Report of the Java 5 Language PSM for DDS Finalization Task Force 1.0 to the OMG Platform Technical Committee 14 November 2011

Document Number:	**ptc/2011-10-05**
Task Force Chair:	**Rick Warren (RTI)**

Specification

Revised specification (clean):	**ptc/2011-10-07**
Revised specification (change-bar):	**ptc/2011-10-06**

Accompanying documents

Inventory:	**ptc/2011-10-10**	**Non-normative**
omgdds.jar:	**ptc/2011-10-09**	**Normative**
omgdds_src.zip:	**ptc/2011-10-08**	**Normative**
issue_diffs.zip	**ptc/2011-11-02**	**Normative**

Template: omg/09-06-01

Table of Contents

Summary of DDS-PSM-Java FTF Activities

Formation

- Chartered By: Platform TC
- On: 10 December, 2010; Santa Clara, CA
- Comments Due Date: 29 August, 2011
- Report Due Date: 14 November, 2011

Revision / Finalization Task Force Membership

Member	Organization	Status
Angelo Corsaro	PrismTech	Charter
Fabrizio Morciano	Selex-SI	Charter
Ken Rode	Gallium	Added
Rick Warren	Real-Time Innovations (RTI)	Charter (chair)
Virginie Watine	Thales	Charter

Issue Disposition:

Disposition	Number of Occurrences	Meaning of Disposition
Resolved	15	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	7	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	8	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	7	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	12 October 2011	16104, 16316, 16317, 16318, 16319, 16320, 16322, 16324, 16325, 16327, 16328, 16541 *Outcome: Issues 16316 and 16325 did not pass; resolutions will have to be updated. All other issues passed.*
2	7 November 2011	15966, 15968, 16050, 16056, 16316, 16321, 16323, 16325, 16326, 16369, 16529, 16530, 16531, 16532, 16533, 16534, 16535, 16536, 16537, 16538, 16539, 16540, 16542, 16543, 16587, 16588, 16589 *Outcome: Issue 16531 did not pass; the resolution will have to be updated. All other issues passed.*
3	14 November 2011	16531 (to defer)

Voter	Vote in poll 1	Vote in poll 2	Vote in poll 3
Angelo Corsaro	Yes on 16104, 16317, 16318, 16319, 16320, 16322, 16324, and 16327. No on 16316, 16325, and 16328. Did not vote on 16541.	Yes on 15966, 15968, 16050, 16056, 16316, 16321, 16325, 16369, 16529, 16535, 16536, 16538, 16540, 16542, and 16587. No on 16323, 16326, 16530, 16531, 16532, 16533, 16534, 16537, 16539, 16543, 16588, and 16589.	Yes on one issue.
Fabrizio Morciano	Did not vote.	Yes on all issues.	Yes on one issue.
Ken Rode	Yes on all issues.	Abstain on 16531. Yes on all other issues.	Yes on one issue.

Rick Warren	Yes on all issues.	Yes on all issues.	Yes on one issue.
Virginie Watine	Yes on 16104, 16317, 16318, 16319, 16320, 16322, 16324, 16327, 16328, and 16541. No on 16316 and 16325.	Yes on 15966, 15968, 16050, 16056, 16316, 16321, 16325, 16369, 16529, 16534, 16535, 16536, 16538, 16540, 16542, 16587, and 16589. No on 16323, 16326, 16530, 16531, 16532, 16533, 16537, 16539, and 16543. Did not vote on 16588.	Yes on one issue.

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	2	16533, 16534
Significant - Fixed problems with normative parts of the specification that raised concern about implementability	6	16050, 16104, 16322, 16326, 16531, 16538

Minor - Fixed minor problems with normative parts of the specification	29	15966, 15968, 16056, 16316, 16317, 16318, 16319, 16320, 16321, 16323, 16324, 16325, 16327, 16328, 16369, 16529, 16530, 16532, 16535, 16536, 16537, 16539, 16540, 16541, 16542, 16543, 16587, 16588, 16589
Support Text -Changes to descriptive, explanatory, or supporting material.	0	*None*

Disposition: Resolved

OMG Issue No: 16050

Title: duplicate `put` definition resulting in a name clash

Source:

Thales (André Bonhof, andre.bonhof@nl.thalesgroup.com)

Nature: Clarification

Severity: Significant

Summary:

The `ModifiableEntityQos` contains `put()` definition that, after type erasure, cannot be distinguished from the inherited put definition in `EntityQos` (or the one inherited from `Map`) resulting in duplicate definitions of put:

```
QosPolicy<?,?> put(QosPolicy.Id, QosPolicy<?,?>)
```

These duplicate definitions result in a compilation error.

Discussion:

[André] Possible ways to resolve this:

1. Drop the "`extends Map`" in `EntityQos` and put a dedicated `get()` in `EntityQos` and a dedicated `put()`/`set()` in `ModifiableEntityQos` and leave it up to the implementation on how to manage these values. This is the preferred solution as it prevents the user of the API to accidently use the `Map` inherited modification methods like put/remove/clear on a non-modifiable `EntityQos`.
2. Modify the signature of `put()` in `ModifiableEntityQos` to match the inherited definitions in `EntityQos` and Map:

```
public QosPolicy<?,?> put(QosPolicy.Id key, QosPolicy<?,?>
value);
```

[Rick] I think the Map extension provides a useful way to navigate QoS objects in a generic way. Therefore, I prefer the second approach.

See also issue #16369, which will potentially impact the same method signature. See also issue #16537, which proposes eliminating the `EntityQos` interface altogether.

See the following revisions, which contain provisional fixes for this issue:

- Revision #116: http://code.google.com/p/datadistrib4j/source/detail?r=116, which updates the method signature as described in (2) above and also addresses issue #16369.

- Revision #136: http://code.google.com/p/datadistrib4j/source/detail?r=136, which further refines the method signature to address a compilation error introduced by Java 7 compliance.

Resolution:

See the modifications described below.

Revised Text:

See revision #161, which rolls up the revisions above and also resolves issue #16369: http://code.google.com/p/datadistrib4j/source/detail?r=161. These changes are also available in the attached file diff_omg_issue_16050_16369.txt.

Disposition: **Resolved**

OMG Issue No: 16104

Title: Missing *behavioral* descriptions of the interface

Source:

Thales (André Bonhof, andre.bonhof@nl.thalesgroup.com)

Nature: Clarification

Severity: Significant

Summary:

Some parts of the interface (JavaDoc) are poorly documented especially with respect to behavior. This Java documentation will be the key documentation for the *new* DDS application programmers. It may be trivial or implicit for the ones writing the standard but it will not be for the application programmers which are not familiar with the existing DDS standard will use it

For example have a look at the method `createDataWriter(Topic<TYPE> topic)` on the `Publisher` interface. What will happen if the middleware cannot create the `DataWriter`? Will an unchecked exception be thrown or is a `null` value returned or even worse the `DataWriter` is simply returned and will fail when the first write action is performed?

I now that the existing OpenSplice DDS implementation will return null when the middleware is not able to create the `DataWriter` but it would be nice that applications are not only portable from interface compliance aspect but also from behavioral aspect (and that application programmers are aware of it)!

Discussion:

[Rick] The behavioral specifications already exist—in the appropriate specification documents: DDS, DDS-XTypes, and DDS-PSM-Java. So I think this issue is not about portability, but really about ease of use: it is more convenient to programmers if more of the relevant specifications are available copied into the JavaDoc.

Resolution:

Merge the descriptions of classes and operations from the DDS specification into the appropriate JavaDoc comments. This PSM does not introduce new concepts, so no merge is necessary in that case. DDS-XTypes is in finalization, so its contents are not yet fixed. Therefore, to avoid the possibility of errors and inconsistencies, we should put it aside for now.

Revised Text:

The contents of the DDS specification have been merged into JavaDoc comments in revision #140: http://code.google.com/p/datadistrib4j/source/detail?r=140. The difference is also available in the attached file diff_omg_issue_16104.txt.

Disposition: **Resolved**

OMG Issue No: 16317

Title: Update specification to reflect DDS-XTypes FTF1 issue resolutions

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The (first) DDS-XTypes FTF has completed. Some of the issue resolutions result in API changes that impact this specification. Those changes should be reflected here to keep this specification aligned with the PIM.

These issues potentially include:

- #15689, Identifiers `TypeId` and `Module` collide with IDL keywords
- #15691, Unclear member names when programming language doesn't support `typedef`
- #15693, Extensibility kinds of new QoS policies are not specified in a consistent way
- #15696, Incorrect `FooDataWriter` overloads for built-in types
- #15706, Reduce size of `DynamicData` API

Note that this issue applies to DDS-PSM-Cxx too.

Resolution:

See the following revisions:

- Revision #128: http://code.google.com/p/datadistrib4j/source/detail?r=128. Reflects DDS-XTypes issue #15696—overloads in writers of built-in types.
- Revision #129: http://code.google.com/p/datadistrib4j/source/detail?r=129. Reflects DDS-XTypes issue #15691—clarity of member names in the absence of `typedef`.
- Revision #130: http://code.google.com/p/datadistrib4j/source/detail?r=130. Reflects DDS-XTypes issue #15706—simplified `DynamicData`.

Other aspects of issue #15689 and 15691 were already addressed in the drafting of DDS-PSM-Java. Issue #15693 is a no-op because of the way inheritance and annotations are used.

Revised Text:

The above revisions have been rolled up in revision #148: http://code.google.com/p/datadistrib4j/source/detail?r=148. The changes are also available in the attached file diff_omg_issue_16317.txt.

Disposition: **Resolved**

OMG Issue No: 16318

Title: `Entity.setListener` is missing listener mask

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The method signature for `Entity.setListener` does not include the listener "mask" (actually, a collection of status classes in this PSM) parameter from the PIM.

Discussion:

The current signature is useful as a convenience for the common case where the application wants all callbacks. But it lacks the expressiveness of the PIM, so an additional overload should be provided.

Resolution:

Add the following method to the Entity interface:

```
public void setListener(
            LISTENER listener,
            Collection<Class<? extends Status<?, ?>>> statuses);
```

Include the appropriate JavaDoc copied from the DDS specification.

Revised Text:

See revision #141: http://code.google.com/p/datadistrib4j/source/detail?r=141.
The changes are also available in the attached file diff_omg_issue_16318.txt.

Disposition: **Resolved**

OMG Issue No: 16319

Title: Unclear blocking behavior for `WaitSet.waitForConditions` overloads that don't specify timeout

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The method `WaitSet.waitForConditions` is provided with several overloads, including some that do not take an explicit timeout. These are intended to wait indefinitely. However, they still throw `TimeoutException`. How can they time out if they wait forever?

Discussion:

`Object.wait` allows indefinite waiting, so it makes sense for this specification to allow it as well. However, these overloads should not ever throw `TimeoutException`.

Resolution:

Remove the clause "throws `TimeoutException`" from these method declarations.

Revised Text:

See revision #142: http://code.google.com/p/datadistrib4j/source/detail?r=142. These changes are also available in the attached file diff_omg_issue_16319.txt

Disposition: **Resolved**

OMG Issue No: 16320

Title: Incorrect topic type specification in `DomainParticipant.createMultiTopic`

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The method `DomainParticipant.createMultiTopic` specifies the type of the resulting object using a registered type name in string form. However, this is inconsistent with the way type registration is handled elsewhere in this PSM: callers provide a `Class` or `TypeSupport` object, and the implementation registers the type implicitly as necessary.

Discussion:

We should follow the model of `createTopic`: provide two overloads, one taking a `Class` and the other a `TypeSupport`.

Resolution:

Replace the existing `createMultiTopic` method declaration with two new overloads. In place of the `typeName` string, the first new overload shall take a `Class` parameter. The second shall take a `TypeSupport` parameter.

Revised Text:

See revision #143: http://code.google.com/p/datadistrib4j/source/detail?r=143. These changes are also available in the attached file diff_omg_issue_16320.txt.

Disposition: **Resolved**

OMG Issue No: 16321

Title: Too many `read`/`take` overloads

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The `DataReader` interface defines a very large number of `read` and `take` variants. While each one has a clear meaning, the sheer number of them makes the API harder to understand.

Discussion:

[Rick] One possibility would be to follow the example of the C++ PSM, and combine things like condition, handle, etc. into a "filter" parameter.

[Angelo] The only overload for the `createReadCondition` should be accepting a parameter that is the same as the `DataReader.read` e.g. a `Query`. This way the API will be more consistent.

See the following revisions, which constitute a provisional resolution to this issue:

- Revision #131: http://code.google.com/p/datadistrib4j/source/detail?r=131, which collapses view states, instance states, sample states, and other parameters into helper types `Subscriber.ReaderState` and `DataReader.Query`.
- Revision #135: http://code.google.com/p/datadistrib4j/source/detail?r=135, which eliminates mandatory memory allocations in some overloads.
- Revision #152: http://code.google.com/p/datadistrib4j/source/detail?r=152, which makes the signatures of `createReadCondition` and `createQueryCondition` consistent with the new signatures of `read` and `take`.

Resolution:

See below.

Revised Text:

With respect to the code, see revision #162: http://code.google.com/p/datadistrib4j/source/detail?r=162, which roles up the revisions above. These changes are also available in the attached file diff_omg_issue_16321.txt.

With respect to the specification document, delete the last three bullets at the end of section 7.6.3, "`DataReader` Interface", which begin with "Operations accepting `ReadCondition`s…" and end with "to simply `read`/`takeNext`, transforming these operations into overloads of one another."

In their place, put the following:

- Qualifications to the data to be read or taken, including the number of samples, a `ReadCondition`, a particular instance, and so on, have been encapsulated in a nested type `DataReader.Query`. This refactoring allows a large number of distinct methods from the PIM, each qualified by a different name suffix, to be collapsed to a very small number of overloads.

Disposition: **Resolved**

OMG Issue No: 16323

Title: Logically ordered types should implement `java.lang.Comparable`

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

Several of the types defined in this PSM have a natural order (such as `Time`). In order to better integrate with the Java platform, these types should implement the standard `java.lang.Comparable` interface.

Discussion:

See the following revisions, which constitute a provisional resolution to this issue:

- Revision #122: http://code.google.com/p/datadistrib4j/source/detail?r=122. This update encompasses `Bit`, `Duration`, and `Time`.
- Revision #134: http://code.google.com/p/datadistrib4j/source/detail?r=134. This update encompasses `InstanceHandle`.

Resolution:

Implement `Comparable` in the following types:

- `Bit`—ordered based on index within a bit set
- `Duration`—ordered from shorter durations to longer ones
- `InstanceHandle`—ordered in an implementation-specific way (DDS specification of `DataReader::read()` requires such an ordering)
- `Time`—ordered from earlier points in time to later ones

Revised Text:

See revision #163: http://code.google.com/p/datadistrib4j/source/detail?r=163, which rolls up the above revisions. These changes are also available in the attached file diff_omg_issue_16323.txt.

Disposition: **Resolved**

OMG Issue No: 16324

Title: Improve polymorphic sample creation

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The specification does not provide a simple, portable way to create a new data sample to use with the middleware. Instead, there are several partial solutions:

- Instantiate a concrete sample type directly: "new Foo()". This approach doesn't work in generic methods—i.e. when the concrete type is not statically known. It also doesn't work with `DynamicData`.
- Instantiate `DynamicData` from `DynamicDataFactory`. But samples of statically known, user-defined types don't have a "data factory".
- Use `DataReader.createData()`. But there is not equivalent on the publishing side.

There should be a single way that works uniformly and generically.

Resolution:

The proposed resolution has several parts:

1. Introduce a new factory instance method to the `TypeSupport` class: `TypeSupport.newData()`. The name of this method is parallel to that of other value type "constructor-like" factories.
2. Support navigation from the `TopicDescription` to the `TypeSupport`. Add a new method `TopicDescription.getTypeSupport()`.
3. Simplify the number of ways to get from the data type's `TypeSupport` to its `Class`. *Add* a method `TypeSupport.getType()`. *Remove* the existing methods `TopicDescription.getType()`, `DataWriter.getType()`, and `DataReader.getType()`: they are redundant.
4. Remove the existing method `DataReader.createData()` and the existing class `DynamicDataFactory`. They are not needed. In the specification document, rename section 7.7.1.1, "`DynamicTypeFactory` and `DynamicDataFactory` Interfaces", to "`DynamicTypeFactory` Interface". In the single paragraph of that section, make the word "factories" singular.

See revision #123, which includes the above changes:
http://code.google.com/p/datadistrib4j/source/detail?r=123.

5. Remove the factory methods on the built-in topic data classes. Objects of these types can be constructed like those of any other sample type. *See also revision #137, which includes this change: http://code.google.com/p/datadistrib4j/source/detail?r=137.*

There will therefore be a single polymorphic and generic way to instantiate a sample of any type: by using its `TypeSupport`. You can get the `TypeSupport` from any related `TopicDescription`, or transitively any `DataReader` or `DataWriter`.

Likewise, there will be a single polymorphic and generic way to get the `Class` object for any data type: from its `TypeSupport`. As described in the previous paragraph, you can get to the `TypeSupport` from a variety of places.

Revised Text:

See revision #144, which rolls up the aforementioned changes: http://code.google.com/p/datadistrib4j/source/detail?r=144. These changes are also available in the attached file diff_omg_issue_16324.txt.

Disposition: **Resolved**

OMG Issue No: 16326

Title: `copyFromTopicQos` signatures are not correct

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Significant

Summary:

The specification currently provides the following APIs:

```
void Publisher.copyFromTopicQos(DataWriterQos dst, TopicQos src);
void Subscriber.copyFromTopicQos(DataReaderQos dst, TopicQos src);
```

There are two problems with these methods. The first is an issue of correctness; the second is an issue of usability.

(1) The methods are supposed to modify the writer or reader QoS that are passed in. However, those objects may not be modifiable. The types of the first parameters should be `ModifiableDataWriterQos` and `ModifiableDataReaderQos` respectively.

(2) The signatures are not consistent with the "bucket" getters in the PSM, which accept an "in-out" container to fill in and then return that same object to facilitate method call chaining. If I want to use one of these methods to create an endpoint, I have to do something like the following:

```
DataWriterQos dwq = pub.getDefaultDataWriterQos().modify();
pub.copyFromTopicQos(dwq, topic.getQos());
DataWriter dw = pub.createDataWriter(…, dwq, …);
```

But if the `copyFromTopicQos` methods simply returned the value of their `dst` arguments, I could avoid the intermediate `dwq` variable:

```
DataWriter dw = pub.createDataWriter(
   …,
   pub.copyFromTopicQos(pub.getDefaultDataWriterQos().modify(),
                        topic.getQos()),
   …);
```

Discussion:

See revision #126: http://code.google.com/p/datadistrib4j/source/detail?r=126, which constitutes a provisional resolution to this issue.

Resolution:

Change the signatures as follows:

In `Publisher.java`:

```
-    public void copyFromTopicQos(DataWriterQos dst, TopicQos src);
+    public ModifiableDataWriterQos copyFromTopicQos(
+            ModifiableDataWriterQos dst, TopicQos src);
```

In `Subscriber.java`:

```
-    public void copyFromTopicQos(DataReaderQos dst, TopicQos src);
+    public ModifiableDataReaderQos copyFromTopicQos(
+            ModifiableDataReaderQos dst, TopicQos src);
```

Revised Text:

See revision #165: http://code.google.com/p/datadistrib4j/source/detail?r=165.

Disposition: **Resolved**

OMG Issue No: 16327

Title: Parent accessors should be uniform across Entities and Conditions

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

All `DomainEntity` interfaces, and some `Condition` interfaces, can provide a reference to the parent object. In the case of Entities, this accessor has been captured in the form of a generic base interface method:

```
PARENT DomainEntity.getParent();
```

However, `StatusCondition` and `ReadCondition` are not parallel. They provide the following methods:

```
ENTITY StatusCondition.getEntity();
DataReader<TYPE> ReadCondition.getDataReader();
```

It would be more consistent if we renamed both of the above methods to `getParent`.

Resolution:

Rename `StatusCondition.getEntity` to `getParent`.

Rename `ReadCondition.getDataReader` to `getParent`.

Revised Text:

See revision #145: http://code.google.com/p/datadistrib4j/source/detail?r=145. These changes are also available in the attached file diff_omg_issue_16327.txt.

In the specification document in section 7.2.7.3, “Conditions”, replace the method name “`getEntity`” with “`getParent`”.

Disposition: **Resolved**

OMG Issue No: 16328

Title: `DataReader.createReadCondition()` is useless

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The `DataReader` interface provides two overloads for the `createDataReader` method: one that takes no arguments and another that takes the appropriate sample states, view states, and instance states. The existence of the first overload supposes that it will be common to create a `ReadCondition` with any sample state, any view state, and any instance state. But in fact, such a `ReadCondition` is not very useful at all: there's no point in passing it to `read`/`take`, because it will not filter the available samples in any way. And although you could use it with a `WaitSet`, it doesn't do anything that you couldn't do with a `StatusCondition` on the `DATA_AVAILABLE` status.

Resolution:

Remove the no-argument overload of `DataReader.createReadCondition`. Leave the three-argument overload unchanged.

Revised Text:

See revision #147: http://code.google.com/p/datadistrib4j/source/detail?r=147. These changes are also available in the attached file diff_omg_issue_16328.txt.

Disposition: **Resolved**

OMG Issue No: 16369

Title: `QosPolicy.Id` enumeration is redundant

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

In the DDS PIM, each QoS policy has a name and an ID that uniquely identify it. In this PSM, these two things are encapsulated in the enumeration `QosPolicy.Id`. But the Java platform already provides equivalent information: the `Class` object. The ability to quickly compare `Class` object pointers is equivalent to comparing ID integer values, and each `Class` already has a name string.

Discussion:

See the following revisions, which contain a provisional resolution to this issue:

- Revision #116: http://code.google.com/p/datadistrib4j/source/detail?r=116
- Revision #121: http://code.google.com/p/datadistrib4j/source/detail?r=121

Resolution:

Remove the enumeration `QosPolicy.Id`. Replace its uses with `Class<? extends QosPolicy>`.

Revised Text:

See revision #161, which rolls up the revisions above and also resolves issue #16050: http://code.google.com/p/datadistrib4j/source/detail?r=161. These changes are also available in the attached file diff_omg_issue_16050_16369.txt.

Disposition: **Resolved**

OMG Issue No: 16532

Title: RxO QoS Policies should be Comparable (idem for QoS)

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Critical~~ Minor

This issue was marked "Critical" when filed but does not impact the implementability of the specification.

Summary:

Some of the DDS QoS Policies are Request vs. Offered in the sense that the value of matching policies on communicating entities have to satisfy a specific ordering relationship. Specifically, the policy set on the receiving side should always be less or equal than the analogous QoS Policy on the sending side. As a result there is a natural total ordering for RxO Policies, which is not currently captured nor reflected in the API.

As a consequence also QoS should be defining a total order.

Discussion:

[Angelo] Proposal: Ensure that all RxO Policies, and all entity QoS types, implement the `Comparable` interface.

[Rick] There is no total order over QoS types, because they consist of multiple QoS policies, which may not all share the same relative ordering.

Re: only QoS policies then, it is true that RxO contracts define an ordering of policies. But saying that a writer with a certain policy offers a "greater" level of service than a reader with another policy is a very different sort of comparison than saying that one integer is "greater" than another. Therefore, I am not completely comfortable with this issue, although I am interested to hear what others have to say.

Therefore, I propose that, rather than *implementing* the `Comparable` interface, these policy types should *provide* a `Comparable` object. This approach is functionally identical but clarifies which comparison contract is being used. For example:

```
if (dwPolicy.requestedOfferedContract().compare(drPolicy) >= 0) {
    System.out.println("Compatible!");
}
```

See revision #155: http://code.google.com/p/datadistrib4j/source/detail?r=155, which contains an initial provisional expression of this resolution. See also

revision #167: http://code.google.com/p/datadistrib4j/source/detail?r=167, which adds support for DDS-XTypes-derived policies.

Resolution:

Define an interface `RequestedOffered`; all QoS policy types that require requested/offered compatibility extend this interface. The interface has one method, `requestedOfferedContract`, which returns a `Comparable` object. That objects compares the policy against another instance of the policy based on the requested/offered compatibility rule(s) for that policy type.

Revised Text:

See revision #168: http://code.google.com/p/datadistrib4j/source/detail?r=168. These changes are also available in the attached file diff_omg_issue_16532.txt.

Disposition: **Resolved**

OMG Issue No: 16541

Title: A Status is not an Event. An Event is not a Status, it notifies a status change.

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked "Major" when filed, but there is no such recognized severity. This issue does not impact the implementability of the specification.

Summary:

The `org.omg.dds.core.status.Status` class currently extends the `java.util.EventObject`.

The issue I have with this is that a status and an event are to different concepts.

A status represents a continuous value or set of values that are always defined, while an event represents and happening. For instance an event could be used to notify the change of status but not the status itself.

Proposed Resolution:

That said the refactoring suggested is to re-organize the current status types so to clearly distinguish what is are statuses and what are the events. As such, all the status currently defined should remove reference to the source. Why? Because the statuses are retrieved from the source thus it is kind of silly to add back the source on the communication status.

Let me give you an example ("`dr`" below is a `DataReader`):

```
   RequestedDeadlineMissedStatus s =
dr.getRequestedDeadlineMissedStatus();
    // this give back the reader we already know, thus it is not
real useful
    // information which should simply be removed.
    s.source())
```

BTW the status types as well as the relative accessor methods should drop the trailing "Status" as it is not so informative.

That said, we should add an event type associated to each status defined like this:

```
    class RequestedDeadlineMissedEvent {
        private RequestedDeadlineMissed status;
        private DataReader source;
```

```
        //... useful methods
    }
```

The event type is the one that should be used as a parameter of the listener methods.

Finally, it is worth noticing that the suggested refactoring will fix the `DataAvailableStatus` anomaly. This type, currently defined as a status, is actually an event and as such should be treated. So where is the anomaly, for this status there are no methods on the data reader and there is really no status information such as saying... Yes there are 15 new samples or something like this.

Revised Text:

See revision #149: http://code.google.com/p/datadistrib4j/source/detail?r=149. These changes are also available in the attached file diff_omg_issue_16541.txt.

Disposition: **Resolved**

Disposition: Deferred

OMG Issue No: 15966

Title: XML-Based QoS Policy Settings (DDS-PSM-Cxx/DDS-PSM-Java)

Source:

PrismTech (Angelo Corsaro, angelo.corsaro@prismtech.com)

Severity: Minor

Summary:

The newly introduced XML Based Policy configuration adds new methods in the core DDS entities that allow fetching 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.

Discussion:

[Angelo] 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 easily extending the supported format for QoS representation without any impact on the core DDS API and overall facilitate the support for multiple approaches.

[Rick] One approach discussed in the Orlando meeting is to provide such a Builder API. A builder would be instantiated in one of two ways:

1. From a profile name, as is described in the issue report above. *The concern was raised that this approach makes administration more difficult for application developers, because it forces them to make additional method calls, and more difficult for administrators, because it raises the possibility that the improper QoS could be used for a given entity.*
2. From an existing QoS object. *This would essentially amount to a refactoring of the existing* `Modifiable*Qos` *and* `Modifiable*QosPolicy` *interfaces. Therefore, the resolution to this issue is related to that of #16529.*

Once a Builder is created, it would allow the QoS values it holds to be modified. Therefore, this issue overlaps with issue #16536, which also calls for a new way to modify QoS policies.

[Angelo] I do not agree with the text as written on point (1). To remove the extra call it would be sufficient to have a method with an abstract policy builder. To

avoid misleading QoS assignment it would be sufficient to address the issue raised on the DDS API and ensure that DDS entities can have names. That way the DDS entity could look up its QoS by name.

Resolution:

Defer this issue because of the dependencies on issues #16529 and #16536. Both of those issues have broader impacts than this one. Unfortunately, those issues were filed outside the FTF comment deadline, and there is insufficient time to address them in this FTF.

Disposition: **Deferred**

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)

Severity: Minor

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.

Discussion:

[Rick] 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. I believe that's the right model to follow in DDS-PSM-Java as well.

The FTF membership discussed this issue at the Orlando meeting and agreed to the following principles:

1. This issue overlaps the existing scope of the specification (i.e. the existing Java Type Representation) sufficiently that the issue should be accepted and resolved.
2. The Plain Language Binding for Java would map type members to Java Bean-style property accessors and mutators. This pattern is familiar to Java programmers, understandable to type designers, and consistent with the approach taken in the DDS-PSM-Cxx spec and the forthcoming IDL-to-C++11 specifications.
3. Ideally, the application of the Java Type Representation and the Plain Language Binding should be idempotent. In other words, if a type designer (*a*) starts with a Java class, (*b*) infers from it a type in the DDS type system (according to the Java Type Representation), and then (*c*) maps that abstract type back to a Java class (according to the Plain Language Binding), the result should be the same as the type he started with in (*a*) (or very close to it).

The implication of the above principles is that unfortunately the Java Type Representation will need to change significantly. Since this realization comes late in the process, and we intend to charter a second FTF anyway (because of the schedule of DDS-XTypes, on which this specification depends), it is prudent to defer this issue until it can be resolved carefully and thoroughly.

Resolution:

Defer this issue.

Disposition: **Deferred**

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)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

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).

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

[Rick] Situational analysis:

- The biggest occurrence of the modifiable/unmodifiable pattern is in the QoS policies and Entity QoS.
- `ModifiableDuration` can easily go away. `Duration` is only returned from QoS policy property accessors; QoS policies are not performance-sensitive. And in every case where durations are passed as arguments, there are already overloads to use an integer and a `TimeUnit`.
- `ModifiableTime` is used in two places: `DomainParticipant.getCurrentTime` and `Sample.getSourceTimestamp`. Both are performance-sensitive, although the latter could potentially be replaced by simply `Time`. Time is accepted as an argument in a number of `DataWriter` methods, though these can be easily eliminated: each already has an overload that accepts an integer and a `TimeUnit`.
- `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.
- `AnnotationDescriptor` and `MemberDescriptor` from the Dynamic Type API are provided in modifiable and unmodifiable versions. This API is not performance-sensitive, so accessors could simply return new copies of modifiable types.

Orlando meeting discussion:

- Consider replacing this pattern with a more explicit Builder pattern (see issue #15966) and/or a DSL (see issue #16536) in the case of Entity QoS and QoS policies.

- Eliminate `ModifiableDuration` and leave `Duration` as an immutable type. Eliminate method overloads that accept `Duration` as an argument, leaving in place those that accept an integer and a `TimeUnit`.
- Implement 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.*
- Remove `AnnotationDescription`, renaming `ModifiableAnnotationDescriptor` to `AnnotationDescriptor`. Remove `MemberDescription`, renaming `ModifiableMemberDescriptor` to `MemberDescription`.
- Remove all “modifiable” packages.

Resolution:

Defer this issue. This issue was filed after the comment deadline, and its resolution will have a broad impact and has dependencies on the resolutions of other issues. It will be better to address it later, when there is sufficient time to make and review the changes thoroughly.

Disposition: **Deferred**

OMG Issue No: 16531

Title: Getting rid of the Bootstrap object

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Critical~~ Significant

This issue was marked "Critical" when it was filed, but it does not impact the implementability of the specification. It concerns fitness for the expected execution environment as well as usability.

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.

Discussion:

The concern for simplicity and usability expressed above is valid and important. The response to this concern will be constrained by requirements, including those below. In particular, the statement above that the `Bootstrap` class exists only to enable single applications to use two DDS implementations is incorrect.

Requirements:

The `Bootstrap` class is designed to avoid the brittle mixing of concrete implementation with abstract specification, which would occur if either the specification mandated implementation or if vendors re-implemented different classes with the "same" names. In addition, it is designed to enable the following deployment scenarios:

1. Standalone deployment of a single application using one or more DDS implementations. *(The expected use case for multiple implementations is a DDS-to-DDS bridge.)*
2. Deployment within an OSGi container, which may host multiple independent applications.

a. More than one of application may use DDS internally, unknown to those applications.

b. A given application should allow the injection of a concrete DDS dependency—that is, it should be able to declare, "I depend on DDS" and allow the platform's administrator to choose the implementation. *(This implies that the DDS API itself and the concrete implementation must be deployable as separate bundles, with multiple "application" bundles all depending on a common "specification" bundle.)*

3. Deployment within a Java EE container, including one that—like JBOSS Application Server—uses a unified class loader design. *(Such a deployment is like [1] above, although it expands the use case for multiple DDS implementations beyond bridges, as in [2a].)*

(As a point of comparison, the above needs are met by JMS.)

Design Implications:

The above requirements have two implications:

➔ *They forbid the keeping of any static state* or static dependency on any concrete implementation. Specifically, state pertaining to multiple DDS implementations must be able to coexist within a single class loader in order to support OSGI dependency injection, deployment to JBOSS AS, or DDS-to-DDS bridges.

➔ *They forbid the keeping of any thread-local state.* Especially in the Java EE environment, container threads (for example, from servlet thread pools) may call into DDS APIs. DDS state attached to these threads constitutes a resource leak that will prevent applications from being properly unloaded from the container.

In other words, it is not possible for OMG-provided classes to know by themselves where to delegate their implementation. That knowledge must come from the application.

Design Alternatives:

There are two ways an application can provide the necessary implementation-specific context:

- ***Indirectly****: An (already-created) object can provide it to another object at the time of creation.* This boils down to factory methods: if instance X can store implementation-specific state, and X is a factory for Y, then X can provide that state transparently to Y in a "`createY()`" method. Currently, the specification takes this approach where it is implied by the PIM. Alternatively, it could be extended everywhere: make the `Bootstrap` class (or another class) an explicit factory for every top-level type in the API.

- ***Directly****: The application can pass the calling environment as an argument*. Currently, the specification takes this approach when accessing singletons or constructing instances of top-level classes.

We can choose the right approach in each instance if we understand how `Bootstrap` is used today.

Background: `Bootstrap` Occurrences:

The class is used in the following places:

1. To access per-DDS-implementation singletons: `DomainParticipantFactory` and `DynamicTypeFactory`.
2. To create `Entity`-independent reference objects: `WaitSet`, `GuardCondition`, and `TypeSupport`.

   We could reduce the number of occurrences of `Bootstrap` by making accessors/factory methods for `DynamicTypeFactory`, `WaitSet`, `GuardCondition`, and `TypeSupport` available as instance methods of `DomainParticipantFactory`.
3. To create standalone value objects: `Time`, `Duration`, and `InstanceHandle`. *These occurrences will be hard to eliminate.*
4. To create instances of `Status` classes. *We could eliminate these occurrences of `Bootstrap` by creating `Status` objects from factory instance methods on the corresponding Entity interfaces.*
5. To create instances of built-in topic data types: `ParticipantBuiltinTopicData`, `BuiltinTopicKey`, etc. *These occurrences will be hard to eliminate.*
6. To access convenience sets of `Status Class` objects—the equivalent of `STATUS_MASK_ALL` and `STATUS_MASK_NONE`. We could eliminate these occurrences by making these accessors instance methods.

[Rick] Proposal

The fact that this issue came up indicates that the rationale for the `Bootstrap` class is not sufficiently clear. Add such an expanded rationale to the specification document. Furthermore, rename the class to `ServiceEnvironment` to make its role clearer.

In addition, the method signatures for `Bootstrap.createInstance()` are incorrect for handling some OSGi deployment scenarios: it must be possible to supply the `ClassLoader` to be used to load the implementation class in order for dependencies to be resolved properly. Fix these signatures at the same time.

With respect to the code, see the following revisions, which address the code changes:

- Revision #139: http://code.google.com/p/datadistrib4j/source/detail?r=139

- Revision #151: http://code.google.com/p/datadistrib4j/source/detail?r=151
- Revision #166: http://code.google.com/p/datadistrib4j/source/detail?r=166

In the specification document, replace all occurrences of "`getBootstrap`" with "`getEnvironment`" and "`Bootstrap`" with "`ServiceEnvironment`". These occurrences are in these sections:

- 7.1.4, "Concurrency and Reentrancy"
- 7.2.1, "`ServiceEnvironment` Class" (previously "`Bootstrap` Class")
- 7.3.1, "`DomainParticipantFactory` Interface"
- 7.7.1.1, "`DynamicTypeFactory` Interface"

At the end of section 7.2.1, "`ServiceEnvironment` Class" (previously "`Bootstrap` Class"), add the following paragraphs:

> **Design Rationale (non-normative)**
>
> This class is designed to avoid the brittle mixing of concrete implementation with abstract specification that would occur if either the specification mandated implementation or if vendors re-implemented different classes with the "same" names. In addition, it is designed to enable the following deployment scenarios:
>
> - *Standalone deployment of a single application using one or more DDS implementations*. (The expected use case for multiple implementations is a DDS-to-DDS bridge.)
> - *Deployment within a Java EE or OSGi container, which may host multiple independent applications*. More than one of application may use DDS internally, unknown to other applications. Each of these should be able to declare, "I depend on DDS" and allow the platform's administrator to inject the implementation.
>
> The requirements above preclude a DDS vendor from reimplementing any OMG-provided type, and they preclude OMG-provided types from keeping any static or thread-local state.

Resolution:

The task force feels that this issue requires further discussion before it can be resolved. (It was filed after the comment deadline.) It is therefore deferred.

Disposition: **Deferred**

OMG Issue No: 16535

Title: Large Number of Spurious Import

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Useless Dependency

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

Discussion:

See revision #132: http://code.google.com/p/datadistrib4j/source/detail?r=132. (This revision does not resolve this issue. It addresses only the JavaDoc package files.)

Resolution:

Eventually, use fully qualified types on the `@link` directives, removing any subsequently unnecessary `import` statements.

However, this issue, which was filed after the comment deadline, does not impact the correctness, performance, compatibility, or user experience of the specification. Therefore, while there is a minor issue of “cleanliness”, resolving it is not a priority. Therefore, this issue is deferred.

Disposition: **Deferred**

OMG Issue No: 16536

Title: QoS DSL Needed

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

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 cumbersome, but it also introduces potential for errors.

Discussion:

[Angelo] Proposal: Define a QoS DSL for the DDS-PSM-Java, 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());
```

Resolution:

Defer this issue. It was filed after the comment deadline, and its resolution will be coupled to those of issues #15966 and #16529. Because of this complexity and the short time available, it is not possible to resolve it effectively at this time.

Disposition: **Deferred**

OMG Issue No: 16587

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

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The DDS-PSM-Java provides bucket accessors that "return" an object by "filling in" 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.

Discussion:

[Angelo] Proposal: The solution suggested to avoid bucket accessors and thus side effects is to rely as much as possible on immutable objects (e.g. true 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.

[Rick] The other place there is a high cost of new allocations in on the critical `read`/`take` and `write` paths. I agree that the number of occurrences of this pattern can be reduced. But due to the complexity of choosing which ones to change, and the late submission date of this issue, I recommend deferring it.

Resolution:

Defer this issue. It was submitted after the comment deadline.

Disposition: **Deferred**

Disposition: Closed, no change

OMG Issue No: 16325

Title: Remove unnecessary `DataWriter.write` overloads

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The specification currently provides overloads for `DataWriter.write` that take the following combinations of parameters

1. The sample to write, *without* an instance handle. (If the type is not keyed, no instance handle is necessary. If it is keyed, the instance handle is implicitly nil and will be inferred by the implementation.)
2. The sample to write, without an instance handle but with a time stamp.
3. The sample to write, *with* an instance handle.
4. The sample to write, with both an instance handle and a time stamp.

The overloads would be easier to understand if they formed a progression from fewer parameters to more. We can do this by removing (2).

Discussion:

[Rick] Proposal: Remove the following methods:

```
-    public void write(
-             TYPE instanceData,
-             Time sourceTimestamp) throws TimeoutException;
-    public void write(
-             TYPE instanceData,
-             long sourceTimestamp,
-             TimeUnit unit) throws TimeoutException;
```

Also, update the documentation of the remaining overloads to clarify that if the topic is not keyed, they can be called with a nil `InstanceHandle`.

[Virginie] The instance handle on one hand and the timestamp on the other hand are two orthogonal parameters; I find it rather confusing to introduce here a “fake ordering” between those two things that are totally unrelated. If we were to remove unnecessary overloads, why not selecting just one way to express timestamps instead of the two currently existing?

[Angelo] Not sure this is the right approach of reducing overloads. Perhaps the approach is to have only one way of passing a time object. Yet, it is better to avoid API that allow for parameters to be `NIL` or `null`.

Resolution:

It is not clear that the current set of overloads is a problem. This issue is closed without any changes.

Disposition: **Closed, no change**

OMG Issue No: 16530

Title: Superfluous "`QosPolicy`" Suffix on Policy Types

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Nomenclature

Severity: ~~Medium~~ Minor

This issue was marked as "Medium" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The DDS-PSM-Java uses a superfluous Policy suffix to name the DDS policies which themselves are already included in a "policy" namespace. This suffix should be removed.

Resolution:

Close this issue without any change. In Java, fully qualified names are almost never used, and Eclipse (the most commonly used editing environment) commonly collapses imports so they will not be seen. Therefore, it is relevant and important to include this suffix in the names of QoS policy types (not to mention consistent with the PIM). "Reliability" and "Durability" are just attributes; they don't make sense unless we know what they describe.

Disposition: **Closed, no change**

OMG Issue No: 16534

Title: Constant Values *shall* be defined by the standard

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: Critical

Summary:

Constant values such as the infinite duration, etc. should be defined by the standard as opposed than the implementation.

Discussion:

Infinite duration and other well-defined values *are* defined by the standard. The class defines an operation `infiniteDuration: Duration`. Furthermore, the documentation of the `getDuration` reads, "If this duration is infinite, this method shall return `Long.MAX_VALUE`...."

Similar operations and documentation are given for invalid `Time`, nil `InstanceHandle`, and so on.

Resolution:

Close this issue without any change.

Disposition: **Closed, no change**

OMG Issue No: 16537

Title: Get rid of the `EntityQos` Class

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: Minor

Summary:

The `EntityQos` class does not seem very useful for the DDS user. It might be more useful for the DDS implementer

Resolution:

Close this issue without any change. This type serves two purposes for the DDS user: it gives them a way to work with QoS policies polymorphically, and it binds a type parameter of the `Entity` interface to provide static type safety.

Disposition: **Closed, no change**

OMG Issue No: 16538

Title: `Entity` class allows for breaking invariants

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Significant

This issue was marked "Major" when it was filed, but that is not a recognized severity.

Summary:

The Entity provides some generic methods that seem of doubtful usefulness but then on the other end open up a door for messing up with the invariant of a type or at least raising runtime errors. For instance via the Entity type I can add a non-applicable QoS policy to a DDS entity—this seems weakening the API.

Discussion:

[Angelo] Proposal: Remove all method that might break invariants such as `setQos`, `setListener`, etc.

[Rick] This issue is unclear to me. It does not indicate which methods it considers unimportant, nor does it indicate how type safety may be weakened.

Resolution:

Reject this issue. The generic parameters of this type provide static type safety and polymorphic behavior.

Disposition: **Closed, no change**

OMG Issue No: 16539

Title: `DomainEntity` should be removed

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: Minor

Summary:

What is the value of having the `DomainEntity` class?

Discussion:

[Angelo] The `DomainEntity` class should be removed and the `getParent` method should be migrated to the `Entity` class. The `DomainEntity` is not on any other PSM, thus I don't see why it should be on the Java PSM. Even at a PIM level it is a class w/o attributes and w/o operations—thus a very debatable abstraction.

Resolution:

Close this issue with no change. `DomainEntity` is a classifier directly from the DDS PIM. Its value is to capture the invariant that domain participants have no parent entity, and entities of all other types do. If this type is removed, and `getParent` is moved to `Entity`, this static invariant will become a run-time invariant (i.e. `getParent` will have to return `null` when the object is a participant), making application code less robust.

Disposition: **Closed, no change**

OMG Issue No: 16588

Title: The `Sample` class should provide a method to check whether the data is valid

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked as "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The `Sample` class as defined in the Beta 1 of the DDS-PSM-Java does not define a method "`isValid(): Boolean`" to check whether the data is actually valid. This is a simple-to-fix oversight.

Discussion:

[Angelo] An implementation could decide to provide the last known value. Would be useful to separate the `null` from the strictly valid data.

Resolution:

This issue is closed without any changes. The `Sample` class *does* provide a method to check whether the data is valid. There is no oversight. As described in section 7.6.2, "Sample Interface":

> Each sample is represented by an instance of the `org.omg.dds.sub.Sample` interface. It provides its data via a `getData` method; if there is no valid data (corresponding to a false value for `SampleInfo.valid_data` in the IDL PSM), this operation returns `null`.

The existing design is appropriate. The alternative—returning a non-`null` data object, and then making people check a flag that tells them not to use that object—would be extremely error-prone.

Disposition: **Closed, no change**

OMG Issue No: 16589

Title: Misnamed Listener Helper

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: Minor

Summary:

The names of the classes `DataReaderAdapter`/`DataWriterAdapter` are misleading since what they are really providing are listeners with some default behavior.

Discussion:

[Angelo] Proposal: Rename the classes `DataReaderAdapter`/`DataWriterAdapter` to `SimpleDataReaderListener` and `SimpleDataWriterListener`.

For the `SimpleDataReaderListener` one could implement trivially all the method but the one that notifies the availability of data, e.g. `onDataAvailable`.

Resolution:

This issue is closed without any changes. Providing no-op listener implementations, and replacing “Listener” with “Adapter” in the class name, is a common JDK idiom. You will see it throughput AWT and Swing, for example, which abound with listeners. The rationale for this pattern is described in section 7.2.7.2, “Listeners”.

Disposition: **Closed, no change**

Disposition: Duplicate/merged

OMG Issue No: 16056

Title: Data access from `DataReader` using `java.util.List`

Source:

Thales (André Bonhof, andre.bonhof@nl.thalesgroup.com)

Nature: Enhancement

Severity: Minor

Summary:

Currently the `DataReader` provides `read()` and `take()` methods that return a special type of `java.util.ListIterator`: `Sample.Iterator`. The `Iterator` is not the most convenient way to access data retrieved from the `DataReader` (e.g. an `Iterator` can only be traversed once).

Propose to modify all `read()`/`take()` operations currently returning an `Iterator` to let them return a `java.util.List`. The `List` is more developer friendly, as it can be traversed multiple times and a `List` is also an `Iterable` with the added benefit that it can be used in Java's "`foreach`" statement:

```
List<Sample<TYPE>> data = dataReader.read();
for (Sample<Type> sample : data) {
  // ...
}
```

Resolution:

Merge this issue with #16321, which proposes other changes to the read/take overloads.

- Overloads that return a loan should do so in the form of a `ListIterator` implementation, which will allows multiple forward and backward navigation of elements. The loaned samples should *not* be returned as a `List`, as retrieving an iterator from a list would force critical-path memory allocation—in direct contradiction of the low-latency goal of the loaning operations.
- Overloads that return a copy should continue to accept a `List` to be filled in and should return a reference to the same list for convenience. These overloads will therefore support the "for-each" construct requested by this issue.

Disposition: **Merged with issue #16321**

OMG Issue No: 16316

Title: Improve usability of “bucket” accessors

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The third bullet at the end of section 7.1.5, “Method Signature Conventions”, reads:

> - Accessors for properties that are of mutable types, and that may change asynchronously after they are retrieved, are named `get<`*`PropertyName`*`>`. They take a pre-allocated object of the property type as their first argument, the contents of which shall be overwritten by the method. To facilitate method chaining, these methods also return a reference to this argument. This pattern forces the caller to make a copy, thereby avoiding unexpected changes to the property. An Entity’s status is an example of a property of this kind.

(This pattern of passing a container to an object for that object to “fill in” has sometimes been referred to as a “bucket” pattern.)

In cases where object-allocation performance is not a significant concern, the usability of this pattern can be improved with a trivial addition: allow the caller to pass in a null “bucket”, and require the implementation to allocate and return a new object with the appropriate contents.

Discussion:

[Rick] To resolve this issue by itself, we could add a sentence to the bullet that indicates that a null argument is permitted. Specifically, replace the third bullet in section 7.1.5, “Method Signature Conventions” with the following:

> - Accessors for properties that are of mutable types, and that may change asynchronously after they are retrieved, are named `get<`*`PropertyName`*`>`. They take a pre-allocated object of the property type as their first argument, the contents of which shall be overwritten by the method. To facilitate method chaining, these methods also return a reference to this argument. The caller may alternatively pass a `null` argument into such accessor methods, in which case the implementation shall allocate a new object, set its contents appropriately, and return it. This pattern forces the caller to make a copy, thereby avoiding unexpected changes to the property. An Entity’s status is an example of a property of this kind.

Resolution:

Issue #16587 also deals with bucket accessors and has a broader scope. This issue should be merged with that one.

Disposition: **Merged with issue #16587**

OMG Issue No: 16322

Title: `DynamicDataFactory.createData` missing a parameter

Source:

RTI (Rick Warren, rick.warren@rti.com)

Severity: Significant

Summary:

According to DDS-XTypes, the operation `DynamicDataFactory.create_data` (`createData` in this PSM) takes a parameter that indicates the `DynamicType` of the new data object to create. That parameter is missing, leaving implementations with no way to determine—and applications with no way to specify—the type of the created object.

Resolution:

This issue is obsolete if the proposal for issue #16324 is accepted: that proposal calls for `DynamicDataFactory` to be eliminated entirely. Merge this issue with that one.

Disposition: **Merged with issue #16324**

OMG Issue No: 16533

Title: QoS Policies ID class vs. numeric ID

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: Critical

Summary:

QoS Policies define a nested ID class for capturing the Policy ID and PolicyName.

Discussion:

[Angelo] Remove the ID class and (1) use Java introspection for accessing the policy name, and (2) define an integral ID for specifying the policy ID.

Notice that `getId` and `getName` methods are also needed.

[Rick] This issue is unclear to me. I do not understand what problem it is describing. It seems to be redundant with issue #16369.

Resolution:

This issue is a duplicate of issue #16369 and should be merged with that one.

Disposition: **Duplicate of issue #16369**

OMG Issue No: 16540

Title: `DataReader` API

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked as "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The read API currently seems a bit too complicated. In some in some instances it provides part of the results as a return value and the rest by means of arguments.

This does not feel right and again violates one of the key goals of having a new PSM: simplicity and usability.

The API does not provide a way of deciding if one wants to `read`/`take` only valid data. This is a remark true in general for DDS, which needs to be fixed for all PSM as well as for the PIM!

The following methods on the `DataReader` interface are superfluous:

- `cast`
- `createSample`

Discussion:

This issue has three parts: removing unnecessary methods, reading only samples with valid data, and simplifying method overloads.

- The `cast()` method is not superfluous; it is the only type-safe way to narrow a `DataReader<?>` to a `DataReader<Foo>`. This method can potentially use internal state of the reader to provide immediate run-time type safety. The only alternative is for the application code to use a type cast like this: "`(DataReader<Foo>)`". But such a cast is meaningless because of type erasure and will generate a compiler warning. If there is a type mismatch, it will potentially not be caught until later.
- Issue #16324 already eliminated the `createSample` method.
- Reading or taking only valid data samples may or may not be semantically meaningful and should be addressed in the PIM first, so that the semantics can be defined. At that point, the method can be introduced into this PSM in an RTF.

- Issue #16321 already proposes simplifying the `read`/`take` overloads. Since this is the only remaining part of this issue not addressed in the bullets above, this issue can be merged with that one.

Resolution:

Merge this issue with issue #16321.

Disposition: **Merged with issue #16321**

OMG Issue No: 16542

Title: Avoid unnecessary side effects on the `DataWriter` API

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked as "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The methods that access the communication statuses should simply return an object as opposed to also require it as an argument. As the Status is immutable there is no risk in the client code changing it, thus no copies required!

Discussion:

[Rick] This issue seems to apply only to writer status accessors. It is unclear to me why this issue is filed against the writer and not all other such status accessors.

Resolution:

Issue #16587 is a blanket issue about "bucket" accessors; this issue should be merged with that one.

Disposition: **Merge with issue #16587**

OMG Issue No: 16543

Title: Statuses API should be improved and made type-safe

Source:

PrismTech (Angelo Corsaro, angelo@icorsaro.net)

Nature: Architectural

Severity: ~~Major~~ Minor

This issue was marked as "Major" when it was filed, but that is not a recognized severity. It does not impact the implementability of the specification.

Summary:

The DDS-PSM-Java currently passes statuses via collections. However this does not make it either efficient or simple to represent collections of related statuses, such as the Read Status, etc.

Discussion:

[Angelo] The suggestion is to add a `ReadState` as currently present on the DDS-PSM-Cxx to simplify the API and make it simpler to support the most common use cases.

[Rick] This issue is unclear to me. I take it that this issue is not referring to actual `Status` objects. The reference to what is called "Read Status" or "Read State" and to DDS-PSM-Cxx makes me think it is related to the DDS-PSM-Cxx class called "`ReaderState`", which encapsulates sets of Sample States, View States, and Instance States. This change is already proposed in DDS-PSM-Java as part of the proposed resolution to issue #16321.

Resolution:

Merge this issue with issue #16321.

Disposition: **Merge with issue #16321**