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

Document Number: ptc/2011-??-??

Task Force Chair: Rick Warren (RTI)

Specification

Revised specification (clean): ptc/2011-??-??

Revised specification (change-bar): ptc/2011-??-??

Accompanying documents

Inventory: ptc/2011-??-?? Non-normative

omgdds.jar: ptc/2011-??-?? Normative

omgdds\_src.zip: ptc/2011-??-?? 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 3

Disposition: Under Discussion 5

OMG Issue No: 15966 6

Title: XML-Based QoS Policy Settings (DDS-PSM-Cxx/DDS-PSM-Java) 6

OMG Issue No: 15968 7

Title: formal description of how topic types are mapped to Java classes needed 7

OMG Issue No: 16050 8

Title: duplicate put definition resulting in a name clash 8

OMG Issue No: 16056 10

Title: Data access from DataReader using java.util.List 10

OMG Issue No: 16104 11

Title: Missing *behavioral* descriptions of the interface 11

OMG Issue No: 16316 13

Title: Improve usability of “bucket” accessors 13

OMG Issue No: 16317 15

Title: Update specification to reflect DDS-XTypes FTF1 issue resolutions 15

OMG Issue No: 16318 17

Title: Entity.setListener is missing listener mask 17

OMG Issue No: 16319 18

Title: Unclear blocking behavior for WaitSet.waitForConditions overloads that don’t specify timeout 18

OMG Issue No: 16320 19

Title: Incorrect topic type specification in DomainParticipant.createMultiTopic 19

OMG Issue No: 16321 20

Title: Too many read/take overloads 20

OMG Issue No: 16322 21

Title: DynamicDataFactory.createData missing a parameter 21

OMG Issue No: 16323 22

Title: Logically ordered types should implement java.lang.Comparable 22

OMG Issue No: 16324 23

Title: Improve polymorphic sample creation 23

OMG Issue No: 16325 25

Title: Remove unnecessary DataWriter.write overloads 25

OMG Issue No: 16326 26

Title: copyFromTopicQos signatures are not correct 26

OMG Issue No: 16327 28

Title: Parent accessors should be uniform across Entities and Conditions 28

OMG Issue No: 16328 29

Title: DataReader.createReadCondition() is useless 29

OMG Issue No: 16369 30

Title: QosPolicy.Id enumeration is redundant 30

OMG Issue No: 16529 31

Title: Modifiable Types should be removed and replaced by values (e.g. immutable types) 31

OMG Issue No: 16530 34

Title: Superfluous "QosPolicy" Suffix on Policy Types 34

OMG Issue No: 16531 35

Title: Getting rid of the Bootstrap object 35

OMG Issue No: 16532 37

Title: RxO QoS Policies should be Comparable (idem for QoS) 37

OMG Issue No: 16533 38

Title: QoS Policies ID class vs. numeric ID 38

OMG Issue No: 16534 39

Title: Constant Values *shall* be defined by the standard 39

OMG Issue No: 16535 40

Title: Large Number of Spurious Import 40

OMG Issue No: 16536 41

Title: QoS DSL Needed 41

OMG Issue No: 16537 42

Title: Get rid of the EntityQos Class 42

OMG Issue No: 16538 43

Title: Entity class allows for breaking invariants 43

Source: 43

OMG Issue No: 16539 44

Title: DomainEntity should be removed 44

Source: 44

OMG Issue No: 16540 45

Title: DataReader API 45

OMG Issue No: 16541 46

Title: A Status is not an Event. An Event is not a Status, it notifies a status change. 46

OMG Issue No: 16542 48

Title: Avoid unnecessary side effects on the DataWriter API 48

OMG Issue No: 16543 49

Title: Statuses API should be improved and made type-safe 49

Disposition: Resolved 50

OMG Issue No: 11111 51

Title: The Title 51

Disposition: Deferred 52

OMG Issue No: 11111 53

Title: The Title 53

Disposition: Closed, no change 54

OMG Issue No: 11111 55

Title: The Title 55

Disposition: Duplicate/merged 56

OMG Issue No: 11111 57

Title: The Title 57

# 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: 7 November, 2011

## Revision / Finalization Task Force Membership

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

## Issue Disposition:

|  |  |  |
| --- | --- | --- |
| **Disposition** | **Number of Occurrences** | **Meaning of Disposition** |
| Resolved | 0 | 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 | 11 February 2011 | 15418, 15688, 15689 |
| 2 | 17 February 2011 | 15702 |

|  |  |  |
| --- | --- | --- |
| **Voter** | **Vote in poll 1** | **Vote in poll 2** |
| Angelo Corsaro | Yes | No | Abstain | Did not vote | Yes | No | Abstain | Did not vote |
| Fabrizio Morciano | Yes | No | Abstain | Did not vote | Yes | No | Abstain | Did not vote |
| Rick Warren | Yes | No | Abstain | Did not vote | Yes | No | Abstain | Did not vote |
| Virginie Watine | Yes | No | Abstain | Did not vote | Yes | No | Abstain | Did not vote |

## 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 | **0** | **15702, 15969, 15976** |
| **Minor** - Fixed minor problems with normative parts of the specification | **0** | **15688, 15689** |
| **Support Text** -Changes to descriptive, explanatory, or supporting material. | **0** | **15690, 15697** |

Disposition: Under Discussion

OMG Issue No: 15966

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

Source:

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

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.

Proposed Resolution:

The 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.
2. From an existing QoS object.

Once a Builder is created, it would allow the QoS values it holds to be modified. In this way, the Builder would take the place of the Modifiable QoS types, contributing also to the resolution of issue #16529.

Proposed Revised Text:

TODO

Proposed Disposition: Resolved

Disposition: Under Discussion

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 used by DDS implementations.

Discussion:

[Rick] The issue report reads, “…to ensure that different/old mappings are used by DDS implementations”. I suspect the intention was to say, “…to ensure that different/old mappings are *not* used by DDS implementations”. 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.

An updated Plain Language Binding for Java could potentially overlap with the Java Type Representation that DDS-PSM-Java specifies. The FTF should discuss the extent to which these two concepts should be aligned/merged.

Proposed Resolution:

TODO

Proposed Revised Text:

TODO

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16050

Title: duplicate put definition resulting in a name clash

Source:

Thales (André Bonhof, [andre.bonhof@nl.thalesgroup.com](mailto: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<?,?>)

Discussion:

Possible ways to resolve this:

* 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.
* 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.

[Rick] See also issue #16369, which will potentially impact the same method signature.

Proposed Resolution:

See revision #116: <http://code.google.com/p/datadistrib4j/source/detail?r=116>.

See also revision #136: <http://code.google.com/p/datadistrib4j/source/detail?r=136>.

Proposed Revised Text:

TODO

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16056

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

Source:

Thales (André Bonhof, [andre.bonhof@nl.thalesgroup.com](mailto: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) {

// ...

}

Proposed Resolution:

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

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

Proposed Disposition: Merged with issue #16321

Disposition: Under Discussion

OMG Issue No: 16104

Title: Missing *behavioral* descriptions of the interface

Source:

Thales (André Bonhof, [andre.bonhof@nl.thalesgroup.com](mailto: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.

Proposed 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.

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16316

Title: Improve usability of “bucket” accessors

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed Resolution:

Add a sentence to the bullet that indicates that a null argument is permitted.

Proposed Revised Text:

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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16317

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

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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.

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16318

Title: Entity.setListener is missing listener mask

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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.

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

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](mailto:rick.warren@rti.com))

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.

Proposed Resolution:

Remove the clause “throws TimeoutException” from these method declarations.

Proposed 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

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16320

Title: Incorrect topic type specification in DomainParticipant.createMultiTopic

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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.

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16321

Title: Too many read/take overloads

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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:

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

Note that this issue should be resolved at the same time as #16056.

Proposed Resolution:

See below.

Proposed Revised Text:

See the following revisions:

* Revision #131: <http://code.google.com/p/datadistrib4j/source/detail?r=131>
* Revision #135: <http://code.google.com/p/datadistrib4j/source/detail?r=135>

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16322

Title: DynamicDataFactory.createData missing a parameter

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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.

Proposed Disposition: Merged with issue #16324

Disposition: Under Discussion

OMG Issue No: 16323

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

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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

Proposed Revised Text:

See revision #122: <http://code.google.com/p/datadistrib4j/source/detail?r=122>. This update encompasses Bit, Duration, and Time.

See revision #134: <http://code.google.com/p/datadistrib4j/source/detail?r=134>. This update encompasses InstanceHandle.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16324

Title: Improve polymorphic sample creation

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed 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>.*

1. 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.

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16325

Title: Remove unnecessary DataWriter.write overloads

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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).

Proposed Resolution:

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.

Proposed Revised Text:

See revision #146: <http://code.google.com/p/datadistrib4j/source/detail?r=146>. These changes are also available in the attached file diff\_omg\_issue\_16325.txt.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16326

Title: copyFromTopicQos signatures are not correct

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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()),

…);

Proposed 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);

Proposed Revised Text:

See revision #126: <http://code.google.com/p/datadistrib4j/source/detail?r=126>.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16327

Title: Parent accessors should be uniform across Entities and Conditions

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed Resolution:

Rename StatusCondition.getEntity to getParent.

Rename ReadCondition.getDataReader to getParent.

Proposed 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”.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16328

Title: DataReader.createReadCondition() is useless

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed Resolution:

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

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16369

Title: QosPolicy.Id enumeration is redundant

Source:

RTI (Rick Warren, [rick.warren@rti.com](mailto:rick.warren@rti.com))

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.

Proposed Resolution:

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

Proposed Revised Text:

See revisions:

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

Proposed Disposition: Resolved

Disposition: Under Discussion

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:

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.

Recommendation:

* Replace this pattern with a more explicit Builder pattern and/or a DSL 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.

Proposed Resolution:

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 fragment both lead 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

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16530

Title: Superfluous "QosPolicy" Suffix on Policy Types

Source:

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

**Nature:** Nomenclature

**Severity:** Medium

Summary:

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

Proposed Resolution:

This suffix should be removed.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

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 which 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 Bootstrap class is one way to meet the following needs:

* Allowing the specification 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.
* Allowing multiple DDS implementations to coexist in the same JVM.

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

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.*

1. To create standalone value objects: Time, Duration, and InstanceHandle. *These occurrences will be hard to eliminate.*
2. 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.*
3. To create instances of built-in topic data types: ParticipantBuiltinTopicData, BuiltinTopicKey, etc. *These occurrences will be hard to eliminate.*
4. To access convenience sets of Status Class objects—the equivalent of STATUS\_MASK\_ALL and STATUS\_MASK\_NONE. We could eiminate these occurrences by moving these accessors

Proposed Resolution:

Rename Bootstrap to ServiceImplementationProvider—the user will no longer create one to “bootstrap” their application, so the name will no longer make sense.

Introduce the concept of a “current” ServiceImplementationProvider—per thread or default for the JVM. By setting which provider is “current”, an application can continue to use different DDS implementations within the same JVM instance.

The existence of this new singleton will allow the standard classes to access that object themselves internally rather than requiring an instance to be passed in. Remove all arguments of this type.

The resulting API will not expose applications to the Bootstrap/ ServiceImplementationProvider.

Proposed Revised Text:

See revision #139: <http://code.google.com/p/datadistrib4j/source/detail?r=139>.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16532

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

Source:

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

**Nature:** Architectural

**Severity:** Critical

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.

Proposed Resolution:

Ensure that all RxO Policies implement the Comparable interface. This is pretty logical as for this QoS Policies it has to be established a total order.

Let QoS classes also implement a comparable interface.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16533

Title: QoS Policies ID class vs. numeric ID

Source:

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

**Nature:** Architectural

**Severity:** Critical

Summary:

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

Proposed Resolution:

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.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16534

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

Source:

PrismTech (Angelo Corsaro, [angelo@icorsaro.net](mailto: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.

Proposed Resolution:

Define constants as part of the API.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16535

Title: Large Number of Spurious Import

Source:

PrismTech (Angelo Corsaro, [angelo@icorsaro.net](mailto: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

Proposed Resolution:

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

Proposed Revised Text:

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.)

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16536

Title: QoS DSL Needed

Source:

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

**Nature:** Architectural

**Severity:** Major

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.

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 classes should implement the Comparable interface as they need to provide a total order... Otherwise how can one do RxO?

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16537

Title: Get rid of the EntityQos Class

Source:

PrismTech (Angelo Corsaro, [angelo@icorsaro.net](mailto: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

Proposed Resolution:

Remove the EntityQos class from the public API

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16538

Title: Entity class allows for breaking invariants

Source:

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

**Nature:** Architectural

**Severity:** Major

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.

Proposed Resolution:

Remove all method that might break invariants such as setQos, setListener, etc.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16539

Title: DomainEntity should be removed

Source:

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

**Nature:** Architectural

**Severity:** Minor

Summary:

What is the value of having the DomainEntity class?

Proposed Resolution:

The DomainEntity class should be removed and the getParent method should be migrated to the Entity class.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16540

Title: DataReader API

Source:

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

**Nature:** Architectural

**Severity:** Major

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 goal 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:

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.

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. Issue #16324 already proposes eliminating the createSample method. This issue can be merged with that one of those.

Proposed Resolution:

Merge this issue with issue #16321.

Proposed Disposition: Merged with issue #16321

Disposition: Under Discussion

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](mailto:ANGELO@ICORSARO.NET))

**Nature:** Architectural

**Severity:** Major

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.

*See revision #133: <http://code.google.com/p/datadistrib4j/source/detail?r=133>.*

*See revision #138: <http://code.google.com/p/datadistrib4j/source/detail?r=138>, which removed DataAvailableStatus and DataOnReadersStatus—those classes don’t do anything anymore.*

Proposed 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.

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16542

Title: Avoid unnecessary side effects on the DataWriter API

Source:

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

**Nature:** Architectural

**Severity:** Major

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!

Proposed Resolution:

Apply the changes suggested in the description above.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

OMG Issue No: 16543

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

Source:

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

**Nature:** Architectural

**Severity:** Major

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.

Proposed Resolution:

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.

Proposed Revised Text:

Proposed Disposition: Resolved

Disposition: Under Discussion

Disposition: Resolved

OMG Issue No: 11111

Title: The Title

Source:

Organization (Person, email)

Summary:

This issue is not a big deal.

Resolution:

Make a very small change.

Revised Text:

The change looks like this in the spec.

Disposition: Resolved

Disposition: Deferred

OMG Issue No: 11111

Title: The Title

Source:

Organization (Person, email)

Summary:

This issue is not a big deal.

Resolution:

Make a very small change.

Revised Text:

The change looks like this in the spec.

Disposition: Deferred

Disposition: Closed, no change

OMG Issue No: 11111

Title: The Title

Source:

Organization (Person, email)

Summary:

This issue is not a big deal.

Resolution:

Make a very small change.

Revised Text:

The change looks like this in the spec.

Disposition: Closed, no change

Disposition: Duplicate/merged

OMG Issue No: 11111

Title: The Title

Source:

Organization (Person, email)

Summary:

This issue is not a big deal.

Resolution:

Make a very small change.

Revised Text:

The change looks like this in the spec.

Disposition: Duplicate