Report of the  
Java 5 Language PSM for DDS Finalization Task Force 2.0  
to the  
OMG Platform Technical Committee  
10 November 2012

Document Number: ptc/2012-12-06

Task Force Chair: Sumant Tambe (RTI)

Specification

Revised specification (clean): ptc/2012-12-01

Revised specification (change-bar): ptc/2012-12-02

Accompanying documents

omgdds.jar: ptc/2012-12-03 Normative

omgdds\_src.zip: ptc/2012-12-04 Normative

**Inventory: ptc/2012-12-05 Non-normative**

**Issue\_diffs.zip ptc/2012-12-08 Non-normative**

Template: omg/09-06-01

Table of Contents

Summary of DDS-PSM-Java FTF Activities 1

Formation 1

Revision / Finalization Task Force Membership 1

Issue Disposition: 1

Voting Record: 3

Summary of Changes Made 4

Disposition: Resolved 5

OMG Issue No: 15966 6

Title: XML-Based QoS Policy Settings 6

OMG Issue No: 15968 8

Title: formal description of how topic types are mapped to Java classes needed 8

OMG Issue No: 16529 10

Title: Modifiable Types should be removed and replaced by values (e.g. immutable types) 10

OMG Issue No: 16531 13

Title: Getting rid of the Bootstrap object 13

OMG Issue No: 16535 15

Title: Large Number of Spurious Import 15

OMG Issue No: 16536 16

Title: QoS DSL Needed 16

OMG Issue No: 16587 19

Title: API Should Avoid Side-Effects, e.g. Remove Bucket Accessors 19

OMG Issue No: 17065 21

Title: Class for Query Expression 21

OMG Issue No: 17204 23

Title: Obsolete EntityQos interface name 23

OMG Issue No: 17302 24

Title: Implement Java5 Closeable interface 24

OMG Issue No: 17303 25

OMG Issue No: 17304 26

Title: Improve compile-time type safety of EntityQos 26

OMG Issue No: 17415 27

Title: Implement java.io.Closeable in Sample.Iterator 27

OMG Issue No: 18285 28

Title: Redundant "QoSPolicy" Suffix on Policy Types. 28

Disposition: Deferred 29

Disposition: Closed, no change 30

Disposition: Duplicate/merged 31

# Summary of DDS-PSM-Java FTF Activities

## Formation

* Chartered By: Platform TC
* On: 16 December, 2011; Santa Clara, CA
* Comments Due Date: 31 August, 2012
* Report Due Date: 10 November, 2012

## Revision / Finalization Task Force Membership

|  |  |  |
| --- | --- | --- |
| **Member** | **Organization** | **Status** |
| Angelo Corsaro | PrismTech | Charter |
| Fabrizio Morciano | Selex-SI | Charter |
| Ken Rode | Gallium | Charter |
| Sumant Tambe | Real-Time Innovations (RTI) | Charter (chair) |
| Virginie Watine | Thales | Charter |
| Clark Tucker | Twin Oaks Computing, Inc. | Joined 23 March 2012 |
| Adam Mitz | OCI | Joined 23 March 2012 |

## Issue Disposition:

|  |  |  |
| --- | --- | --- |
| **Disposition** | **Number of Occurrences** | **Meaning of Disposition** |
| Resolved | 14 | The RTF/FTF agreed that there is a problem that needs fixing, and has proposed a resolution (which may or may not agree with any resolution the issue submitter proposed) |
| Deferred | 0 | The RTF/FTF agrees that there is a problem that needs fixing, but did not agree on a resolution and deferred its resolution to a future RTF/FTF. |
| Transferred | 0 | The RTF/FTF decided that the issue report relates to another specification, and recommends that it be transferred to the relevant RTF. |
| Closed, no change | 0 | The RTF/FTF decided that the issue report does not, in fact, identify a problem with this (or any other) OMG specification. |
| Closed, Out of Scope | 0 | The RTF/FTF decided that the issue report is an enhancement request, and therefore out of scope for this or any future FTF or RTF working on this major version of the specification. The RTF/FTF has closed the issue without making any specification changes, but RFP or RFC submission teams may like to consider these enhancement requests when proposing future new major versions of the specification. |
| Duplicate or merged | 0 | This issue is either an exact duplicate of another issue, or very closely related to another issue: see that issue for disposition. |

## Voting Record:

|  |  |  |
| --- | --- | --- |
| **Poll No.** | **Closing date** | **Issues included** |
| 1 | 26 October 2012 | 15968, 15966, 16529, 16531, 16536, 17065, 17204, 17302, 17304, 17415 |
| 2 | 1 November 2012 | 17303 |
| 3 | 5 December 2012 | 16535, 16587, 18285 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Voter** | **Vote in poll 1** | **Vote in poll 2** | **Vote in poll 3** |
| Angelo Corsaro | Yes to all | Yes | Yes |
| Fabrizio Morciano | Yes to all | Abstain | Abstain |
| Ken Rode | Abstain | Abstain | Abstain |
| Sumant Tambe | Yes to all | Yes | Yes |
| Virginie Watine | Yes to all | Yes | Yes |
| Clark Tucker | Yes to all | Yes | Yes |
| Adam Mitz | Abstain | Abstain | Yes |

## Summary of Changes Made

The DDS-PSM-Java FTF made changes that:

* Corrected features that impeded implementation of the specification
* Clarified ambiguous aspects of the specification, especially with respect to certain error-prone constructions
* Provided additional convenience for users, especially those upgrading from previous versions of DDS

Here is the FTF's categorization of the resolutions applied to the specification according to their impact on the clarity and precision of the specification:

|  |  |  |
| --- | --- | --- |
| Extent of Change | Number of Issues | OMG Issue Numbers |
| **Critical/Urgent** - Fixed problems with normative parts of the specification which prevented implementation work | **0** | **None** |
| **Significant** - Fixed problems with normative parts of the specification that raised concern about implementability | **6** | 15966,16529,16531,16536,  17303, 18285 |
| **Minor** - Fixed minor problems with normative parts of the specification | **6** | 16530, 17065, 17302, 17304, 17415, 16535, |
| **Support Text** -Changes to descriptive, explanatory, or supporting material. | **2** | 15968, 17204 |

Disposition: Resolved

OMG Issue No: 15966

Title: XML-Based QoS Policy Settings

Source:

PrismTech (Dr. Angelo Corsaro, PhD., angelo.corsaro(at)prismtech.com)

**Nature:** Uncategorized Issue

**Severity:** Minor

Summary:

ISSUE

The newly introduced XML Based Policy configuration adds new methods in the core DDS entities that allow to fetch QoS from XML filers. This solution is not ideal since if generalized, e.g. QoS configuration from an URI, JSON stream, etc., would lead to an explosion of the core DDS API.

Proposed Resolution:

The suggestion is to remove the added methods from the core API and use instead a Builder pattern (of some form).

A sketch of the suggested change is provided below:

PolicyBuilder builder = PolicyBuilder::load("XMLBuilder");

TopicQos tqos = builder.topic\_qos(file\_name, profile\_name);

Notice that the suggested approach allows to easily extend the supported format for QoS representation w/o any impact on the core DDS API and overall facilitate the support for multiple approaches.

Revised Text:

A QoSProvider interface has been added in response to this isse.

The updated section 7.2.5.4 is as follows: The QosProvider interface allows Entity’s Qos to be obtained from the names of QoS library and profile. The Qos library source is provided as a uniform resource identifier (URI). Conforming implementation must support “file://” prefix. For instance, “file:///path/to/qos/library”.

An instance of QosProvider is obtained from the ServiceEnvironment. For example,

serviceEnv.newQosProvider(String uri, String profile);

The uri parameter uses the standard uri syntax. The profile parameter identifies a uniquely identified profile in the document referred by the uri.]unquote

See revision #196: <https://code.google.com/p/datadistrib4j/source/detail?r=196>

These changes are also available in file diff\_omg\_issue\_15966.txt in ptc/2012-12-08 (issue\_diffs.zip).

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 15968

Title: formal description of how topic types are mapped to Java classes needed

Source:

PrismTech (Angelo Corsaro, [angelo.corsaro@prismtech.com](mailto:angelo.corsaro@prismtech.com))

Summary:

The DDS-PSM-Java currently provides examples of the new mapping from the DDS type system to the Java programming language but does not provide a formal description of how topic types are mapped to Java classes. This under-specification should be filled to align the DDS-PSM-Java with the DDS-PSM-Cxx and to ensure that different/old mappings are not used by DDS implementations.

**Revised Text:**

Section 9 has been introduced in the specification document that describes the mapping formally. The section has been reproduced verbatim except the example for brevity.

9. Improved Plain Language Binding for Java

9.1 TypeMapping

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 Java 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-Java mapping [Java-MAP]).

9.1.1 Mapping Aggregation Types

DDS aggregation types shall be mapped to a final Java class. Contained attributes shall be encapsulated. Java Bean style accessors shall be provided. Special mapping rules for boolean properties are allowed. The representation of internal state shall be private.

9.1.2 Mapping Sequences and Arrays

Unbounded DDS sequences are mapped to Collection<E> interface. The state is encapsulated and getters/setters are provided through bean style property accessors. Bounded sequences and arrays are mapped to Java arrays.

**Discussion:**

Note that DDS-PSM-Cxx does not *require* implementations to use the new Plain Language Binding it defines; that binding is an optional conformance point. The Java Type Representation will need to change significantly to address this issue. It is a monumental undertaking and the benefit of that is not clear. This issue is therefore rejected.

However, a clarification has been added in response to this issue. The specification document now describes plain language binding for Java in section 9. Majority of plain language binding for Java is borrowed from the X-Types specification (ptc/2010-05-12) and IDL-to-Java mapping (formal/2008-01-11). Two exceptions and a detailed example are added in Section 9 of the specification.

1. Attributes in IDL map to getter/setter methods. Java Bean style convention will be followed.
2. Unbounded sequences will be mapped to Java interface java.util.Collection<E> and bounded sequences and arrays will map to Java arrays.

**Proposed Resolution: Resolved**

**Resolution:** Resolved.

OMG Issue No: 16529

Title: Modifiable Types should be removed and replaced by values (e.g. immutable types)

Source:

PrismTech (Angelo Corsaro, [angelo@icorsaro.net](mailto:ANGELO@ICORSARO.NET))

**Nature:** Architectural

**Severity:** Major

Summary:

The DDS-PSM-Java introduces modifiable versions for conceptually immutable classes as a way to save a few object allocations. However this is done for QoS which are not changed so often and that are overall very "thin" object.

Discussion:

[Angelo] The proposed resolution is to get rid of these modifiable types and to ensure that value types are used everywhere. Although this solution might lead to think that immutable types induce the creation of more objects this is not necessarily the case if the API is designed carefully as done for policies and QoS on simd-java (see [git@github.com:kydos/simd-java.git](mailto:git@github.com:kydos/simd-java.git)).

As an example, with the API included in the current DDS-PSM-Java modifying a policy would require the following steps:

  // Get unmodifiable QoS for inspection:

  DataWriterQos udwq = dw.getQos();

  // Get the Modifiable QoS

  ModifiableDataWriterQos mdwq = udwq.modify();

  // Modify the Qos

  mdwq.setReliability(...);

With immutable Policies and QoS the same code could be rewritten as follows:

 DataWriterQos dwq = dw.getQos().with(Reliability.Reliable());

But you could also do:

 DataWriterQos dwq = dw.getQos().with(

Reliability.Reliable(),

Durability.Transient());

Notice that both code fragments lead to the lead the creation of a single new object. Yet the proposed approach not only gets rid of the complexity of the mutable objects, but it also get rids of the danger introduced by having mutable objects into multi-threaded applications. In summary, the proposed change (1) simplifies the API, (2) makes it safer, and (3) does not introduce runtime overhead (it actually allows for an higher degree of object sharing and thus better space efficiency).

NOTE: Cloneable interface: No need to implement the interface once the mutable package is removed

**Revised Text:**

Section 7.2.3 (Value Types) in the specification has been updated as a result of this issue resolution. In particular, the last paragraph in section 7.2.3 and two sub-bullets under it have been removed because the discussion is no longer pertinent. Previously it used to read as follows: Quote [

Some value types come in modifiable and unmodifiable varieties—notably QoS and QoS policies. The “modifiable” interface extends the “unmodifiable” one.

* The latter provides an operation modify that returns an instance of the former. Classes that implement the unmodifiable interface but not the modifiable one shall implement this operation to return a new modifiable object containing a copy of the state of the target unmodifiable object. Classes that implement the modifiable interface shall return a pointer to themselves.
* Modifiable value types with unmodifiable counterparts have an inverse operation: finishModification. In many cases, calling this operation is optional, as modifiable interfaces extend unmodifiable ones. However, in some cases, a truly unmodifiable object is desirable, such as when it will be shared among threads without locking.] unquote

1. **Rationale:**The biggest occurrence of the bucket pattern—QoS policies—now use a DSL as described in issue #16536. Additionally, a PolicyFactory has been added as described in issue #15966. See Section 7.2.5.1.
2. ModifiableDuration has been removed.
3. Sample.getSourceTimestamp return changed from ModifiableTime to Time
4. ModifiableInstanceHandle is used in statuses and in lookupInstance, where it needs to support being copied over. However, other values—like the nil handle constant, Entity instance handles, and the result of registerInstance—should not be changed. All of these APIs can be performance-sensitive.
5. Implemented a lighter-weight version of this pattern specifically for Time and InstanceHandle rather than retaining it for all value types. *To avoid race conditions, these classes should NOT be related by inheritance.*
6. Removed AnnotationDescription, renaming ModifiableAnnotationDescriptor to AnnotationDescriptor. Removed MemberDescription, renaming ModifiableMemberDescriptor to MemberDescription.

Also see FTF1 report (ptc/2011-10-05) for earlier discussion.

There are no source diffs for this issue. Please see the issue diffs for #16536 and #15966.

Proposed Resolution: Resolved

Resolution: Resolved

OMG Issue No: 16531

Title: Getting rid of the Bootstrap object

Source:

PrismTech (Angelo Corsaro, [angelo@icorsaro.net](mailto:ANGELO@ICORSARO.NET))

**Nature:** Architectural

**Severity:** Critical

Summary:

The Bootstrap class is a pain for users and is in place only to allow users to run 2 different DDS implementations on the same application.  The introduction of the Bootstrap object makes it impossible to use natural constructors for creating DDS types, even for types such as Time and Duration.

As one of the main goal of the new DDS PSM was to simplify the user experience and make the API as simple and natural as possible, it seems that the introduction of the Bootstrap object goes exactly on the opposite direction—all of this to be able to cover the case in which a user wants 2 different DDS implementation on the same application. Considering the wire-protocol interoperability this use case seems marginal and perhaps does not even count for 1% of DDS uses.

**Revised Text:**The following text has been added in section 7.2.1 in the specification.

ServiceEnvironement provides factory mehods for the following objects: *DynamicTypeFactory, WaitSet, GuardCondition, TypeSupport, Time, Duration, and InstanceHandle.* It also provides helper functions allStatuses and noStatuses to create special instances of Status objects.

**Discussion:**

The concern behind this issue is valid but the current proposed resolution is not acceptable. The Bootstrap class has been renamed ServiceEnvironement. See Section 7.2.1. The functionality offered by this class is valuable and the details are discussed in FTF1 report (ptc/2011-10-05). In short, this class is useful

1. To supports multiple DDS implementations in the same process. DDS-DDS bridge from one vendor to another
2. To support OSGi, and J2EE containers.

The ServiceEnvironment object need not be passed around because it is very easy access it as long as there is a DDSObject “around”. Every DDSObject provides an interface to retrieve the parent ServiceEnvironment that created the object. Given a ServiceEnvironment object, other objects can be created in two ways. Two example of creating a WaitSet are given.

ServiceEnvironment env = (DDSObject) obj.getServiceEnvironment();

WaitSet.newWaitSet(env); // (1)

env.getSPI().newWaitSet(); // (2)

Number of occurrences of the ServiceEnvironment are reduced as follows:

1. Instances of Status classes are *created from factory methods on the corresponding Entity interfaces.*
2. Instances of built-in topic data types are created using a factory method in DomainParticipant: ParticipantBuiltinTopicData, BuiltinTopicKey

The following instance is necessary and can’t be removed:

1. To access per-DDS-implementation singletons: DomainParticipantFactory

See Revision 202<https://code.google.com/p/datadistrib4j/source/detail?r=202>

The code changes are also available in diff\_omg\_issue\_16531.txt in ptc/2012-12-08 (issue\_diffs.zip).

**Proposed Resolution:** Resolved

**Resolution:** Resolved

OMG Issue No: 16535

Title: Large Number of Spurious Import

Source:

PrismTech (Dr. Angelo Corsaro, PhD., angelo.corsaro(at)prismtech.com)

**Nature:** Revision

**Severity:** Minor

Summary:

The dds-psm-java makes use of import as a way to take care of the @link directive on Javadoc. This is not a good practice and it is better to use the fully qualified type name on the @link javadoc directive

Proposed Resolution:

Clean up all the spurious import and use fully qualified types on the @link directives.

Revised Text:

There are no changes in the specification document for this issue.

**Discussion:**

Spurious import statements have been removed (as indicated by the Checkstyle plugin for Eclipse). Java doc comments are updated to use the fully qualified class names. This issue does not affect the specification text.

See Revision 181: <https://code.google.com/p/datadistrib4j/source/detail?r=181>

See Revision 210:<https://code.google.com/p/datadistrib4j/source/detail?r=210>

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 16536

Title: QoS DSL Needed

Source:

PrismTech (Dr. Angelo Corsaro, PhD., angelo.corsaro(at)prismtech.com)

**Nature:** Revision

**Severity:** Significant

Summary:

The absence of a DSL for facilitating the correct creation of QoS (in QoS classes such as:

TopicQos, DataWriterQos, etc.) in the

dds-psm-java not only makes QoS manipulation cumbersone, but it also

introduces potential for errors.

Proposed Resolution:

Define a QoS DSL for the dds-psm-cxx which might look like this:

TopicQos topicQos =

(new TopicQos())

.with(Reliability.Reliable(), Durability.Transient());

This is also legal:

TopicQos topicQos =

(new TopicQos())

.with(Reliability.Reliable())

.with(Durability.Transient());

- These class should implement the Comparable interface as they need to

provide a total order... Otherwise how can one do RxO?

Revised Text:

Section 7.2.5.3 in the specification describes the QoS DSL as follows:

Modifying QoS objects and their constituent policies is disallowed but a QoS DSL shall support creation of new QoS objects and policies from the existing objects using Java fluent interface design. QoS classes shall provide withPolicy and withPolicies methods which accept one or more policy objects to create a new QoS objects. Policy classes shall provide *with* methods to specify policy parameters and to create new policy objects from the existing ones. Each *with* method call will create a new policy object because the target object of the method call is immutable. The *with* methods shall support method chaining (QoS DSL).

**Example (non-normative)**

**PolicyFactory pf = … // object policy factory reference**

**ResourceLimits rl = pf.ResourceLimits().withMaxSamples(P).withMaxInstances(Q);**

* Design Rationale (non-normative)
* The getQos operation can operate maximally efficiently: it need not allocate any memory or perform any copies.
* The immutable result of getQos can be used safely concurrently from multiple threads.
* The getQos and setQos methods form a conventional Java-Bean-style property.

**Discussion:**

The primary motivation here is to make use of auto-completion feature of modern Java IDEs as much as possible. A new DSL has been implemented, which is, however, slightly different from what is proposed above.

1. Qos objects are accessible only via (a) QosProvider, (b) getDefault\*\*\*Qos, and (c) getQos. I.e., Qos objects are created using the factory pattern. They can’t be created out of thin air. Section 7.2.5.3 in the specification describes the QosProvider interface.
2. Qos objects have withPolicy and withPolicies methods (instead of just “with”). It serves as a reminder that only policy objects are allowed. Auto-completion provides no help here at all because the interface is quite generic. For example, DataReaderQos.withPolicies method has no idea if you want to pass Reliability or History or something else.
3. The Qos policies themselves are Java interfaces. As a consequence, their objects can’t be created. Therefore, a PolicyFactory class has been introduced. An instance of PolicyFactory can be obtained as follows.

PolicyFactory pf = anyDDSObject.getServiceEnvironment().getPolicyFactory();

1. PolicyFactory class provides many methods to create “default” DDS Qos policies. The “default” configuration of the policy objects is unspecified to allow vendor-specific optimized values (for instance, resource-limits). Portable code should specify all the relevant configuration parameters.
2. The Qos policy classes provide descriptive methods to change the policy setting. For example, the following code creates a Reliable reliability qos policy

pf.Reliability().withReliable();

The “with” prefix is used here because (1) it maintains consistency, (2) typing “w” “I” “t” “h” quickly reduces auto-completion options, and (3) it improves readability when used with a number of qos policies that are configurable with some integer or time values. For instance,

pf.ResourceLimits().withMaxSamples(P).withMaxInstances(Q);

1. Finally, the QosPolicy interface extends the raw java.lang.Comparable interface as opposed to each qos policy extending the Comparable<T> interface. This is because it allows a heterogeneous container of Comparable QosPolicies. Comparison of two “vectors” of qos policies can be trivially implemented in that case.

See revision #200: <https://code.google.com/p/datadistrib4j/source/detail?r=200>

These changes are also available in file diff\_omg\_issue\_16536.txtin ptc/2012-12-08 (issue\_diffs.zip)

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 16587

Title: API Should Avoid Side-Effects, e.g. Remove Bucket Accessors

Source:

PrismTech (Dr. Angelo Corsaro, PhD., angelo.corsaro(at)prismtech.com)

**Nature:** Revision

**Severity:**

Summary:

The DDS-PSM-Java provides bucket accessors that allow to "return" an object by "filling" a method parameter. As an example, for a property Foo there would be a method:

Foo f = // some foo

x.getFoo(f)

The rationale for this API is to avoid a defensive copy of Foo each time it is accessed.

However the cost of this "optimization" is an API that has side-effects everywhere, with all the nasty implications of side-effects.

Proposed Resolution:

The solution suggested to avoid bucket accessors and thus side-effects is to rely as much as possible on immutable objects (e.g. value-types). This ensures that (1) defensive copies are unnecessary since the attribute returned is immutable, and (2) new objects are created when new values are required.

If properly designed (as shown on an issue posted on QoS and Policies) this approach not only leads to a simpler and safer API, but it also leads to actually save memory in most of the cases.

The only case where the suggested approach has a cost is when a property changes very often. However, in many of these cases (often found in loops) the new JDK7 escape analysis will help greatly help in dealing with the potential garbage as it will allocate these short-lived objects on the stack.

Revised Text:

**This issue does not make any changes to the specification document.**

**Discussion:**

Most instances of the “bucket getter” pattern have been removed. For instance, DataWriter.getMatchedSubscriptions. DataReader.getMatchedPublications, etc. The pattern is only used in the performance critical methods (e.g., DomainParticipant.getCurrentTime). Please see revision 184:

<https://code.google.com/p/datadistrib4j/source/detail?r=184>

The source diffs are also available in ptc/2012-12-08 (issue\_diffs.zip)

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 17065

Title: Class for Query Expression

Source:

PrismTech (Dr. Angelo Corsaro, PhD., angelo.corsaro(at)prismtech.com)

**Nature:** Uncategorized Issue

**Severity:** Minor

Summary:

ContentFiltered topics, QueryCondition, and MultiTopic all require a "Query" parameter made by an expression and a set of parameters. The current API, however treats the expression and the parameter as individual parameters and does not provide any abstraction of what could represent a generic DDS query. This makes the API more verbose and more error prone.

Proposed Resolution:

Add a Query class that abstracts over the concept of a DDS query: an expression and a collection of mutable parameters

Revised Text:

A new paragraph has been added in Section 7.6.3 (DataReader interface) as follows: Quote[

* Instead of overloading several operation variants that accept large numbers of infrequently used parameters, a DataReader.Selector is provided to encapsulate various selection criteria (for example, sets of sample, instance, and view states). DataReader.select method returns a Selector object, which encapsulates the *default* selection criteria. For portability, the default state of the Selector object is defined as instanceHandle=null, nextInstance=false, dataState=any, queryExpression=null, and maxSamples=unlimited. Selector provides fluent interface to modify the default selection parameters. For convenience, Selector provides read and take methods.

]Unquote.

**Discussion:**

The concerns raised by the issue are valid but the proposed resolution (adding a Query class) only partially addresses the concerns. A more general abstraction, namely “Selector” has been introduced to capture a generic DDS query. Section 7.6.3 in the specification describes the Selector concept in detail. A summary is presented here.

A selector object is obtained from a datareader. The Selector object provides a fluent interface to specify query parameters, such as instance handles, data states, etc. Finally, the read method executes the query encapsulated in the selector object.

For instance

DataReader<Track> dr = …  
dr.select().instance(someHandle)  
 .dataState(someState)  
 .maxSamples(500)  
 .read();

Alternatively,

DataReader<Track> dr = …  
dr.read(dr.select().instance(someHandle)  
 .dataState(someState)  
 .maxSamples(500));

See revision #198: <https://code.google.com/p/datadistrib4j/source/detail?r=198>

These changes are also available in file diff\_omg\_issue\_17065.txt in ptc/2012-12-08 (issue\_diffs.zip).

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 17204

Title: Obsolete EntityQos interface name

Source:

DECA (Mr. Rick Warren)

**Nature:** Uncategorized Issue

**Severity:**

Summary:

The base interface for all Entity-level QoS objects (e.g. DataReaderQos) is org.omg.dds.core.EntityQos. At one time during the evolution of the specification, this interface was called org.omg.dds.core.Qos. Due to an editorial oversight, this obsolete name persists in the specification document and should be updated.

\* Section 7.2.5, "QoS and QoS Policies"

\* Section 7.2.5.2, "Entity QoS"

Proposed Resolution:

Update the specification document.

Revised Text:

The specification document has been updated as follows

Section 7.2.5 QoS and QoS Policies

QoS-related types fall into two categories, as expressed in the DDS PIM: individual QoS policies (such as reliability) and the collections of policies that apply to a particular DDS Entity type. This PSM represents the former with the base interface org.omg.dds.core.policy.QosPolicy and the latter with the base interface org.omg.dds.core.EntityQos.

Section 7.2.5.2 Entity QoS

Each Entity QoS (e.g., DataReaderQos) is an interface extending org.omg.dds.core.EntityQos.

There are no issue\_diffs for this issue.

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 17302

Title: Implement Java5 Closeable interface

Source:

DECA (Mr. Rick Warren)

**Nature:** Uncategorized Issue

**Severity:**

Summary:

DDS code will be easier to integrate into third-party I/O code if the Entity, ReadCondition, and TopicDescription interfaces implement the java.util.Closeable interface. This is especially true under Java 7, which provides specific new language constructs for dealing with this interface.

The only method in the interface is a no-argument close(), which all of these interfaces already have.

Proposed Resolution:

Update Entity, ReadCondition, and TopicDescription to inherit from java.io.Closeable.

Revised Text:

Section 7.6.3 describes the use of java.io.Closeable interface as follows:

* Applications that read or take loans must eventually return those loans; this PSM maps the return\_loan operation from the DDS PIM to an operation returnLoan on the Sample.Iterator. Moreover, the iterator implements the Java.io.Closeable interface so that try-with-resources construct can be used in Java 7,

**Discussion:**

See revision #186: <https://code.google.com/p/datadistrib4j/source/detail?r=186>

These changes are also available in file diff\_omg\_issue\_17302.txt in ptc/2012-12-08 (issue\_diffs.zip)

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 17303

**Title:** **Update specification for final DDS-XTypes**

Source:

DECA (Mr. Rick Warren)

Summary:

The second FTF of the DDS-XTypes spec introduced several API changes that should be incorporated into the DDS-PSM-Java spec.

At the same time, the contents of the relevant portions of the DDS-XTypes spec should be incorporated as JavaDoc comments, just as has already been done for DDS itself.

Revised Text:

Section 9 in the specification now describes the new type mapping with respect to the final DDS-XTypes specification. Section 9.1 in the specification has been updated as follows.

Quote [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 Java 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-Java mapping [Java-MAP]).] Unquote

**Discussion:**

The DDS Java PSM API has been revised to reflect the latest revision of XTypes specification. The details can be seen here

Change-set for Built-in types: <https://code.google.com/p/datadistrib4j/source/detail?r=207>

Change-set for dynamic type: <https://code.google.com/p/datadistrib4j/source/detail?r=206>

The change-sets are also available in diff\_omg\_issue\_17303.txt in ptc/2012-12-08 (issue\_diffs.zip)

**Proposed Disposition:** Resolved

**Disposition:** Resolved

OMG Issue No: 17304

Title: Improve compile-time type safety of EntityQos

Source:

DECA (Mr. Rick Warren)

**Nature:** Uncategorized Issue

**Severity:**

Summary:

The EntityQos interface implements java.util.Map. However, all checking of which policies apply to which Entity types is deferred to run time. The extension of Map should be updated to constrain which policies may legally be used.

Proposed Resolution:

Introduce marker interfaces for each Entity type and parameterize Map with these interfaces.

Revised Text:

This issue requires no changes in the specification document.

**Discussion:**

See revision #185: <https://code.google.com/p/datadistrib4j/source/detail?r=185>

These changes are also available in file diff\_omg\_issue\_17304.txt in ptc/2012-12-08 (issue\_diffs.zip)

Disposition: Resolved

OMG Issue No: 17415

Title: Implement java.io.Closeable in Sample.Iterator

Source:

Real-Time Innovations (Mr. Sumant Tambe, sumant(at)rti.com)

**Nature:** Clarification

**Severity:** Minor

Summary:

Java 7 has a try-with-resources construct that allows a close() method to be called automatically when a certain code block ends. Java PSM can support this construct with sample loans in a way that's backwards compatible with Java 5. All we have to do is to rename the Sample.Iterator.returnLoan() method to close() and make Sample.Iterator implement the interface java.io.Closeable. Note: java.lang.AutoCloseable is available only since 1.7

Proposed Resolution:

Inherit Sample.Iterator from Java.io.Closeable and rename the Sample.Iterator.returnLoan() method to close()

Revised Text:

Section 7.6.3 describes the role of java.io.Closeable. Specifically, the following sentence was added: quote [Moreover, the iterator implements the Java.io.Closeable interface so that try-with-resources construct can be used in Java 7 ]unquote.

**Discussion:**

The DataReader.returnLoan method has been renamed as DataReader.close.

See revision #192: <https://code.google.com/p/datadistrib4j/source/detail?r=192>

These changes are also available in file diff\_omg\_issue\_17415.txt in ptc/2012-12-08 (issue\_diffs.zip)

Proposed Disposition: Resolved

Disposition: Resolved

OMG Issue No: 18285

Title: Redundant "QoSPolicy" Suffix on Policy Types.

Source:

Sumant Tambe (sumant@rti.com)

**Nature:** Clarification

**Severity:** Significant

Summary:

The dds-psm-java uses a superfluous "QoSPolicy" suffix to name the DDS policies which themselves are already included in a "policy" namespace.

This issue is identical to issue #16530. This issue is created to revote on the decision taken on issue #16530 in the earlier FTF.

Proposed Resolution:

This suffix should be removed. This resolution will also make Java PSM consistent with the C++ PSM, which does not use "QosPolicy" suffix.

Revised Text:

This issue requires no changes to the specification document.

**Discussion:**

Removed “QosPolicy” suffix.

See revision #191: <https://code.google.com/p/datadistrib4j/source/detail?r=191>. These changes are also available in file diff\_omg\_issue\_16530.txt in ptc/2012-12-08 (issue\_diffs.zip)

**Proposed Disposition:** Resolved

**Disposition:** Resolved

Disposition: Deferred

Disposition: Closed, no change

Disposition: Duplicate/merged