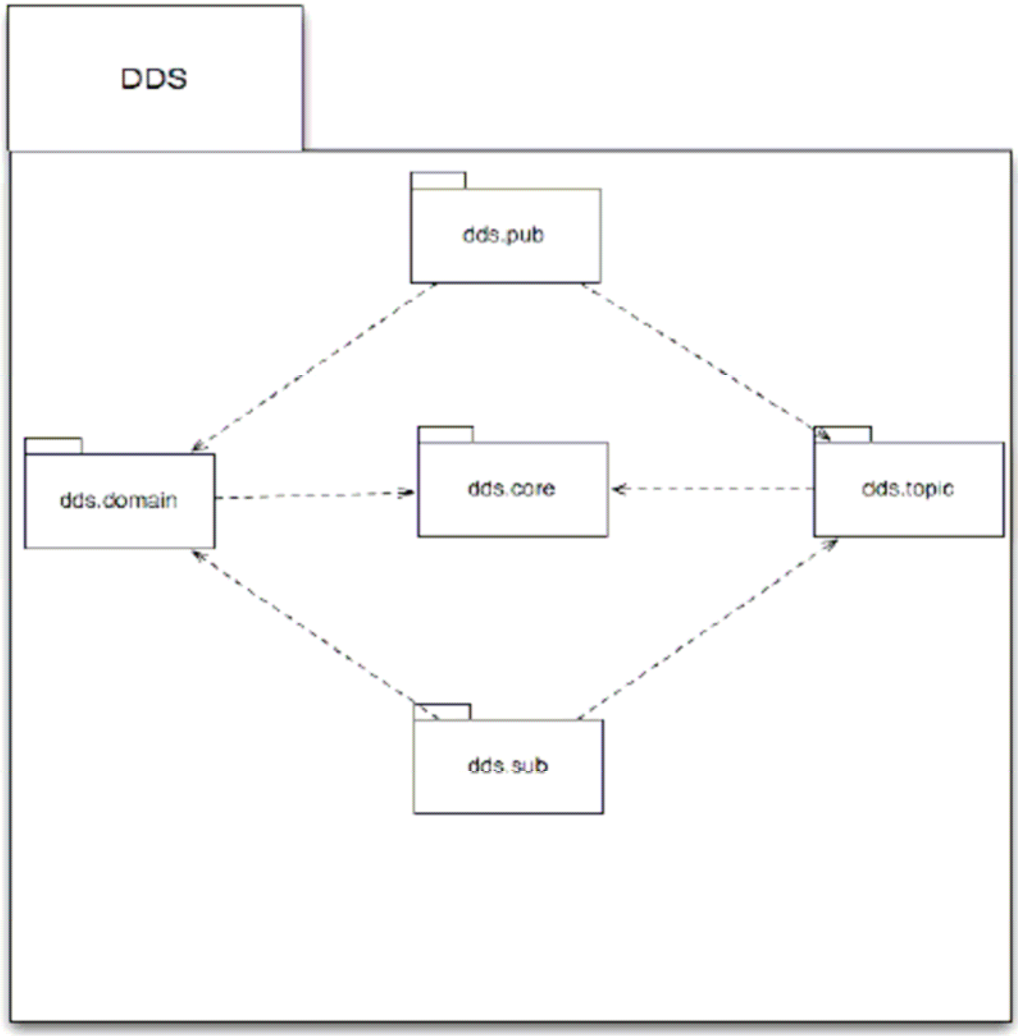


Figure 7.3 – Standard Package Organization

Figure 7.1 and Figure 7.2 show that the DDS-PSM-Cxx API is parameterized with respect to a DELEGATE. The delegation layer is provided by vendors and used to instantiate the DDS-PSM-Cxx API into a concrete API. The DDS-PSM-Cxx API instantiation is as simple as the instantiation of a set of C++ templates. This standard provides a non-mandatory reference implementation showing how that can be done. Under any circumstances, compliant implementation shall not change the DDS-PSM-Cxx API vendor-specific extensions shall be added only via DELEGATEs. The DDS-PSM-Cxx API provides a standard way of accessing vendor specific extensions.

Application source code imports the DDS API by including one or more header files from the dds/ directory hierarchy. There are three ways to do this, depending on how the application programmer wishes to manage file dependencies.

- The entire DDS API can be included at once:

  #include <dds/dds.hpp>

- Individual DDS modules can be included. These headers have the form dds/*module*/*module*_module.hpp. For example:

  #include <dds/pub/pub_module.hpp>

- Individual types can be included. These headers have the form dds/*module*/*ClassName*.hpp. For example:

  #include <dds/pub/DataWriter.hpp>

7.3 Concurrency and Reentrancy

It is expected that most Service implementations will support multi-threaded environments. Therefore, for the sake of portability, this PSM constrains the level of thread safety that applications may expect:

- All DataReader and DataWriter operations shall be reentrant.
- All Topic (and other TopicDescription extension interfaces), Publisher, Subscriber, and DomainParticipant operations shall be reentrant with the exception that close may not be called on a given object concurrently with any other call of any method on that object or on any contained object.
- All DomainParticipantFactory operations shall be reentrant with the exception that DomainParticipantFactory.close may not be called on a given object concurrently with any other call of any method on that object or on any contained object.
- All WaitSet and Condition (including Condition extension interfaces) operations shall be reentrant with the exception that their close() operations may not be invoked concurrently with any other method on the same object.
- Code within a DDS listener callback may not safely call any method on any DDS Entity but the one on which the status change occurred.
- Any method of any value type may be non-reentrant.

A Service implementation may choose to provide unspecified stronger guarantees than the rules above.

7.4 General Rules for Mapping the DDS PIM to the DDS-PSM-Cxx

This specification defines some general rules to map DDS PIM classes to DDS-PSM-Cxx classes. These rules are applicable to a subset of classes, luckily the most numerous, while special mapping is required for some of the DDS entities as described below.

7.4.1 Mapping Classes

As a general rule all classes included in the DDS PIM have to be mapped into a C++ class. The specific nature of this class depends on whether the DDS PIM element has reference or value semantics.

NOTE: An implication of this mapping is that no DDS PIM class ever maps to a C++ struct.

7.4.2 Mapping Primitive and Container Types

The table below provides a complete mapping between the types defined and used by the DDS PIM and the corresponding types used by the DDS-PSM-Cxx:

Table 7.1 – Primitive and Container Types Mapping

DDS Type	C++ Type
Boolean	bool
Char8	char
Char32	wchar_t
Byte	uint8_t
Int16	int16_t
UInt16	uint16_t
Int32	int32_t
UInt32	uint32_t
Int64	int64_t
UInt64	uint64_t
Float64	double
Float128	long double
Float32	float
string<Char8>	std::string
string<Char32>	std::wstring
sequence<T>	std::vector<T>
map<K, V>	std::map<K, V>
T[]	T[]

The above fixed-size integer types shall conform to the types of the same names as defined by [C99] in the header stdint.h.

- The presence of these types shall not be construed to require that DDS implementations only support [C99]-compliant platforms. Implementations for non-[C99]-compliant platforms shall provide their own conformant integer type definitions.
- It shall not be construed to imply the existence of any other definitions that would be found in the header stdint.h on a [C99]-compliant platform or even the existence of that header itself.

Note that these types are defined in the global namespace, not in the std namespace.

7.4.3 Mapping Parameters Passing and Parameters Return Rules

The DDS PIM defines parameters as being either IN/OUT/INOUT depending on whether the parameter has no side effect, is used only for side effect, or whether it provides data that then is changed by the invoked method. Likewise the PIM defines return types.

The table below provides a mapping between IN/OUT/INOUT for a generic type T, distinguishing between primitive and non-primitive types. To this end, container types are considered as non-primitive types.

PIM Native Type Parameter	DDS-PSM-Cxx Native Type Parameter
IN T	T
OUT T	T&
INOUT T	T&

PIM Native Return Type	DDS-PSM-Cxx Native Return Type
T	T

PIM Type Parameter	DDS-PSM-Cxx Type Parameter
IN T	const T&
OUT T	T&
INOUT T	T&

PIM Native Return Type	DDS-PSM-Cxx Native Return Type
T	One of the following, depending on whether the return parameter is an attribute or not. • T • const T&

7.4.4 Mapping Attributes

Attributes defined by DDS PIM classes have to be mapped into:

- Implementation-defined state,
- Accessors named after the attribute, and
- A constructor argument that allows initializing the attribute.

7.5 Core Package

The core package of the ISO/IEC C++ PSM for DDS (DDS-PSM-Cxx) defines the classes at the foundation of the API object model as well as all the DDS types used by all other modules. This section describes the most important classes of the package. The full list of mandatory classes is included in the appendix.

7.5.1 Object Model

The ISO/IEC C++ PSM for DDS (DDS-PSM-Cxx) is based on an object model that is structured in two different kinds of object types: reference-types and value-types.

7.5.1.1 Reference Types

All objects that have a reference-type have an associated shallow (polymorphic) assignment operator that simply changes the value of the reference. Furthermore reference-types are safe, meaning that under no circumstances can a reference point to an invalid object. At any single point in time a reference can either refer to the *null* object or to a valid object.

The semantics for Reference types is defined by the DDS-PSM-Cxx class dds::core::Reference. In the context of this specification the semantics implied by the ReferenceType is mandatory, yet the implementation provided as part of this standard is provided to show one possible way of implementing this semantics.

All DDS-PSM-Cxx reference-types store references to a delegate. To avoid imposing too many constraints on the actual implementation of the DDS-PSM-Cxx standard while ensuring that efficiency can be retained, all DDS-PSM-Cxx reference-types are template classes whose parameter is the DELEGATE. Each vendor will plug-in his implementation simply by providing a file that instantiates the DDS-PSM-Cxx API with its own delegates. Furthermore, by using this approach, the same API can be used without changes on multiple implementations. At the limit, it is possible for end-users to program to the OMG provided DDS-PSM-Cxx and then switch from one DDS to another by simply switching to use his own mapping file and his libraries. Finally, the PSM also provides weak references.

The table below lists all the DDS PIM classes that have reference semantics:

Table 7.2: DDS Classes with Reference semantics

Namespace		Class
tdds	core	• Entity • Condition • GuardCondition • ReadCondition • QueryCondition • Waitset
	domain	• DomainParticipant
	pub	• Publisher • DataWriter
	sub	• Subscriber • DataReader
	topic	• Topic

Namespace		Class
dds	pub	• AnyDataWriter
	sub	• AnyDataReader
	topic	• AnyTopic

7.5.1.2 Resource for Reference Types

Instances of reference types are created by the factory methods specified in the DDS PIM or (in the case of WaitSet and GuardCondition, which have no PIM-specified factory classes) by static factory methods in the classes themselves. Declaring an object of a reference type on the stack with its default constructor, without assigning to it the result of any factory method or other previously created object, initializes a null reference.

Resource management for some reference types might involve relatively heavyweight operating-system resources—such as e.g., threads, mutexes, and network sockets—in addition to memory. These objects therefore provide a method close() that shall halt network communication (in the case of entities) and dispose of any appropriate operating-system resources.

Users of this PSM are recommended to call close on objects of all reference types once they are finished using them. In addition, implementations may automatically close objects that they deem to be no longer in use, subject to the following restrictions:

- Any object to which the application has a direct reference (not including a WeakReference) is still in use.
- Any entity with a non-null listener is still in use.
- Any object that has been explicitly retained is still in use
- The creator of any object that is still in use is itself still in use.

7.5.2 Value Types

All objects that have a value-type have a deep-copy assignment and copy construction semantics. It should also be pointed out that value-types are not “pure-value-types” in the sense that they are immutable (as in functional programming languages). The DDS-PSM-Cxx makes value-types mutable to limit the number of copies as well limit the time-overhead necessary to change a value-type (note that for immutable value-types the only form of change is to create a new value-type).

The DDS-PSM-Cxx models all DDS PIM classes beyond what is listed in Table 7.2 as value-types. In other terms, QoS, Policy, Statuses, and Topic samples are all modeled as value-types.

7.5.3 Any Types

The DDS-PSM-Cxx has been designed to take advantage of the compile time polymorphism provided by C++ templates. As such, the whole standard interface only has a few virtual methods, and in general does not rely on inheritance but as opposed exploits delegation.

Since the DDS API requires at times to pass DDS entities without exposing the complete type, while other times requires to store in containers list of objects of different types, the DDS-PSM-Cxx provides a selection of “Any” types.

These Any types safely store references in generic container objects without losing type information while at the same time exposing some type-independent operations.

7.5.4 Status Classes

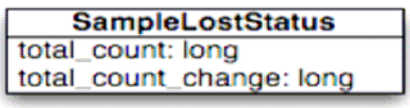
The DDS-PSM-Cxx mapping for the status classes as defined in the DDS v1.2 specification is obtained by applying the generic mapping rules described in Section 7.4 with the following exception:

- inheritance from the root status class has been ignored.

The reason for ignoring the inheritance from the root **Status** class is that this super-class does not provide any common behavior, or common state.

Status classes are part of the dds::core::status namespace.

As an example, consider the following PIM Status class:



Based on the mapping rules defined so far, the associated DDS-PSM-Cxx class would be the following:

```
namespace dds { namespace core { namespace status {
template <typename D>
class SampleLostStatus : public  dds::core::Value<D>{
public:
    SampleLostStatus();
    SampleLostStatus(uint32_t total_count, uint32_t total_count_change);

public:
     uint32_t total_count() const;
     uint32_t& total_count();
     void total_count(uint32_t total_count);
};  }  }  }
```

The full set of status classes is included in the mandatory standard headers in the file dds/core/status/Status.hpp.

7.5.5 Error Codes

Table 7.3: Mapping between PIM Error Codes and C++ Exceptions

DDS PIM Return Code	DDS-PSM-Cxx Exception Class	Std C++ Parent Exception
RETCODE_OK	Normal return; no exception	

RETCODE_NO_DATA	An informational state attached to a normal return; no exception	
RETCODE_ERROR	Error	std::logic_error
RETCODE_BAD_PARAMETER	InvalidArgument Error	std::invalid_ argument
RETCODE_TIMEOUT	TimeoutError	std::runtime_error
RETCODE_UNSUPPORTED	UnsupportedError	std::logic_error
RETCODE_ALREADY_DELETED	AlreadyClosed Error	std::logic_error
RETCODE_ILLEGAL_OPERATION	IllegalOperation Error	std::logic_error
RETCODE_NOT_ENABLED	NotEnabledError	std::logic_error
RETCODE_PRECONDITION_NOT_MET	PreconditionNot MetError	std::logic_error
RETCODE_IMMUTABLE_POLICY	ImmutablePolicy Error	std::logic_error
RETCODE_INCONSISTENT_POLICY	Inconsistent PolicyError	std::logic_error
RETCODE_OUT_OF_RESOURCES	OutOfResources Error	std::runtime_error

7.5.6

The DDS-PSM-Cxx maps error codes to C++ exceptions defined in the dds::core namespace and inheriting from a base Exception class and the appropriate standard C++ exception. Table 7.3 lists the mapping between error codes as defined in the DDS PIM and C++ exceptions as used in this specification. Exceptions have value semantics, this means have to always have deep copy semantics. The full list of exceptions is included in the file dds/core/Exceptions.hpp.

7.5.7 Time and Duration

This PSM maps the DDS `Time_t` and `Duration_t` types into the value types `Time` and `Duration` respectively. In addition to providing their seconds and nanoseconds state through accessor and mutator methods, these classes provide a small number of convenience operations:

- `Time` objects can be incremented by durations expressed as seconds, nanoseconds, milliseconds, or `Duration` objects.
- `Time` objects can be converted to and from times expressed in milliseconds (or other units) as integer types.
- `Duration` objects can be incremented by durations expressed as seconds, nanoseconds, milliseconds, or `Duration` objects.
- `Duration` objects can be converted to and from durations expressed in milliseconds (or other units) as integer types.

7.6 QoS Packages

The QoS package provides all definitions for Policy and QoS.The DDS-PSM-Cxx provide extensible policy and extensible QoS. This means that vendor can easily add additional attributes to policy as well as new policies to Qos. All of this without requiring changes in the public API. As explained above, the PSM uses the "operator ->", or equivalently the "delegate()" method to access vendor-specific extensions.

7.6.1 Policy Classes

The DDS-PSM-Cxx mapping for the policy classes as defined in the DDS v1.2 specification is obtained by applying the generic mapping rules described in Section 7.4 with the following guidelines:

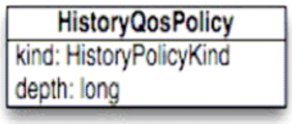
- the inheritance from the root Policy class has been ignored
- the trailing "QosPolicy" has to be discarded from the name as redundant.
- Policy kind is represented with a C++ enumeration and an associated constructor type as shown in the example below.

Policy classes are part of the dds::qos namespace and the Policy Name and Policy ID are to be provided by specialization of the following trait classes:

```
namespace dds { namespace qos {
  template <typename Policy>
  class policy_id {
  public:
  enum {
    id = -1
  };
};
template <typename Policy>
class policy_name {
};

} }
```

As an example let's consider the following Policy class as modeled in the DDS PIM:



This would map to the following set of types

```
namespace dds { namespace qos {
namespace HistoryKind {
enum Type {
     KEEP_LAST,
     KEEP_ALL
};} } }


namespace tdds { namespace qos {

template <typename D>
class History : public dds::core::Value<D> {

public:
    History();

    History(HistoryKind::Type kind, int32_t depth);

    HistoryKind::Type kind() const;
    HistoryKind::Type& kind();
    void kind(HistoryKind::Type kind);

    int32_t depth() const;
    int32_t& depth();
    void depth(int32_t depth);

    static History KeepAll();
    static History KeepLast(uint32_t depth);
    };
    } }
```

As shown in the example above, when a policy presents a variability that is captured at a PIM-Level by a kind, the DDS-PSM-Cxx captures this variability into two ways, first it associates an enumeration with the Policy defining a code for the variation (as it was done in the IDL PSM), then, it defines a set of helper methods to construct the possible variants. The full set of policies is included in the mandatory standard headers in the file dds/qos/Policy.hpp.

7.6.2 Entity Class

The Entity class is the root for all DDS entities, as specified in the DDS v1.2 specification. Since an Entity is a reference type, its resources are automatically managed by the middleware. Specifically, the resources associated with the entity will be reclaimed either when the number of live reference from the user application to the entity drops to zero, or when the user explicitly invokes the method **close**.

7.6.2.1 QoS and Profiles

The *DDS for Lightweight CCM* specification [DDS-CCM] defines a format for QoS libraries and profiles. This PSM provides the following APIs for accessing these:

- Entity classes provide a method to set their QoS based on the names of a QoS library and profile.
- Each Entity factory interface—DomainParticipantFactory, DomainParticipant, Publisher, and Subscriber—provides methods to create new “product” Entities and to set their default QoS based on the names of a QoS library and profile.

7.7 Domain Package

The domain package defines the DomainParticipantFactory, DomainParticipant, and DomainParticipantListener. For a complete reference see the standard header files.

7.8 Topic Package

The topic packaged defines the classes related to topic management. As such it provides definitions for the Topic, TopicDescription, ContentFilteredTopic, MultiTopic, and the TopicListener.

The topic class is parameterized in the topic type and transparently performs the registration of type support.

If we consider the RadarTrack topic type used in the example above, we can create a topic for this type as follows:

```
DomainParticipant dp
ThePartcipantFactory().create_participant(domainId);
dds::topic::Topic<RadarTrack> topic =
    dp.create_topic<RadarTrack>("RadarTrackTopic");
```

If the topic is to be created with a QoS different from the default, than the code above would be:

```
DomainParticipant dp =
TheParticipantFactory().create_participant(domainId);
dds::qos::TopicQos tqos = dp.default_topic_qos();
tqos << Reliability::Reliable() << Ownership::Exclusive();
dds::topic::Topic<RadarTrack> topic =
    dp.create_topic<RadarTrack>("RadarTrackTopic", "RadarTrack",
tqos);
```

7.9 Pub Package

The publication (pub) package defines all the classes associated with the production of data. As such, it defines the Publisher, the DataWriter and their associated listeners as well as any types.

The mandatory classes are specified in the standard header files. Below, we focus on the specifics of the DataWriter class.

7.9.1 Data Writer Class

The DataWriter class is parametrized with respect to the delegate and the topic type that it writes. The class provides several different overloaded methods for writing data by providing single samples or iterators over samples.

7.10 Sub Package

The subscription (sub) package defines all the classes associated with the consumption of data. As such, it defines the Subscriber, the DataReader and their associated listeners as well as any types. The mandatory classes are specified in the standard header files. Below, we focus on the specifics of the DataReader class.

7.11 Extensible and Dynamic Type Support Package

The Extensible and Dynamic Type Support (xtypes) package defines all the classes associated with the definition of extensible topics, such as annotations and the definition and manipulation of dynamic types. As such, this package introduces all classes necessary for describing dynamic types and their attributes, creating and annotating them.

7.12 Example

This section provides an example for full application writing and reading RadarTracks topics.

```
// ================== DataWriter ===================
   try {
      DomainId id = 0;
      DomainParticipant dp =
             TheParticipantFactory().create_participant(id);

      pub::qos::PublisherQos pqos;
      pqos << policy::Partition("Tracks");

      pub::Publisher pub =
             dp.create_publisher(pqos);

      topic::qos::TopicQos tqos;
      tqos << policy::Reliability::Reliable()
           << policy::Durability::Transient()
           << policy::History::KeepLast(10)
           << policy::TransportPriority(14);



      dds::topic::Topic<RadarTrack> topic =
             dp.create_topic<RadarTrack>("TrackTopic", "RadarTrack", tqos);

      pub::qos::DataWriterQos dwqos(tqos);

      pub::DataWriter<RadarTrack> dw =
             pub.create_datawriter(topic, dwqos);

      RadarTrack track("alpha", 100, 200);

      dw.write(track);
      // or
      dw << track;

 } catch (const dds::core::Exception& e) { }

// ================== DataReader===================

   try {
      DomainId id = 0;
      DomainParticipant dp =
             TheParticipantFactory().create_participant(id);

      sub::qos::SubscriberQos sqos;
      sqos << policy::Partition("Tracks");

      sub::Subscriber sub =
             dp.create_subscriber(sqos);

      topic::qos::TopicQos tqos;
      tqos << policy::Reliability::Reliable()
           << policy::Durability::Transient()
           << policy::History::KeepLast(10)
           << policy::TransportPriority(14);

      dds::topic::Topic<RadarTrack> topic =
```

```
        dp.create_topic<RadarTrack>("TrackTopic", "RadarTrack", tqos);

    sub::qos::DataReaderQos dwqos(tqos);

    sub::DataReader<RadarTrack> dr =
        sub.create_datareader(topic, drqos);
    std::vector<RadarTrack> samples(MY_MAX_LEN);
    std::vector<SampleInfo> info(MY_MAX_LEN);
    dr.read(samples.begin(), info.begin(), MY_MAX_LEN);


} catch (const dds::core::Exception& e) { }
```

8 Improved Plain Language Binding for C++

8.1 Type Mapping

The type system for DDS topic types is defined by the *Extensible and Dynamic Topic Types for DDS* specification [DDS-XTypes].

This section defines the set of rules to be used in order to map abstract DDS topic types into C++ types that can be used by application programmers. Those aspects of the DDS Type System that are not addressed below are as specified in the Plain Language Binding as defined by [DDS-XTypes] (which in turn is defined in terms of an IDL-to-C++ mapping).

The example below illustrates the application of these simple rules.

8.1.1 Mapping Aggregation Types

DDS aggregation types shall be mapped to a C++ class. Contained attributes shall be encapsulated: accessors shall be provided following the rules described in Section 7.4. The representation of internal state is unspecified.

8.1.2 Mapping Primitive and Collection Types

IDL primitive and collection types used to define a topic type shall be mapped to C++ following the rules listed in Table 7.1.

8.1.3 Mapping Enumerations

IDL enumerations shall be mapped into C++ enumerations with exactly the same enumeration name and enumeration constants.

8.1.4 Mapping Optional Attributes

Attributes annotated though the @optional annotation are mapped to a template instantiation of the class dds::core::optional<T> with T equal to the type attribute would normally map as per the rules specified above.

8.1.5 Mapping Shared Attributes

Attributes annotated through the @shared annotation are mapped to a pointer of the type they would normally map as per the rules specified above.

8.2 Example

This section provides a simple yet representative example demonstrating the ISO/IEC mapping for DDS types.

<table>
<tr><th>Topic Type Declaration (IDL)</th><th>C++ Reprentation</th></tr>
<tr><td>

```
struct RadarTrack {
    string id;
  long x;
  long y;
  long z; //@optional
    sequence<octet> plot; //@shared
};
```

</td><td>

```
class RadarTrack {
public:
    RadarTrack();
    RadarTrack(const std::string& id,
         int32_t x, int32_t y,
                int32_t z,
         std::vector<uint8_t>* plot);
public:
    std::string id() const;
  void id(const std::string& s);

    int32_t x() const;
  void x(int32_t v);

  int32_t y() const;
  void y(int32_t v);

  dds::core::optional<int32_t> z() const;
    void z(int32_t v);
  void z(const
      dds::core::optional<int32_t>& z)

  std::vector<uint8_t>* plot() const;
  void plot(std::vector<uint8_t>* p)

// State representation is implementation
// dependent.

};
```

</td></tr>
</table>