Report of the Extensible and Dynamic Topic Types for DDS Finalization Task Force 1.0 to the OMG Platform Technical Committee 30 March 2012

Document Number:	**ptc/2012-02-01**
Task Force Chair:	**Rick Warren (RTI)**

Specification

Revised specification (clean):	**ptc/2012-02-02**
Revised specification (change-bar):	**ptc/2012-02-03**

Accompanying documents

Inventory:	**ptc/2012-02-04**	**Non-normative**
dds-xtypes_model.eap:	**ptc/2012-02-05**	**Non-normative**
dds-xtypes_model.xmi:	**ptc/2012-02-06**	**Normative**
dds-xtypes_type_definition.xsd:	**ptc/2012-02-07**	**Normative**
dds-xtypes.idl:	**ptc/2012-02-08**	**Normative**

Template: omg/09-06-01

Table of Contents

Rick Warren 3/13/12 9:01 AM
Deleted: 2/13/12

Summary of DDS-XTypes FTF Activities

Formation

- Chartered By: Platform TC
- On: 25 March, 2011; Arlington, VA
- Comments Due Date: 28 November 2011
- Report Due Date: 20 February 2012
- Report Deadline: 30 March 2012

Revision / Finalization Task Force Membership

Member	Organization	Status
Angelo Corsaro	PrismTech	Charter
Dario Di Crescenzo	Selex SI	Charter
Charlie Fudge	Naval Surface Warfare Center (NSWC)	Charter
Sam Mancarella	Sparx Systems	Charter
Nikolay Malitsky	Brookhaven National Laboratory	Charter
Ken Rode	Gallium Visual Systems	Charter
John Thier	General Dynamics AIS	Charter
Char Wales	MITRE	Charter
Rick Warren	Real-Time Innovations (RTI)	Charter (chair)
Virginie Watine	Thales	Charter
Johnny Willemson	Remedy IT	Charter
~~Chuck Zublic~~	~~Northrop Grumman~~	~~Charter~~ ~~*Removed*~~

Issue Disposition:

Disposition	Number of Occurrences	Meaning of Disposition
Resolved	27	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	2	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	1	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	1	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	9 December 2011	15946, 15981, 16271, 16364, 16365
2	13 January 2012	15969, 15976, 16007, 16237, 16240, 16241, 16242, 16243, 16366, 16367, 16368, 16559, 16720 *The proposals for the following issues did not pass; they will be updated and re-voted later: 16237, 16243, 16368*
3	13 February 2012	15702, 16097, 16236, 16237, 16238, 16239, 16243, 16368, 16561, 17018, 17019, 17020, 17021, 17022, 17023, 17090

Voter	Vote in poll 1	Vote in poll 2	Vote in poll 3
Angelo Corsaro	Yes on all issues	Yes on 15969, 15976, 16007, 16240, 16241, 16242, 16366, 16367, 16559, 16720 No on 16237, 16243, 16368	Yes on all issues
Dario Di Crescenzo	Yes on all issues	Yes on 15969, 15976, 16007, 16240, 16241, 16242, 16243, 16366, 16367, 16559, 16720 No on 16237, 16368	Yes on all issues
Charlie Fudge	Abstain on all issues	Abstain on all issues	Abstain on all issues
Sam Mancarella	Abstain on all issues	*Did not vote*	Abstain on 17018 Yes on all other issues
Nikolay Malitsky	Yes on all issues	Yes on all issues	Yes on all issues
Ken Rode	Abstain on all issues	Abstain on all issues	Yes on all issues

John Thier	Yes on all issues	Abstain on all issues	Did not vote
Char Wales	Yes on all issues	Yes on 15969, 15976, 16007, 16240, 16241, 16242, 16366, 16367, 16559, 16720 Abstain on 16237, 16243, 16368	Yes on all issues
Rick Warren	Yes on all issues	Yes on all issues	Yes on all issues
Virginie Watine	Yes on all issues	Yes on 15969, 15976, 16007, 16240, 16241, 16242, 16366, 16367, 16559, 16720 No on 16237, 16243, 16368	Yes on all issues
Johnny Willemson	Yes on all issues	Abstain on all issues	Yes on all issues
~~Chuck Zublic~~	~~*Did not vote*~~	~~*Did not vote*~~ ~~*After failing to vote in a series of two consecutive polls, Chuck Zublic is removed from this FTF.*~~	—

Summary of Changes Made

The DDS-XTypes 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	—

Significant - Fixed problems with normative parts of the specification that raised concern about implementability	13	15702, 15969, 15976, 15981, 16007, 16097, 16240, 16243, 16365, 16366, 16367, 16559, 16561
Minor - Fixed minor problems with normative parts of the specification	10	15946, 16237, 16238, 16239, 16241, 16242, 16364, 16368, 16720, 17090
Support Text -Changes to descriptive, explanatory, or supporting material.	8	16236, 16271, 17018, 17019, 17020, 17021, 17022, 17023

Disposition: Resolved

OMG Issue No: 15702

Title: Allow more flexible type consistency enforcement

Source:

Real-Time Innovations (RTI) (Mr. Rick Warren, rick.warren@rti.com)

Thales (Ms. Virginie Watine, virginie.watine@thalesgroup.com)

Severity: Significant

Summary:

Section 7.6.1.3.1, *`TypeConsistencyEnforcementQosPolicy`: Conceptual Model*, identifies the type consistency enforcement kinds. This QoS policy is request-offer (RxO); readers and writers must match exactly in order to communicate. However, this requirement for an exact match is not required by implementations and is too restrictive for users.

Discussion:

A primary use case for the DDS-XTypes specification is the independent evolution of components and subsystems. One important subset of this use case is the initial introduction of a Service implementation compliant with this specification: a reader or writer may support the Type Consistency Enforcement policy while its matched peer does not. Therefore, the following invariants must hold:

1. *Reader and writer must reach the same conclusion about whether communication is possible*. Note that a non-DDS-XTypes-compliant implementation will enforce type consistency on the basis of exact name equality.
2. *The reader must be able to parse correctly the serialized data it receives from the writer*. Note that a non-DDS-XTypes-compliant reader implementation may not be able to skip information that it does not understand nor to infer information it did not receive. A non-DDS-XTypes-compliant writer implementation may not be able to determine structural compatibility with its matched readers at all.
3. *Upgrading the DDS implementation without modifying application code must not cause communication to cease*. The default QoS of a DDS-XTypes-compliant implementation—as well as the QoS that it infers from a non-compliant peer—must not result in a QoS incompatibility or type inconsistency that would not otherwise have occurred.
4. *Upgrading the DDS implementation without modifying application code MAY cause communication to begin*. It could be that two applications have types that are not quite the same; however, they may be close enough to allow communication under this specification.

The following table matches compliant and non-compliant readers and writers and indicates desirable and permissible behaviors in each case. The notation "X" indicates an implementation compliant with DDS-XTypes. The notation "–" indicates an implementation that is *not* compliant with DDS-XTypes. The first marking indicates whether the middleware has been upgraded; the second indicates whether the application code/configuration has been upgraded. "Equal" means exact structural type equality. "Assignable" means type assignability but not necessarily strict equality. "None" means no structural type checking at all.

	XX Reader	**X– Reader**	**– – Reader**
XX Writer	*Assignable* is the desired default, although either party might override this behavior to *Equal* or *None*.	*Assignable* is the desired default, although the writer might override this behavior to *Equal* or *None*.	*Equal* is the desired default, because the reader cannot respond flexibly. The writer might theoretically elect to change this to *None*, although this would be unsafe, because the writer might push incompatible data to the reader with no assurance that the reader is prepared to handle it.
X– Writer	*Assignable* is the desired default, although the reader might override this behavior to *Equal* or *None*.	*Assignable* is the desired default, although it is likely the same as *Equal*, if the endpoints were already communicating.	*Equal* is the desired default, because the reader cannot respond flexibly.

– – Writer	*Assignable* is the desired default, but it can be enforced only on the reader side[1], which might override it to *Equal* or *None*. The writer will perform no checking in any case.	*Assignable* is the desired default, but it can be enforced only on the reader side. It is likely the same as *Equal*, if the endpoints were already communicating.	*None* is the existing behavior; neither party can be configured to do anything else. However, the implied contract is *Equal*.

Resolution:

Rick Warren 2/13/12 9:28 AM
Deleted: Proposed

The resolution below is closely related to the resolutions of issues #16097 and #16561.

The existing type consistency enforcement QoS policy, which governs both type signature consistency and type definition structural consistency, shall be simplified. It shall be *removed* from the Topic and `DataWriter` QoS and the corresponding built-in topic data types. It shall *remain* in the `DataReader` QoS and in the Subscription built-in topic data type. Its "kind" enumeration shall have the following values:

- `DISALLOW_TYPE_COERCION`: The `DataWriter` and the `DataReader` must support the same data type in order for them to communicate.
- `ALLOW_TYPE_COERCION`: The `DataWriter` and the `DataReader` need not support the *same* data type(s) in order for them to communicate as long as the reader's type is assignable from the writer's type.

The default settings shall be as follows:

- For compliant readers: `ALLOW_TYPE_COERCION`
- Inferred for non-compliant readers: `DISALLOW_TYPE_COERCION`

Type compatibility shall be determined according to the following steps:s

1. The `TypeObject` type definition is considered first, provided that it is available for both the Publication and the Subscription. (This is anticipated to be the most common case for DDS-XTypes-conformant implementations.) If the Subscription allows type coercion, then its type must be assignable from the type of the Publication. If the Subscription does not allow type coercion, then its type must be equal to that of the Publication.

[1] If the reader determines that the types are not consistent, a warning might be in order, because the writer will think that the reader is communicating with it, but the reader will in fact be ignoring it.

2. If either the Publication or the Subscription does not provide any `TypeObject` definitions, then the type names are consulted. (This case is important in cases where a given component cannot, or does not wish to, provide concrete type definitions. For example, it may not support XTypes, it may wish to conserve bandwidth, it may not wish to reveal its type definition(s), or it may not even be type-aware—it may simply record binary payloads, for example.) The Subscription and Publication `type_name` fields must match exactly.

If the reader and writer are *not* type-compatible, the middleware raises an `INCONSISTENT_TOPIC` status change.

The following behaviors will result from the default values and evaluation orders above when writers and readers have default QoS:

- *Compliant writers will communicate with compliant readers.* The reader will allow type coercion. This behavior is what we want, and it is unchanged from the current DDS-XTypes default.
- *Compliant writers will communicate with non-compliant readers.* The reader will not allow type coercion, so the types (really the type names) must match exactly. This behavior is unchanged from the implied contract of DDS implementations prior to DDS-XTypes.

  (Suppose the writer has out-of-band knowledge of the reader's type definition. If the writer determines that the types are not equal, it will consider the topic inconsistent and forbid communication. The reader will be unable to detect this inconsistency. However, the reader will not receive any data from the writer, so deserialization errors will be prevented just the same.)
- *Non-compliant writers will communicate with compliant readers.* The reader will allow type coercion. This behavior provides maximum flexibility without requiring changes to deployed non-compliant applications.

  (Suppose the reader has out-of-band knowledge of the writer's type definition. If the reader determines that the types are not assignable, the writer will be unable to detect this inconsistency and may send data to the reader nonetheless. The reader implementation will need to detect and eliminate this extraneous data without attempting to deserialize it—for example, by checking upon receiving the data from the socket whether the source data is actually from a matched writer. This edge case is little different from other preexisting cases, such as the use by a non-matching writer of a multicast address used by the reader. Therefore, it is not expected to be a barrier to implementation.)

Revised Text:

Rick Warren 2/13/12 9:28 AM
Deleted: Proposed

Rename section 7.6.1, "Endpoint Matching", to "Discovery and Endpoint Matching". Remove section 7.6.2, "Discovery"—its contents need significant

revision, and it makes more sense to merge them into 7.6.1 than to leave them as a separate section.

In the first paragraph of section 7.6.1, replace, "…endpoint matching process…" with "discovery and endpoint-matching process…". At the end of the same paragraph, replace "two categories" with "three categories" and insert a new bullet in the middle of the list:

- **Discovery-Time Data Typing**: The dynamic features of this specification depend on the ability of components to discover the data types used by their peers.

Add the following paragraph to the end of section 7.6.1.1.3, "`DataRepresentationQosPolicy`: Platform-Specific API":

The topic, publication, and subscription built-in topic data types shall each indicate the data representation of the associated entity with a new member:

```
@ID(0x0073) DDS::DataRepresentationQosPolicy representation;
```

Add the following new section after the existing section 7.6.1.1, "Data Representation QoS Policy". (The existing section 7.6.1.2, "Type Signature", is removed by the resolution to issue #16561.)

7.6.1.2 Discovery-Time Data Typing

The topic, publication, and subscription built-in topic data structures shall each indicate the type(s) used for communication by the associated entity. These declarations shall be as follows:

```
@ID(0x0007) ObjectName type_name;
@ID(0x0072) @Optional DDS::TypeObject type;
```

The `the_type` member of the `TypeObject` object shall indicate the type(s) associated with the corresponding entity. A Publication or Subscription shall be associated with one of the types of the corresponding Topic.

The beginning of section 7.6.1.3, "Type Consistency Enforcement QoS Policy, says, "It applies to `Topic`s, `DataWriter`s, and `DataReader`s." Remove `Topic`s and `DataWriter`s from that list.

In section 7.6.1.3.1, *`TypeConsistencyEnforcementQosPolicy`: Conceptual Model*, replace the bulleted list with the following:

- **`DISALLOW_TYPE_COERCION`**: The `DataWriter` and the `DataReader` must support the same data type in order for them to communicate. (This

> is the degree of type consistency enforcement required by the DDS specification [DDS] prior to this specification.)
>
> - **`ALLOW_TYPE_COERCION`**: The `DataWriter` and the `DataReader` need not support the *same* data type in order for them to communicate as long as the reader's type is assignable from the writer's type.

The same section contains the following paragraph:

> This policy has request-offer semantics [DDS]. The type compatibility kind of a `DataWriter` and a `DataReader` must be the same in order for the two to communicate.

This paragraph shall be replaced by the following:

> This policy applies only to `DataReader`s; it does not have request-offer (RxO) semantics [DDS].

The first sentence of the following paragraph (regarding default values) shall be replaced with the following:

> The default enforcement kind shall be `ALLOW_TYPE_COERCION`.

The next sentence ends with "…it shall assume a kind of `EXACT_TYPE`." This phrase shall be rewritten "…it shall assume a kind of `DISALLOW_TYPE_COERCION`."

The final sentence of the paragraph ends with "…—`EXACT_TYPE` in this case." This phrase shall be deleted.

In section 7.6.1.3.2, *Rules for Type Consistency Enforcement*, the last sentence of the first paragraph reads: "These rules are based on the type consistency kind of these entities." Replace this sentence with the following: "These rules are based on the data types of these entities and on the type consistency kind of the `DataReader`."

The next paragraph begins, "The type-consistency-enforcement rules consist of two steps." Remove the remainder of that paragraph after that first sentence.

"Step 1" in the same section consists of a bulleted list of three items. Replace the text following the words "Step 1" up through and including that list with the following:

> If both the Publication and the Subscription specify a `TypeObject`, consider it first. If the Subscription allows type coercion, then `the_type` indicated there must be assignable from `the_type` of the Publication. If the

> Subscription does not allow type coercion, then its type must be equal to the type of the Publication.

"Step 2" in the same section consists of a bulleted list of three items. Replace the text following the words "Step 2", up through and including the paragraph that ends "…shall be assumed to succeed", with the following:

> If either the Publication or the Subscription does not provide a `TypeObject` definition, then the type names are consulted. The Subscription and Publication `type_name` fields must match exactly, as in [DDS] prior to this specification.

A later paragraph in the same section begins, "Note that the `DataWriter` and the `DataReader` can each execute the algorithm independently…". In that first sentence, replace the phrase "`TypeSignature` and `TypeRepresentation`" with the word "metadata".

Immediately following are the words "The table below…" and Table 29. Remove this introductory sentence as well as the table.

Append the following new paragraph to section 7.6.1.3.3, *TypeConsistencyEnforcementQosPolicy: Platform-Specific API*:

> The subscription built-in topic data type shall indicate the type consistency requirements of the corresponding reader:
>
> ```
> @ID(0x0074) DDS::TypeConsistencyEnforcementQosPolicy
> type_consistency;
> ```

The machine-readable IDL file and *Annex D: DDS Built-in Topic Data Types* each contain a definition of the enumeration `TypeConsistencyKind`. Replace these definitions with the following:

```
@BitBound(16)
enum TypeConsistencyKind {
    DISALLOW_TYPE_COERCION,
    ALLOW_TYPE_COERCION
};
```

Also in the IDL, *remove* the member `type_consistency` from `TopicBuiltinTopicData`, `TopicQos`, `PublicationBuiltinTopicData`, and `DataWriterQos`.

Disposition: **Resolved**

Rick Warren 2/13/12 9:29 AM
Deleted: Proposed

Rick Warren 2/13/12 9:29 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 15969

Rick Warren 2/13/12 9:29 AM
Deleted: Disposition: Under Discussion

Title: Anonymous types should not be used in IDL examples and the `TypeObject` representation IDL

Source:

Remedy IT (Mr. Martin Corino, mcorino@remedy.nl)

Nature: Clarification

Severity: Significant

Summary:

IDL Anonymous type declarations should not be used in IDL examples and the "`dds_xtypes.idl`" file as this form of type declaration has been deprecated by the latest CORBA specs and will cause compatibility problems with modern CORBA IDL compilers.

Discussion:

This issue impacts not only examples and the `TypeObject`-related IDL but also other portions of the IDL. All of these places should be corrected uniformly.

Resolution:

- In the definitions of `GroupDataQosPolicy`, `TopicDataQosPolicy`, and `UserDataQosPolicy`, replace `sequence<octet>` with `ByteSeq`. This change removes occurrences of an anonymous type inherited from DDS itself.
- In the definitions of `TopicBuiltinTopicData`, `PublicationBuiltinTopicData`, and `SubscriptionBuiltinTopicData`, replace the type of the `name`, `topic_name`, and `type_name` fields (`string<256>`) with the existing `typedef ObjectName`. This change removes occurrences of an anonymous type inherited from DDS itself.
- In the definitions of `Bytes` and `KeyedBytes`, replace `sequence<octet>` with `ByteSeq`.
- Introduce two `typedef`s: "`VerbatimLanguage`" and "`VerbatimPlacement`", the first a string bounded at 32 characters and the second at 128 characters. Change the type of the member `Verbatim::language` to `VerbatimLanguage` and the type of `Verbatim::placement` to `VerbatimPlacement`.

Revised Text:

In the IDL file and in the corresponding appendices, make the changes described above. Insert the definitions of `VerbatimLanguage` and `VerbatimPlacement`

directly before the definition of `Verbatim`. (These are the only new types to be introduced.)

Disposition: **Resolved**

OMG Issue No: 15976

Title: Restrictions on MAP key element type not (clearly) documented/specified

Source:

Remedy IT (Mr. Martin Corino, mcorino@remedy.nl)

Nature: Clarification

Severity: Significant

Summary:

Although the specification of the MAP_TYPE in paragraph 7.2.2.3.4 refers to restrictions for key element type this is not formalized in either the specification for the IDL representation for maps or the C++ language mapping.

It should be clearly noted (for the IDL representation in BNF) that the key element type is restricted to integer and string types only.

The current IDL representation specs for map leave all options open for defining the key element type.

Although the specs in paragraph 7.2.2.3.4 describe the use of other element types as 'behavior undefined' this is not clear enough as the chosen C++ representation simply will not work when for instance STRUCT or UNION types are chosen as key element type.

Discussion:

[Rick] To avoid forcing implementers to define hashing and/or total ordering for all types, we only *require* vendors to support keys of string and integer types: they are trivial to order and/or hash.

At the same time, it is not desirable to *prevent* vendors from supporting keys of other types. For example, platforms like Java and .Net provide object hashing as a uniform operation, so there is no reason why vendors focusing on those platforms should not be able to leverage that capability to the benefit of their customers.

Implementations that strictly adhere to the standard should simply issue a compile error upon seeing a construct that is unspecified and unsupported. Perhaps we just need to make this clearer in section 7.2.2.3.4.

[Martin] Yes […]. I would prefer a clear explanation that supporting the use of complex types like STRUCT, UNION etc. will require non-trivial support for non-default ordering/hashing options.

Also I would prefer a clear statement like 'when an IDL compiler implementation does not support <*these kind of types as keys*> it should issue a compile error'.

BTW, rehashing this I realized that […] the default instantiation of the `less<>` template for the standard mapping of string types (`char*` and `CORBA::WChar*`)

is of little or no use for actual applications. […T]he map is actually indexed by the hashed pointer value of the strings instead of (what he will expect) by the string content.

So, I feel the spec should either state that for string keys the C++ mapping is *required* to provide a "`Compare`" specialization or put them in the same category as STRUCT, UNION etc.

Resolution:

In section 7.2.2.3.4, provide a rationale for the restriction on key types.

In the section on the Plain Language Binding for map types (7.5.1.3), clarify that the use of an unsupported key element type shall result in a compilation error. Further indicate that in C++ (section 7.5.1.3.1), the `Compare` function implementation shall consider the character contents of strings; it shall not compare pointer values.

Revised Text:

In section 7.2.2.3.4, "Collection Types", after the table, there is a bullet "Element type". The last sentence of that bullet ends, "…behavior of maps with other key element types is undefined and may not be portable." Append the following:

> (**Design rationale, non-normative**: Support for arbitrary key element types would require implementers to provide uniform sorting and/or hashing operations, which would be impractical on many platforms. In contrast, these operations have straightforward implementations for integer and string types.)

Insert the following new paragraph after the first paragraph of the introduction to section 7.5.1.3:

> As indicated in Section 7.2.2.3.4 above, implementers are only required to support keys of integer and string types. If a Type Representation compiler encounters a key type that it does not support, it shall fail with an error.

The last paragraph of section 7.5.1.3.1 currently reads, "The instantiations for the `Compare` and `Allocator` parameters are undefined and may or may not take their default values." Replace this sentence with the following:

> When a map has keys of a string type, the `Compare` function shall operate on the character contents of the strings; it shall not operate on the strings' pointer values (as `std::less` does). The instantiations for the `Compare` and `Allocator` parameters are otherwise undefined and may or may not take their default values.

Disposition: **Resolved**

OMG Issue No: 16007

Title: DDS-XTypes `@Key` annotation Issue

Source:

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

Severity: Significant

Summary:

The `@key` annotation is currently defined as:

```
@Annotation
local interface Key {
    attribute boolean value default true;
};
```

Now suppose I have the following topic type:

```
struct ATopicType {
    string a; //@Key @ID(20)
    long b;
    long c;   //@Key @ID(10)
};
```

What is the key: (a,c) or (c,a)? What if there were no `@ID` annotations?

The order of the key should be formally defined by the spec as this has an impact, among other things, on the computation of the key hash.

The rule could be as simple as taking the key element in the same order of the attribute ID. The other option is to extend the key to support a value, yet I think the first alternative is more orthogonal.

Discussion:

[Rick] The specification today deliberately does not consider member order and member ID to be the same thing. The only relationship between them is that the order is used to infer the ID in case the ID is not otherwise specified. Per the spec today, the key above is always (a, c) regardless of whether an ID annotation is there.

My understanding of this issue is that the spec currently defines key assignability of mutable types based on IDs and key type assignability, independent of order. Therefore, if the order of the key fields changes in a mutable type, the specification says the types should remain assignable. (If the type is not mutable, changing the order of any fields, keys or not, will prevent assignability.) However, because of the way the key hash is calculated, they will not in fact be interoperable, because the key hashes will change.

Therefore, we must either specify that key fields cannot be reordered, or we must define some canonical ordering of key fields to be used when calculating the key hash.

Resolution:

The key hash is currently calculated in the following way, according to the DDS-RTPS specification:

1. Serialize the key members…
    a. …in their declaration order
    b. …in big-endian CDR format
2. If the serialization is less than or equal to 128 bits, that is the key hash.
3. Otherwise, apply an MD5 hash to reduce the result to 128 bits.

Update step (1a) above such that key fields are always serialized in ascending order by member ID rather than by declaration order. Note that this change applies only when calculating the key hash, not at any other time.

This change is backwards compatible with pre-XTypes DDS implementations, because legacy type definitions will lack `@ID` annotations, and hence all members will always be in ascending ID order.

Revised Text:

Add the following new subsection at the end of section 7.6, “Use of the Type System By DDS”:

> **7.6.5 Interoperability of Keyed Topics**
>
> As described in [RTPS] section 9.6.3.3, “KeyHash (PID_KEY_HASH)”, the key hash for a given object of a keyed type is obtained by first serializing the values of the key members in their declaration order. The algorithm described in that section shall be amended such that key member values shall be serialized in the ascending orders of their member IDs.
>
> **Design rationale (non-normative)**: This change ensures that key hash values remain stable in the face of member order permutations. It is backwards compatible, because this specification interprets all pre-existing type definitions (which lack explicit member IDs) as implying member IDs in declaration order. Thus all pre-existing key hashing algorithm implementations already conform to this specification when applied to pre-existing type definitions.

In section 2.1, “Programming Interface Conformance”, in the bullet that refers to section 7.6, “Use of the Type System by DDS”, append “…excluding ‘*Interoperability of Keyed Topics*’ (Section 7.6.5)”.

In section 2.2, “Network Interoperability Conformance”, make the following changes to the bullets list of sections:

- Remove the first bullet, which ends “…up to and including Scope”. This bullet does not accurately reflect the structure of the document since the reformatting that occurred in the Beta 1 specification.
- To the bullet that refers to section 7.6, “Use of the Type System by DDS”, append the text, “…and well as ‘*Interoperability of Keyed Topics*’ (Section 7.6.5)”.

Disposition: **Resolved**

OMG Issue No: 16097

Title: Different applications in the same domain may associate the same Topic with different types

Source:

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

Summary:

Based on the DDS-XTypes specification, different applications in the same domain may associate the same Topic with different types. As a result through discovery it should be possible to find all such definitions. However this is not currently possible with the DDS v1.2 specification and the DDS-XTypes does not provide an extension for this API.

Discussion:

It is not feasible to prevent more than one type from existing for the same topic:

- Within a given version of a system, different endpoints may publish and subscribe to different types within a polymorphic inheritance hierarchy.
- As a system evolves, new components may be integrated that use updated versions of previously existing data types.

This is an issue not only within a single domain (i.e. a discovery issue), but also within a single participant in that domain (i.e. a local entity management issue). If it is legal for variants of a topic to be associated with multiple related types (it is), then more than one of those variants should be able to exist locally within the same participant. We should address the local and remote aspects of this issue at the same time.

Resolution:

Rick Warren 2/13/12 9:30 AM
Deleted: Proposed

The following resolution is related to the resolutions of issues #15702, #16007, and #16720.

Abstract Model:

A given topic is associated with one or more types. (This is a “topic” in the “virtual” system-wide sense independent of the way a topic is reflected through the API to any given component.) A given writer or reader endpoint is associated with one of the types of its topic.

If a writer and a reader share the same topic, then it is assumed that they are intended to communicate with one another. At that point, the two endpoints are evaluated to make sure that they are type-compatible—see the resolution to issue #15702.

On the Network:

The Topic, Publication, and Subscription built-in topic data types already contain a `type_name` member. In addition, these data types shall contain a list of type

definitions encapsulated in a `TypeObject` (defined in the resolution to issue #15702). In the Publication and Subscription cases, this "list" shall consist of only a single element. In the Topic case, it may consist of multiple elements.

There is a related issue that will be addressed in the context of this issue, because it will improve matching performance and prepare the specification for the possibility of future support for polymorphic readers and writers: Today, the `TypeObject` type representation is not amenable to compact representation or fast comparison: types are identified by a "type ID" that an implementation can generate any way that it wishes. This approach has two weaknesses: (1) A definition is not self-contained—it requires a full tree of the types it uses to also be present. Also, (2) two types may be logically "equal" but may not have the same binary representation, because they refer to the "same" types by different indexes. Therefore, modify the `TypeObject` representation to replace *ad hoc* type indexes with deterministic hashes. (The lower-order 64-bits of an MD5 hash of the `TypeObject` representation of the type is sufficient: a system would require 1e6 types before it would have a 1e-6 probability of a collision. A full 128-bit hash could be used, but the overhead would be significant when data sample sizes are very small.) This change will address both of the above issues. It will also allow implementations to flexibly truncate the amount of detail they put on the network. The overhead can be further reduced if we assume that most members will be of primitive type—we can define a type ID as a union of a (small) primitive type ID and a (hashed, big) complex type ID.

In the Application Code:

As always, a `Topic` object is not the real virtual topic—it is only a local proxy. We retain the constraint that a single local `Topic` object is associated with only a single type. This constraint keeps the programming model the same for both XTypes-supporting and non-XTypes-supporting implementations, and it keeps the mental model simple for the majority of applications, which will not be aware of the presence of multiple types in their topics.

To allow endpoints of the same *virtual* topic to be associated with different types, we make one change: The names of local `Topic` objects do not need to be unique within a given `DomainParticipant`, provided that all of those `Topic`s of the same name really represent different slices of the same virtual topic—in other words, they must all have the same name.

Revised Text:

Rick Warren 2/13/12 9:30 AM
Deleted: Proposed

Insert the following new subsections at the beginning of section 7.6, "Use of the Type System by DDS":

7.6.1 Topic Model

A DDS topic exists in two senses of the word:

1. **On the network**, with respect to interoperability: This is the sense in which we say that a reader and a writer share the "same" topic, even

though they obtain the topic's definition independently within their implementations.

2. **In application code**, with respect to portability: Each component that uses a topic creates or looks up a local proxy for that topic.

On the network, a given topic is associated with one or more types. A given writer or reader endpoint belongs to one topic and is associated with one of the types of that topic. If a writer and a reader share the same topic, it is assumed that they are intended to communicate with one another. At that point, the Service evaluates the two endpoints to make sure that they specify consistent types (see Section 7.6.1.3.2, "Rules for Type Consistency Enforcement") and compatible QoS (see [DDS]).

In application code, a topic is associated with a single type (as has always been the case in the [DDS] API). [*Footnote*: **Design rationale (non-normative)**: This constraint keeps the programming model the same for both XTypes-supporting and non-XTypes-supporting implementations, and it keeps the mental model simple for the majority of programmers, who will not be aware of the presence of multiple types in their topics.] Therefore, multiple API topics may correspond to (different views of) the same network topic. A given reader or writer endpoint is associated with one of them. See Section 7.6.2, "Local API Extensions", for definitions of the programming interfaces that support this polymorphism.

Insert the following after section after 7.6.2, "Endpoint Matching":

7.6.2 Local API Extensions

The following subsections define changes in behavior to existing operations defined by [DDS].

7.6.2.1 Operation: `DomainParticipant::create_topic`

As defined in [DDS], a local `Topic` object is uniquely identified by its name. In implementations conforming to *this* specification, that restriction shall be removed. The Service may instantiate multiple objects of the same name, provided that all of them represent type-based subsets of "the same" network topic; therefore, they must have consistent QoS with one another.

7.6.2.2 Operation: `DomainParticipant::lookup_topicdescription`

As defined in [DDS], a local `TopicDescription` object is uniquely identified by its name. In implementations conforming to *this* specification, that restriction shall be removed. This operation shall return one of the local `TopicDescription` objects of the given name; which one is unspecified.

Next, update the definition of `TypeObject`*:*

The first paragraph of section 7.3.4.1 ends, "…(*b*) (optionally) identifies a single type within that library." Change this to, "…(*b*) identifies some number of types within that library."

In section 7.3.4.1.1, "References Among Types", replace the first paragraph with the following:

> Rather than refer to one another by name, as in some other Type Representations (such as IDL), types within this Type Representation refer to one another by a "type ID" for the sake of compactness. The representation of the type ID depends on whether the type is primitive or constructed—it is a union. To save space, primitive types are identified for a small integral ID. Constructed types are identified by a hash; see Section 7.3.4.1.1, "Type Hierarchy", below.

In section 7.3.4.1.1, "Type Hierarchy", add the following new paragraph immediately after the bulleted list:

> The type ID of a constructed type shall be calculated in the following way.
>
> 1. Serialize the type (as an `ArrayType`, `StructureType`, etc. as appropriate) in big-endian CDR Data Representation. Note that this step is recursive, as the serialization may require calculating the IDs of types used by this type—for example, to type structure members.
> 2. Apply the MD5 hash algorithm to that serialized representation.
> 3. The type ID is the less-significant 64 bits of the hash, represented as an unsigned 64-bit integer. [*Footnote*: **Design rationale (non-normative)**: The entire 128 bits could have been used. However, two factors argue in favor of a 64-bit hash: (*1*) It reduces the size of the `TypeObject` by approximately eight bytes per type member, decreasing network overhead and speeding the discovery process. (*2*) The availability of a 64-bit integer type makes dealing with data of this size simple and fast. Note that 64 bits provide an extremely small chance of collision, even in a system with many thousands of types.]

Section 7.3.4.2, "Primitive Types", ends with the sentence, "The assigned ID values for constructed types must not overlap with these predefined values." Remove this sentence; it is unnecessary now.

In the IDL file and in "Annex B: Representing Types with `TypeObject`", the definition of the `TypeId` type is given as a `typedef` followed by a series of constants corresponding to the primitive types. Replace this `typedef` and these constants with the following:

```
typedef short PrimitiveTypeId;

const PrimitiveTypeId NO_TYPE_ID         = NO_TYPE;
const PrimitiveTypeId BOOLEAN_TYPE_ID    = BOOLEAN_TYPE;
const PrimitiveTypeId BYTE_TYPE_ID       = BYTE_TYPE;
const PrimitiveTypeId INT_16_TYPE_ID     = INT_16_TYPE;
const PrimitiveTypeId UINT_16_TYPE_ID    = UINT_16_TYPE;
const PrimitiveTypeId INT_32_TYPE_ID     = INT_32_TYPE;
const PrimitiveTypeId UINT_32_TYPE_ID    = UINT_32_TYPE;
const PrimitiveTypeId INT_64_TYPE_ID     = INT_64_TYPE;
const PrimitiveTypeId UINT_64_TYPE_ID    = UINT_64_TYPE;
const PrimitiveTypeId FLOAT_32_TYPE_ID   = FLOAT_32_TYPE;
const PrimitiveTypeId FLOAT_64_TYPE_ID   = FLOAT_64_TYPE;
const PrimitiveTypeId FLOAT_128_TYPE_ID = FLOAT_128_TYPE;
const PrimitiveTypeId CHAR_8_TYPE_ID     = CHAR_8_TYPE;
const PrimitiveTypeId CHAR_32_TYPE_ID    = CHAR_32_TYPE;

union _TypeId switch (TypeKind) {
    case BOOLEAN_TYPE:
    case BYTE_TYPE:
    case INT_16_TYPE:
    case UINT_16_TYPE:
    case INT_32_TYPE:
    case UINT_32_TYPE:
    case INT_64_TYPE:
    case UINT_64_TYPE:
    case FLOAT_32_TYPE:
    case FLOAT_64_TYPE:
    case FLOAT_128_TYPE:
    case CHAR_8_TYPE:
    case CHAR_32_TYPE:
        PrimitiveTypeId primitive_type_id;
    default:
        unsigned long long constructed_type_id;
};

typedef sequence<_TypeId> TypeIdSeq;
```

The `TypeObject` type contains a member `the_type`, defined like this:

```
@Optional _TypeId the_type;
```

Change the definition of this member like this:

```
TypeIdSeq the_type;
```

Disposition: **Resolved**

Rick Warren 2/13/12 9:30 AM
Deleted: Proposed

Rick Warren 2/13/12 9:30 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 16236

Title: Annex D ("DDS Built-in Topic Data Types") is out of sync with IDL file

Rick Warren 2/13/12 9:30 AM
Deleted: Disposition: Under Discussion

Source:

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

Severity: Support Text

Summary:

The `equivalent_type_name` and `base_type_name` fields from `TopicBuiltinTopicData`, `PublicationBuiltinTopicData`, and `SubscriptionBuiltinTopicData` are described in the prose of the specification and are reflected in the IDL file. However, they missing from the annex; they need to be added.

Discussion:

Any annex that contains a copy of a machine-readable file should be considered a convenience for the reader, so that s/he can see everything in a single file. We should fix this issue, and avoid future similar issues, by simply replacing the annex content in bulk with the analogous content from the appropriate machine-readable file.

This issue resolution should be implemented and voted on in the last round before the FTF report is submitted to ensure that it contains all relevant changes.

Resolution:

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

- Replace all literal-formatted text in Annex A (beginning with "`<?xml...`") with the contents of the XSD file accompanying the specification.
- Replace all literal-formatted text in Annex B (beginning with "`module DDS`") with the corresponding content from the IDL file accompanying the specification.
- Replace all literal-formatted text in Annex C (beginning with "`module DDS`") with the corresponding content from the IDL file accompanying the specification.
- Replace all literal-formatted text in Annex D (beginning with "`module DDS`") with the corresponding content from the IDL file accompanying the specification.
- Replace all literal-formatted text in Annex E (beginning with "`module DDS`") with the corresponding content from the IDL file accompanying the specification.

- Replace all literal-formatted text in Annex F (beginning with "`module DDS`") with the corresponding content from the IDL file accompanying the specification.

Revised Text:

See the description under "Resolution" above.

Disposition: **Resolved**

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 16237

Title: No way to get or set length of collection-typed `DynamicData` objects

Source:

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

Brookhaven National Laboratory (Dr. Nikolay Malitsky, malitsky@bnl.gov)

Severity: Minor

Summary:

A `DynamicData` object may encapsulate the state of an object of a collection type—a sequence, array, string, or map. In such a case, it is necessary to be able to get and set the length of the collection.

Discussion:

We need to add an accessor for the current length. This accessor can also be useful for other object of other types; for example, it can return the number of members in a structure.

However, it is not appropriate to add a mutator for the length directly:

- If the length of a collection is increased, that implies the addition of new elements. But the values of these new elements will not be meaningful.
- If the length of a collection is decreased, that implies the removal of existing elements. But that may not be meaningful (what does it mean to remove a non-optional member of a structure?) of useful (why would you want to clear the fifth element of a bit set just because of what its index is?).
- The lengths of some types cannot legally be changed at all—for example, array types.

Therefore, it is better to allow elements to be added and removed, where appropriate, and thereby modify the length implicitly.

Resolution:

Add an operation `get_item_count` to the `DynamicData` class.

- *If the object is of a collection type*, return the number of elements currently in the collection. In the case of an array type, this value will always be equal to the product of the bounds of all array dimensions.
- *If the object is of a bit set type*, return the number of named flags that are currently set in the bit set.
- *If the object is of a structure or annotation type*, return the number of members in the object. This value may be different than the number of members in the corresponding `DynamicType`—for example, some

Rick Warren 2/13/12 9:31 AM
Deleted: Disposition: Under Discussion

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

optional members may be omitted; see the existing `DynamicData` documentation.

- *If the object is of a union type*, return the number of members in the object. This value will always be two: the discriminator and the current member corresponding to it.
- *If the object is of a primitive or enumeration type*, it is atomic: return one.
- *If the object is of an alias type*, return the value appropriate for the alias's base type.

Expand the contracts of the existing `set_<type>_value` operations to allow them to append new elements to resizable collections—strings, sequences, and maps. (This behavior is aligned with the existing contract of the `clear` operations, which states that cleared elements of resizable collections shall be removed.) Obtain the member ID associated with the new value as follows:

- For a string or sequence type, use `get_member_id_at_index` to obtain an ID for the index one greater than the current length.
- For a map type, use `get_member_id_by_name` to obtain an ID for the new map key.

Revised Text:

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Add a row for the new `get_item_count` operation in Table 27, *`DynamicData` properties and operations*, in section 7.5.2.10, *`DynamicData`*. The new row follows `get_member_id_at_index` and precedes `equals`.

`get_item_count`		`UInt32`

Add a new section 7.5.2.10.6, *Operation: `get_item_count`*, after the existing section 7.5.2.10.5, *Operation: `equals`*:

7.5.2.10.6 Operation: `get_item_count`

The "item count" of the data depends on the type of the object.

- *If the object is of a collection type*, return the number of elements currently in the collection. In the case of an array type, this value will always be equal to the product of the bounds of all array dimensions.
- *If the object is of a bit set type*, return the number of named flags that are currently set in the bit set.
- *If the object is of a structure or annotation type*, return the number of members in the object. This value may be different than the number of members in the corresponding `DynamicType`—for example, some optional members may be omitted.

- *If the object is of a union type*, return the number of members in the object. This value will always be two: the discriminator and the current member corresponding to it.
- *If the object is of a primitive or enumeration type*, it is atomic: return one.
- *If the object is of an alias type*, return the value appropriate for the alias's base type.

In what is now section 7.5.2.10.9, but which was previously 7.5.2.10.8, *Platform-Specific Model: IDL*, there is a bulleted list of name expansions. Immediately following this list is a paragraph that begins, "As mentioned above, it shall be possible to implicitly promote integral types." Before this paragraph, insert the following new paragraph:

If a `DynamicData` object represents an object of a resizable collection type (string, sequence, or map), these setters may also be used to append new elements to the collection.

- For a string or sequence type, use `get_member_id_at_index` to obtain an ID for the index one greater than the current length.
- For a map type, use `get_member_id_by_name` to obtain an ID for the new map key.

Add the new `get_item_count` operation to the IDL file as well.

Disposition: **Resolved**

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 16239

Title: Typographical errors

Source:

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

Severity: Minor

Summary:

The specification contains the following identified typographical errors:

- The third bullet in section 2.2, “Network Interoperability Conformance”, which begins with “Data Representation…”, is lacking an opening parenthesis at the start of the second sentence. The closing parenthesis is already present.
- The first bullet in the second bulleted list in the `STRUCTURE_TYPE` row of the table in section 7.2.4.5, *Aggregation Types*, begins, “Any members in T1 and T2 that have the same a name….” The word “a” before “name” is extraneous and should be removed.
- The last sentence of the first paragraph of section 7.3.4.4, “Aggregated Types”, says, “…such as the members name and type…”. The word “members” should have an apostrophe: “member’s”.

Resolution:

See the trivial corrections described above.

Revised Text:

See the trivial corrections described above.

Disposition: **Resolved**

Rick Warren 2/13/12 9:31 AM
Deleted: Disposition: Under Discussion

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Deleted: Proposed

Rick Warren 2/13/12 9:31 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 16240

Rick Warren 2/13/12 9:31 AM
Deleted: Disposition: . Under Discussion .

Title: Changing a `DynamicType` object is ambiguous

Source:

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

Brookhaven National Laboratory (Dr. Nikolay Malitsky, malitsky@bnl.gov)

Severity: Significant

Summary:

The current `DynamicType` API allows a type definition to be changed. Such a change can be ambiguous at best and dangerous at worst.

- Suppose that the type already types (`DynamicData`) objects. Should all of those objects refer to the old version of the type or the new version? If the new version, how shall we address the situation where the new type is inconsistent with the existing data contents?
- Suppose that the type has already been registered with some domain participant. Does the registered type name refer to the old version of the type or the new version? What shall be done about samples that may already have been published or received using the old version?
- Can `DynamicTypeFactory` cache and reuse identical types that are requested more than once? Ideally yes, but the ability to change a type makes the state management necessary to support this behavior very complicated.

It would be safer, clearer, and more efficient if `DynamicType` objects were immutable once created. That implies that either the factory must take sufficient inputs to fully create a type in one step, or we must employ a builder or similar pattern.

Discussion:

A builder pattern would give the application more flexibility in how to create new types. It would also nicely encompass the current `load_type` operations as well as hypothetical PSM-specific operations, such the creation of a `DynamicType` from a Java `Class` object.

Resolution:

Define a new class `DynamicTypeBuilder` as a copy of the current definition of `DynamicType`. Remove from `DynamicType` all operations that modify an object, and add to `DynamicTypeBuilder` an operation build that returns a corresponding (immutable) `DynamicType`.

Update the existing `DynamicTypeFactory` to create instances of `DynamicTypeBuilder` instead of `DynamicType`. Rename this class `DynamicTypeBuilderFactory` accordingly.

Replace the existing operation `DynamicType::clone` with a corresponding create operation in `DynamicTypeBuilderFactory`.

Revised Text:

Replace all occurrences of the class name `DynamicTypeFactory` with `DynamicTypeBuilderFactory`.

At the beginning of section 7.5.2.2, "`DynamicTypeBuilderFactory`" (formerly "`DynamicTypeFactory`"), is a sentence that begins, "Its 'only' instance is the starting point…." Replace that sentence with the following:

> Its "only" instance is the starting point for creating and deleting `DynamicTypeBuilder` objects.

In the table of operations that follows, replace every incidence of return type `DynamicType` with `DynamicTypeBuilder`—except for the return type of `get_primitive_type`.

In the table and in the subsections that follow, rename some of the methods as follows:

- `create_type_from_type_object` → `create_type_w_type_object`
- `load_type_from_url` → `create_type_w_uri`
- `load_type_from_document` → `create_type_w_document`

In addition, add the following new rows in the table immediately after the `create_type` rows:

`create_type_copy`		`DynamicTypeBuilder`
	`type`	`DynamicType`

Add the corresponding new subsections as follows, immediately after section 7.5.2.2.6, "Operation: `create_type`":

> **7.5.2.2.7 Operation: `create_type_copy`**
>
> Create and return a new `DynamicTypeBuilder` object with a copy of the state of the given type. All objects returned by this operation should eventually be deleted by calling `delete_type`.

Parameter `type` - The initial state of the new type to create. If this argument is nil, this operation shall fail and return a nil value.

In all of the subsections of section 7.5.2.2, "`DynamicTypeBuilderFactory`" (formerly "`DynamicTypeFactory`"), corresponding to the operations on that class, update references to creating `DynamicType` objects to refer to `DynamicTypeBuilder` instead.

In section 7.5.2.6, `DynamicTypeMember`, remove the operation `apply_annotation`, both from the table of properties and operations as well as from the subsections. Delete the phrase "…with `apply_annotation`" from the end of section 7.5.2.6.1, "Property: `annotation`".

In section 7.5.2.8, `DynamicType`, remove the operations `add_member`, `apply_annotation`, and `clone` from the UML diagram (Figure 36, "Dynamic Type"), the table of properties and operations, and the subsections. Also in the UML diagram, replace `DynamicTypeFactory` with `DynamicTypeBuilder`. Delete the phrase "…with `apply_annotation`" from the end of section 7.5.2.8.2, "Property: `annotation`".

In the subsection "Property: `member_by_name`", the second sentence begins, "As described in the case of `add_member`, not only…." Replace this with "As described in the table below, not only…." Move Table 25, "`DynamicType::add_member` behavior", to immediately after this paragraph from its current location in the to-be-removed section "Operation: `add_member`" and replace "`add_member`" in the caption with "`member_by_name`".

Immediately after section 7.5.2.8, `DynamicType`, add the following new section. (These contents are based on the existing contents of section 7.5.2.8, "`DynamicType`".)

7.5.2.9 `DynamicTypeBuilder`

A `DynamicTypeBuilder` object represents a transitional state of a particular type defined according to the Type System. It is used to instantiate concrete `DynamicType` objects.

Table 25 - `DynamicTypeBuilder` properties and operations

`DynamicTypeBuilder`	
Properties	
`descriptor`	`read-only TypeDescriptor`

member_by_name	read-only string<Char8,256> → DynamicTypeMember [0..1]	
member	read-only MemberId → DynamicTypeMember [0..1]	
annotation	read-only AnnotationDescriptor [*]	
Operations		
equals		Boolean
	other	DynamicTypeBuilder
get_name		String<Char8,256>
get_kind		TypeKind
add_member		ReturnCode_t
	descriptor	MemberDescriptor
apply_annotation		ReturnCode_t
	descriptor	AnnotationDescriptor
apply_annotation_to_member		ReturnCode_t
	member_id	MemberId
	descriptor	AnnotationDescriptor
build		DynamicType

7.5.2.9.1 Operation: `add_member`

Add a "member" to this type, where the new "member" has the meaning defined in the specification of the `DynamicTypeMember` class. Specifically, the behavior shall be as described in the table in Section 7.5.2.8.6, "*Property: member_by_name*". For type kinds not given in that table, this operation shall fail with `RETCODE_PRECONDITION_NOT_MET`.

Following a successful return, the new member shall appear in the member property and possibly in the `member_by_id` property, based on the definition of that property.

Parameter `descriptor` - A descriptor of the new member to be added. If this argument is nil, the operation shall fail with `RETCODE_BAD_PARAMETER`.

7.5.2.9.2 Property: `annotation`

This property provides all annotations that have previously been applied to this type with `apply_annotation`.

7.5.2.9.3 Operation: `apply_annotation`

Apply the given annotation to this type. It shall subsequently appear in the `annotation` property.

Parameter `descriptor` - A consistent descriptor for the annotation to apply. If this argument is not consistent, the operation shall fail with `RETCODE_BAD_PARAMETER`.

7.5.2.9.4 Operation: `apply_annotation_to_member`

Apply the given annotation to this member. It shall subsequently appear in the `annotation` property of the identified member.

Parameter `member_id` - Identifies the member to which the annotation shall be applied.

Parameter `descriptor` - A consistent descriptor for the annotation to apply. If this argument is not consistent, the operation shall fail with `RETCODE_BAD_PARAMETER`.

7.5.2.9.5 Operation: `build`

Create an immutable `DynamicType` object containing a snapshot of this builder's current state. Subsequent changes to this builder, if any, shall have no observable effect on the states of any previously created `DynamicType`s.

7.5.2.9.6 Property: `descriptor`

This property provides a summary of the state of this type.

7.5.2.9.7 Operation: `equals`

Two types shall be considered equal if and only if all of their respective properties, as identified in the table above, are equal.

7.5.2.9.8 Operation: `get_kind`

This convenience operation indicates the kind of this type (e.g., integer, structure, etc.). Its result shall be the same as the kind indicated by the type's descriptor property.

7.5.2.9.9 Operation: `get_name`

This convenience operation provides the fully qualified name of this type. It shall be identical to the name string that is a member of the descriptor property.

7.5.2.9.10 Property: `member_by_name`

This property contains a mapping from the name of a member of this type to the member itself. As described in the case of `add_member`, not only members of aggregated types are considered "members" here: the constituents of enumerations, bit sets, and other kinds of types are also considered to be "members" for the purposes of this property.

The lifecycle of a `DynamicTypeMember` object is governed by that of the `DynamicTypeBuilder` that contains it. The former shall be considered to exist logically from the time the corresponding member is added to the latter and until such time as the latter is deleted. Implementations may allocate and de-allocate `DynamicTypeMember` objects more frequently, provided that:

- Users of the `DynamicTypeMember` class are not required to explicitly delete objects of that class.
- Changes to one `DynamicTypeMember` object representing a given member shall be reflected in all observable `DynamicTypeMember` objects representing the same member.
- All `DynamicTypeMember` objects representing the same member shall compare as equal according to their equals operations.

7.5.2.9.11 Property: `member`

This property contains a mapping from the member ID of a member of this (aggregated) type to the member itself.

- If this type is an aggregated type, the collection of members available through this property shall be equal to (element order notwithstanding) that available through the `member_by_name` property.
- If this type is not an aggregated type, the collection of members available through this property shall be empty.

Implement all of the above changes described above in the IDL file as well.

Disposition: **Resolved**

OMG Issue No: 16241

Title: Difficult to apply automation to statically defined types

Source:

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

Severity: Minor

Summary:

Application code (i.e. business logic) generally depends statically on particular types and their members. Therefore, it is appropriate to define these types statically (e.g. in IDL) and generate code for them in order to take advantage of static type safety and improved performance.

In contrast, infrastructure code (i.e. logic that is independent of particular applications) generally must not depend on application-specific types, because such dependencies prevent that code from being reused.

These two kinds of code can exist within a single component. For example:

- The properties of application-specific types might be checked against a common set of rules to ensure that they meet organizational or program-specific guidelines.
- Certain design patterns may call for arbitrary application-specific types to be augmented in uniform ways.
- Type-specific application activities, such as reading and/or writing data, might be logged or recorded.

The straightforward way to handle these scenarios would be to:

1. Convert a generated type definition into a `DynamicType` object.
2. Convert an object of a generated type into a `DynamicData` object.
3. Pass these two dynamic objects to the infrastructure code and allow it to operate on them.

Unfortunately, the specification does not provide such a capability. Instead, applications have to manually construct `DynamicType` and `DynamicData` objects member by member. This code is not only verbose but also brittle, because if the static type definition changes (e.g. during development or across versions), it can grow out of sync with the hand-written code that manipulates that definition.

Discussion:

We need three new conversions:

1. `TypeSupport` → `DynamicType`
2. `DynamicData` → sample object

3. Sample object → `DynamicData`

These operations could either be added to the `TypeSupport` interface or to a "type builder" interface. The former approach is preferred, because the generic type parameter of the `TypeSupport` in the C++ and Java PSMs allows a degree of type safety that way.

Resolution:

Introduce three new `TypeSupport` operations:

1. `get_type`: Get a `DynamicType` object corresponding to the `TypeSupport`'s data type.
2. `create_sample`: Create a sample of the `TypeSupport`'s data type from an input `DynamicData` object.
3. `create_dynamic_sample`: Create a `DynamicData` object from an input sample of the `TypeSupport`'s data type.

Revised Text:

Demote the existing section 7.6.4.1, *`DynamicTypeSupport`*, to a subsection 7.6.4.1.3 and insert the following above it.

7.6.4.1 Type Support

Application code (i.e. business logic) generally depends statically on particular types and their members. In contrast, infrastructure code (i.e. logic that is independent of particular applications) generally must not depend on application-specific types, because such dependencies prevent that code from being reused. These two kinds of code can exist within a single component.

Therefore, it is desirable to allow conversions among static and dynamic bindings for the same types and samples. These conversions shall be provided by operations on the generic `TypeSupport` interface and its extended interfaces.

7.6.4.1.1 `TypeSupport` Interface

The following operations shall be added to the `TypeSupport` interface defined by [DDS]. (The operations on this interface already defined in [DDS] are unchanged.)

Table 31—New `TypeSupport` operations

Operations		
`get_type`		`DynamicType`

7.6.4.1.1.1 Operation: `get_type`

Get a `DynamicType` object corresponding to the `TypeSupport`'s data type.

7.6.4.1.2 `FooTypeSupport` Interface

The following operations shall be added to the `FooTypeSupport` interface defined by [DDS]. (The operations on this interface already defined in [DDS] are unchanged.)

Table 32—New `FooTypeSupport` operations

Operations		
`create_sample`		`Foo`
	`src`	`DynamicData`
`create_dynamic_sample`		`DynamicData`
	`src`	`Foo`

7.6.4.1.2.1 Operation: `create_sample`

Create a sample of the `TypeSupport`'s data type with the contents of an input `DynamicData` object.

Parameter `src` – The source object whose contents are to be reflected in the resulting object. This method shall fail with a nil return result if this object is nil or if the `DynamicType` of this object is not compatible with the `TypeSupport`'s data type.

7.6.4.1.2.2 Operation: `create_dynamic_sample`

Create a `DynamicData` object with the contents of an input sample of the `TypeSupport`'s data type.

Parameter `src` – The source object whose contents are to be reflected in the resulting object. This method shall fail with a nil return result if this object is nil.

Make the following changes to section 7.6.4.1.3 (formerly 7.6.4.1), *`DynamicTypeSupport`*:

- The first paragraph ends with the sentence: "In addition, it provides access to the type from which it was created." Remove this sentence.
- Remove the Properties section, including the "type" property, from the table of properties and operations.

- Remove the section 7.6.4.1.3.1 (formerly 7.6.4.1.1), *Property:* `type`.

Add the new `TypeSupport` and `FooTypeSupport` operations to the IDL file. Insert the following immediately before `DynamicTypeSupport`:

```
interface TypeSupport {
//  ReturnCode_t register_type(
//      in DomainParticipant domain,
//      in string type_name);
//  string get_type_name();

//  DynamicType get_type();
};

/* Implied IDL for type "Foo":
interface FooTypeSupport : DDS::TypeSupport {
    DDS::ReturnCode_t register_type(
        in DDS::DomainParticipant participant,
        in string type_name);
    string get_type_name();

    DynamicType get_type();

    Foo create_sample(in DynamicData src);
    DynamicData create_dynamic_sample(in Foo src);
};
*/
```

Also, declare that `DynamicTypeSupport` extends `TypeSupport`, and consequently remove the "`local`" qualifier.

Disposition: **Resolved**

OMG Issue No: 16242

Title: New QoS policies don't indicate whether they can be changed

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Minor

Summary:

Each DDS QoS policy must define a couple of attributes: whether or not the policy has request-offer (RxO) semantics and whether or not it can be changed after an entity is enabled. The two new QoS policies defined by DDS-XTypes—`TypeConsistencyEnforcementQosPolicy` and `DataRepresentationQosPolicy`—define the first part but not the second.

Discussion:

Because of their potential interactions with historical data, neither of these policies should be changeable after an entity is enabled.

Resolution:

State in the introduction to each policy that its value cannot be changed after an entity is enabled.

Revised Text:

In section 7.6.1.1.1, *`DataRepresentationQosPolicy`: Conceptual Model*, the following sentence precedes the final bulleted list: "This policy has request-offer semantics [DDS]." Replace this sentence with the following:

> This policy has request-offer semantics, and its value cannot be changed after the entity in question has been enabled [DDS].

In section 7.6.1.3.1, *`TypeConsistencyEnforcementQosPolicy`: Conceptual Model*, the second paragraph after the bulleted list begins, "This policy has request-offer semantics [DDS]." Append the following sentence to this paragraph:

> The value of this policy cannot be changed after the entity in question has been enabled.

(The resolution of issue #15702 changes the contents of this paragraph. However, the resolution to *this* issue is the same in either case: append the new sentence to whatever is there.)

Disposition: **Resolved**

OMG Issue No: 16243

Title: `Optional`, `must_understand` **members of non-mutable aggregation types cannot be accurately represented in CDR**

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Significant

Summary:

Members of a structure or union type have a small number of metadata attributes, among them:

- `optional`: indicates that the member may or may not take a value in a particular sample
- `must_understand`: indicates that a consumer of the data must discard the sample if it encounters such a field in it

The type-level assignability rules detect mismatches respecting non-`optional` `must_understand` members. (Such members will always take a value, and therefore there is no possibility of data exchange between the producer and consumer.) However, `must_understand` members that are `optional` must be detected on a sample-by-sample basis: any sample that does not contain the offending member can be exchanged, but those that do cannot.

The XML Data Representation allows the encoding of the `must_understand` attribute within each sample, as does the parameterized encapsulation variants of the Extended CDR Data Representation. However, the traditional, non-parameterized encapsulation variants—used by final and extensible structures and unions—do not permit this attribute to be expressed. As a result, consumers will not be able to detect when it should have been present, and may therefore present incorrect data to applications.

Discussion:

The traditional CDR encapsulations prefix each optional member with a 32-bit "header" that indicates the presence and size of the member. In contrast, the parameterized CDR encapsulations prefix each member with a 32-bit parameter header that indicates the presence, size, ID, and metadata flags of the member, including `must_understand`. Therefore, we can reuse the parameter header structure we already have to solve this problem, instead of using just a 32-bit size, with no increase in overhead.

Resolution:

Rick Warren 2/13/12 9:32 AM
Deleted: Proposed

Update section 7.4.1.1.5.2, "Optional Members." Instead of preceding each optional member with a 32-bit size, use a parameter header. Note that either the

four-byte header defined by DDS-RTPS or the 12-byte extended header defined by DDS-XTypes may be used.

Revised Text:

Rick Warren 2/13/12 9:32 AM
Deleted: Proposed

Replace the contents of section 7.4.1.1.5.2, *Optional Members*, with the following:

> Structure members marked as optional shall be preceded by a parameter header as described in Section 7.4.1.2, "*Parameterized CDR Representation*", below.

Replace the introductory paragraphs of section 7.4.1.2, *Parameterized CDR Representation*, which precedes the beginning of section 7.4.1.2.1, with the following:

> The parameterized CDR representation is based on the RTPS Parameter List CDR data representation defined in [RTPS].
>
> Each element, or parameter, within a parameter list data structure is simply a CDR-encapsulated block of data. Preceding each one is a parameter header consisting of a two-byte parameter ID followed by a two-byte parameter length. One parameter follows another until a list-terminating sentinel is reached.
>
> This data representation uses elements of the parameter list data structure for two purposes:
>
> - Any object of a mutable aggregated type shall be serialized as a parameter list. Each of its members shall correspond to a single parameter within that list.
> - Any optional member of a final or extensible structure shall be preceded by a parameter header describing that member. If the member takes no value within that particular object, the data length indicated by the header shall be zero. This reuse of the parameter header data structure does not constitute a complete parameter list: the optional member shall not be followed by list-terminating sentinel.

Following Table 27, *Reserved parameter ID values*, are two paragraphs describing the use of `PID_EXTENDED`. Add the following as a third paragraph:

> These extended parameter headers, based on `PID_EXTENDED`, shall be legal within the parameter list data structures used to serialize objects of mutable aggregated types. They shall also be legal when preceding optional members of final or extensible structures, as described above.

In section 7.4.1.2.3, *Omission and Reordering of Members of Aggregated Types*, the first paragraph pertains to structures. Precede each occurrence of “structure” or “structures” in this paragraph with the word “mutable”. Then insert the following new paragraph immediately afterwards:

> Objects of final or extensible structures are not serialized as full parameter lists, even if some members are optional. Therefore, the members of these types may not be omitted or reordered.

Disposition: **Resolved**

Rick Warren 2/13/12 9:32 AM
Deleted: Proposed

Rick Warren 2/13/12 9:32 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

Rick Warren 2/13/12 9:32 AM
Deleted: Disposition: Under Discussion

OMG Issue No: 16271

Title: Incorrect default extensibility kind value in XSD file

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Support Text

Summary:

The declaration of the complex type `structOrUnionTypeDecl` in the XSD file contains the following attribute declaration:

```
<xs:attribute name="extensibility"
              type="extensibilityKind"
              use="optional"
              default="false"/>
```

However, “false” is not a valid value for the `extensibilityKind` type.

Resolution:

Change the default value to “`extensible`”.

Revised Text:

Replace the following attribute declaration within the complex type `structOrUnionTypeDecl`:

```
<xs:attribute name="extensibility"
              type="extensibilityKind"
              use="optional"
              default="false"/>
```

…with this one:

```
<xs:attribute name="extensibility"
              type="extensibilityKind"
              use="optional"
              default="extensible"/>
```

Disposition: **Resolved**

OMG Issue No: 16364

Title: Ambiguous description of serializing discovery types

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Minor

Summary:

The definition of `PID_LIST_END` in Table 27 in section 7.4.1.2.1, *Interpretation of Parameter ID Values*, reads:

> RTPS specifies that the PID value 1 shall be used to terminate parameter lists within the DDS built-in topic data types. Rather than reserving this parameter ID for all types, thereby complicating the member ID-to-parameter ID mapping rules for all producers and consumers of this Data Representation, Simple Discovery types only shall be subject to a special case: member ID 1 shall not be used, and either parameter ID 0x3F02 or parameter ID 1 shall terminate the parameter list.

The meaning of "Simple Discovery types" is not clear. It should not apply to all mutable types that might ever be used as part of the Simple Discovery process. It is intended only to maintain backward compatibility with types defined in DDS-RTPS.

Resolution:

See the Revised Text below.

Rick Warren 2/13/12 10:30 AM
Deleted: Proposed

Revised Text:

Append an additional sentence to the definition of `PID_LIST_END` in Table 27 in section 7.4.1.2.1, *Interpretation of Parameter ID Values*:

"These types consist of the built-in topic data types, and those other types that contain them as members, as defined by [RTPS]."

Disposition: **Resolved**

OMG Issue No: 16365

Title: Inconsistent bit set/integer equivalency

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Significant

Summary:

A bit set with bit bound <= 8 is mapped to a byte in some places (page 78 and 40) and to an unsigned short in others (page 88). (Page numbers refer to beta 2.)

Resolution:

A bit set with bit bound <= 8 should be serialized as a single byte.

Revised Text:

In the table in section 7.5.1.2.2, *BitSet Types*, replace these rows:

Bit Set Bound	***Unsigned Integer Equivalent***
1–16	`unsigned short`

…with the following:

Bit Set Bound	***Primitive Equivalent***
1–8	`octet`
9–16	`unsigned short`

Disposition: **Resolved**

OMG Issue No: 16366

Title: Incorrect union assignability rules

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Significant

Summary:

The type assignability rules for unions have a number of problems (see the `UNION_TYPE` row of the table in section 7.2.4.5, *Aggregation Types*):

- The second and third bullets in the list of rules are redundant.
- According to the fourth bullet (assignability of default members), a union with a default label may not even be assignable to itself.
- The rules do not account for member IDs and names, which can be significant when deserializing objects in some data representations.
- The rules do not account for the fact that the discriminator may or may not be a key.
- The rules do not account for extensibility kind.
- The rules are under-specified with respect to malformed samples. Specifically, the discriminator may select a particular member *X* of the union, but the implementation may discover in its place a different member *Y*. (This situation will be detectable only with certain data representations.)

Discussion:

See also the following issues, which also deal with assignability.

Rick Warren 2/13/12 10:30 AM
Deleted: At this time, their proposed resolutions do not overlap.

- Issue #16367 addresses the concept of “strong assignability”.
- Issue #16368 proposes making object-level assignability rules more flexible.

Resolution:

Apply the following rules:

- Apply the member ID, name, and type matching rules for structures to unions as well.
- Every discriminator value in T2 must select the “same” member in T1—or no member at all. This can be checked in three steps:
    1. Every discriminator value in a non-default label of T2 selects the same member in T1 or none at all.
    2. Every discriminator value in a non-default label of T1 selects the same member in T2 or none at all.

3. If both T1 and T2 have a default label, the corresponding members must be assignable.

- Either both types must have a key or neither.
- The extensibility kinds must match.
- Specify explicitly that an object of a union with a mismatch between its discriminator and non-discriminator members is considered malformed. If an implementation is able to detect this case, it shall consider the object unassignable. If not, then the impact is unspecified.

Revised Text:

Replace the center column of the `UNION_TYPE` row of the table in section 7.2.4.5, *Aggregation Types*, with the following:

> UNION_TYPE if and only if it is possible to unambiguously identify the appropriate T1 member based on the T2 discriminator value and to transform both the discriminator and the other member correctly. Specifically:
>
> - T1.discriminator.id == T2.discriminator.id and T1.discriminator.type *is-assignable-from* T2.discriminator.type.
> - Either the discriminators of both T1 and T2 are keys or neither are keys.
> - T1.extensibility == T2.extensibility.
> - Any members in T1 and T2 that have the same name also have the same ID and any members with the same ID also have the same name.
> - For each member “m1” in T1, if there is a member m2 in T2 with the same member ID then m1.type *is-assignable-from* m2.type if T1 is mutable or strongly assignable if T1 is final or extensible.
> - A discriminator value appearing in a non-default label of T2 selects a member m2. If the same discriminator value selects a member m1 of T1, then m1.id == m2.id.
> - A discriminator value appearing in a non-default label of T1 selects a member m1. If the same discriminator value selects a member m2 of T2, then m1.id == m2.id.
> - If both T1 and T2 have a default label, then the IDs of the members selected by those labels must be equal.
>
> AND if T1 is final, the number of members in T1 is equal to the number of members in T2.

Remove the footnote that is linked to the old contents of that table cell.

Append the following paragraphs to the right-hand column of the same table row:

If either member of the union is unassignable, then the T2 object is unassignable to T1.

If the discriminator value of a union object and its non-discriminator member do not agree with one another, the object is considered malformed. The implementation may or may not be able to detect this error. If it can, it shall consider the object unassignable. If it cannot, the behavior is unspecified.

Disposition: **Resolved**

OMG Issue No: 16367

Title: Definition of "strongly assignable" is incomplete

Source:

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

Severity: Significant

Summary:

The definition of the term "strongly assignable" in section 7.2.4, *Type Compatibility: "is-assignable-from" relationship*, states:

> If types T1 and T2 are both mutable and T1 is-assignable-from T2, then T1 is said to be "strongly" assignable from T2.

The purpose of this concept is to distinguish cases where it is possible to delimit adjacent objects. This is necessary, for example, when deserializing arrays, to avoid a situation where extra members in an array element could throw off the deserialization of subsequent elements.

This definition is overly restrictive and can even prevent arrays of non-mutable elements from being assignable to themselves.

Discussion:

Note that:

- Objects of mutable types can *always* be reliably delimited, because the parameterized CDR representation they use contains an explicit termination sentinel. This is why the original definition of "strongly assignable" is the way that it is.
- Objects of final and extensible types can nevertheless be delimited if they are top-level objects, not contained within any other—the enclosing encapsulation (e.g. an RTPS packet) identifies the length of its contents. The distinction between "assignable" and "strongly assignable" captures this property.
- Objects of final and extensible types can nevertheless be delimited if they occur as members of mutable structures. This freedom is already captured by the existing structure assignability rules.

See also the following issues, which also deal with assignability.

- Issue #16366 addresses the assignability of unions specifically.
- Issue #16368 proposes making object-level assignability rules more flexible.

Rick Warren 2/13/12 10:31 AM
Deleted: At this time, their proposed resolutions do not overlap.

Resolution:

Amend the definition of “strongly assignable” to consider any type strongly assignable to itself.

Revised Text:

After the example table in section 7.2.4, *Type Compatibility: “is-assignable-from” relationship*, there is a paragraph of one sentence: “Any type is assignable from itself.” Remove this sentence.

The definition of the term “strongly assignable” that immediately follows states:

> If types T1 and T2 are both mutable and T1 *is-assignable-from* T2, then T1 is said to be “strongly” assignable from T2.

Immediately append the following sentence:

> Any type is also considered (trivially) strongly assignable from itself, regardless of its extensibility kind.

Disposition: **Resolved**

OMG Issue No: 16368

Title: Object assignability rules should be more resilient

Source:

RTI (Mr. Alejandro de Campos, alejandro@rti.com)

Severity: Minor

Summary:

A small object-level assignability problem within a complex sample can result in an entire sample being discarded. For example, a string whose length is greater than the bound of a string it's being assigned into must be discarded. A sequence with an unassignable member must be discarded. A structure with an unassignable member must be discarded. As a result, a small issue deep within a structure could cause a whole sample to be discarded. For example, an issue within a `TypeObject` could result in an entire built-in topic data sample being discarded and consequently a loss of discovery information.

Discussion:

The consensus of the task force is that the built-in assignability rules should be rigorous and robust to avoid introducing semantic problems that applications will have to work around. However, this issue is not rejected, as there are still ambiguities and weaknesses to resolve in this area.

See also the following issues, which also deal with assignability.

- Issue #16366 addresses the assignability of unions specifically.
- Issue #16367 addresses the concept of "strong assignability".

Rick Warren 2/13/12 10:31 AM
Deleted: At this time, their proposed resolutions do not overlap.

Resolution:

Rick Warren 2/13/12 9:32 AM
Deleted: Proposed

The following are the cases identified in the specification in which individual objects may be unassignable along with the changes to be made, if any:

- *Strings* that are too long: If the bound of a string increases, the string is unassignable. Strings are a basic type, and truncation can generally not be easily detected by applications.
- *Arrays* with unassignable elements: No change. In CDR, arrays have no length or other means of delimiting elements, so it is not possible to "truncate" them.
- *Sequences* that are too long or contain unassignable elements: If the bound of a sequence increases, or an element is unassignable, the whole sequence is unassignable. Applications have little ability to detect or mitigate sequence truncation, so automatic truncation is undesirable.
- *Maps* that are too long or contain unassignable elements: If the bound of a map increases, or an element is unassignable, the whole map is

unassignable. Applications have little ability to detect or mitigate map truncation, so automatic truncation is undesirable.

- *Enumeration* with unrecognized value: No change.
- *Union* whose discriminator does not select a value in the destination type: No change—otherwise, the union would be malformed.
- *Structure* with a "must understand" member not present in the destination type: No change—otherwise, the structure would be malformed.

Add the following new rules for structures, which cover behavior that was previously underspecified:

- *An unassignable non-optional member*: The entire structure is considered unassignable.
- *An unassignable optional member*: The member takes no value. The rest of the structure can be retained.
- *Mismatched "optional" attribute* in a final or extensible structure: The types are not assignable. This restriction is necessary because of the header that precedes such members.

Revised Text:

Rick Warren 2/13/12 9:33 AM
Deleted: Proposed

In section 7.2.4.3, *Collection Types*, in Table 13, *Definition of the is-assignable-from relationship for collection types,* locate the `STRING_TYPE` row. In the middle cell of that row, append: "…and T1.bound >= T2.bound". In the rightmost cell of that row, the second paragraph reads, "T2 strings of length greater than T1.bound are unassignable to T1." Remove this paragraph.

Locate the `SEQUENCE_TYPE` row immediately below. In the middle cell of that row, append: "…and T1.bound >= T2.bound". In the rightmost cell of that row, the middle paragraph reads, "T2 sequences of length greater than T1.bound are unassignable." Remove this paragraph.

In the `MAP_TYPE` row immediately below, in the middle column, append a bullet to the list: "• T1.bound >= T2.bound". In the rightmost cell of that row, the middle paragraph reads, "T2 maps of length greater than T1.bound are unassignable." Remove this paragraph.

In section 7.2.4.5, *Aggregation Types*, in Table 15, *Definition of the is-assignable-from relationship for aggregated types*, locate the `STRUCTURE_TYPE` row. In the rightmost cell of that row, append the following to the first paragraph:

> If a member is unassignable and it is optional, that member takes no value. If it is non-optional, the entire structure is unassignable.

In the same row, in the center column, there is a sentence that begins, "AND if T1 is extensible, …". In that sentence, after the phrase "…have the same

member ID…", insert the phrase "…and the same value of the 'optional' attribute…".

Immediately following is a sentence that starts "AND if T1 is final, …" and a list of conditions. In the second bullet, after the phrase "…have the same member ID…", insert the phrase "…and the same value of the 'optional' attribute…".

Disposition: **Resolved**

Rick Warren 2/13/12 9:33 AM
Deleted: Proposed

Rick Warren 2/13/12 9:33 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

Rick Warren 2/13/12 9:33 AM
Deleted: Disposition: Under Discussion

OMG Issue No: 16559

Title: Inconsistent extensibility kind for enumerations

Source:

RTI (Bettina Swynnerton, bettina@rti.com; Rick Warren, rick.warren@rti.com)

Nature: Clarification

Severity: Significant

Summary:

The specification of the extensibility kind for enumeration types is inconsistent. Figure 10 in section 7.2.2.3.1, “Enumeration Types”, indicates that the `extensibility_kind` of all enumeration types is `FINAL`. However, Table 11 in section 7.2.3, “Type Extensibility and Mutability”, says, “Enumeration types may be final, extensible, or mutable on a type-by-type basis.”

Resolution:

The assignability rules for enumerations already address different extensibility kinds, so we can preserve the more general statement from 7.2.3. Fix Figure 10.

Revised Text:

Remove the following constraint from the depiction of the Enumeration classifier in Figure 10, *Enumeration Types*, in section 7.2.2.3.1, *Enumeration Types*:

```
{extensibility_kind = ExtensibilityKind::FINAL_EXTENSIBILITY}
```

Disposition: **Resolved**

OMG Issue No: 16561

Title: Inability to consume data using new base class

Source:

RTI (Reinier Torenbeek, reinier@rti.com; Rick Warren, rick.warren@rti.com)

Severity: Significant

Summary:

Declared compatibility of types (i.e. type signature) includes base types but not derived types. Unfortunately, that means that a consumer using a base class unknown to an existing producer will not be able to consume data from that producer, because neither party will declare the relationship between the types.

For example: I publish Dog, which extends Animal. In version 2, a new intermediate class Mammal is introduced. However, a new consumer of Mammal won't match with a legacy producer of Dog, because both the Dog and Mammal endpoints declare that their types extend Animal, and neither one asserts that Dog should now extend Mammal.

Resolution:

Rick Warren 2/13/12 9:33 AM
Deleted: Proposed

Remove the type signature concept and go back to registering simple names.

Revised Text:

Rick Warren 2/13/12 9:33 AM
Deleted: Proposed

The resolution below is closely related to the resolutions of issues #15702 and #16097.

In the introduction to section 7.6.1, “Endpoint Matching”, the second bullet begins, “Type Signature”. Replace “Type Signature” with “Type Consistency Enforcement”.

Remove section 7.6.1.2, “Type Signature”.

In section 7.6.4.1.3, “`DynamicTypeSupport`”, the table of properties and operations names an argument to the `register_type` operation “`type_signature`”. Rename this argument to “`type_name`”. Make the same change in the IDL file.

The following sections also mention type signatures, but changes to them are covered by the resolutions to other issues:

- *Changes to section 7.6.1.3, “Type Consistency Enforcement QoS Policy”, are described in the resolution to issue #15702.*
- *Changes to section 7.6.2.1, “Types Used by Publications and Subscriptions”, are described in the resolutions to issues #15702 and #16097.*

Disposition: **Resolved**

Rick Warren 2/13/12 9:33 AM
Deleted: Proposed

Rick Warren 2/13/12 9:33 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 16720

Rick Warren 2/13/12 9:33 AM
Deleted: Disposition: . Under Discussion .

Title: Content-filtered topic doesn't address optional members

Source:

RTI (Alejandro de Campos Ruiz, alejandro@rti.com; Rick Warren, rick.warren@rti.com)

Severity: Minor

Summary:

The specification introduces the concept of optional members into DDS data types. However, it does not address how comparisons of these members should work in content-filtering expressions. For example, if a content-filtered topic indicates that all samples are of interest where "`x != 5`", but x is not present in the sample at all, does the sample pass the filter or not?

Discussion:

The same applies to other new constructs in the type system: we need to specify how to refer to them in a content-filter expression. These include not only optional members of structures but also maps and bit sets.

Resolution:

Optional members:

Introduce the notional value "`null`" that may be used in a content-filter expression. The `null` value is not equal to any value in the domain of any data type. (That is, `x != null` for any non-`null` value of x.)

- An optional member's value in a given sample shall be considered to be equal to `null` if that member is not present in the given sample.
- Its value shall *not* be considered equal to `null` if it is present.
- A non-optional member's value shall *never* be considered equal to `null`.

Maps:

Use the same syntax as arrays and sequences, but instead of a numeric index, use the stringified representation of the key to be queried as a string literal. Such an expression evaluates to the "value" element of the map that is associated with the given key. For example:

```
my_map['foo'] = 'bar'
my_map['5'] = 'baz'
```

Bit sets:

Use the same syntax as arrays and sequences, but instead of a numeric index, use the enumerated constant identifying the desired bit. Such an expression

evaluates to a Boolean value: true if the named bit is set or false if it is not. For example:

```
my_bitset['MY_BIT_CONSTANT']
```

Specify the above rules in a new section 7.6.5, “DCPS Queries and Filters”. Include the relevant updates to the DCPS Queries and Filters grammar (from Annex A in the DDS 1.2 specification).

In addition, include a description of the following things that should be described in DDS already but seem to be missing:

- The use square bracket syntax for arrays, strings, and sequences
- Boolean literals

An issue should be filed against DDS for this, but the resolution of this issue should not wait for it to be acted upon.

Revised Text:

Introduce the following new subsection to section 7.6, *Use of the Type System by DDS*:

7.6.5 DCPS Queries and Filters

[DDS] defines the syntax for content-based filters, queries, and joins in “*Annex A: Syntax for DCPS Queries and Filters*”. This syntax shall be extended as follows.

7.6.5.1 Member Names

[DDS] Section A.2 defines the syntax for referring to a member of a (potentially nested) data structure. Such a reference is known as a `FIELDNAME`. The syntax shall be extended as follows:

- *Arrays and sequences*: Elements in these ordered collections shall be indicated by a zero-based subscript enclosed in square brackets, e.g. `my_collection[0]`. Such an expression shall be considered to have the type that is the element type of the collection.
- *Maps*: Value elements in these unordered collections shall be indicated by a string representation of a corresponding key element, according to the syntax of `STRING`, enclosed in square brackets, e.g. `my_map['key']`. They key shall be expressed as a string even if the map’s key type is an integer type; this distinguishes a map lookup from an index into an ordered collection. Such an expression shall be considered to have the type that is the value element type of the map.
- *Bit sets*: A flag in a bit set shall be indicated by its name, according to the syntax of `ENUMERATEDVALUE`, enclosed in square brackets, e.g.

`my_bitset['MY_FLAG']`. Such an expression shall be considered to have a Boolean type: true if the bit is set or false if it is not. Comparisons with the integer literals 1 and 0 shall also be allowed.

7.6.5.2 Optional Type Members

A member of an aggregated type may be compared to the special value `null`. Such comparisons obey the following rules:

- If the member is optional, and it takes no value in the given object, it shall be considered equal to `null`.
- If the member is optional, and it does take a value in the given object, it shall *not* be considered equal to `null`.
- No non-optional member shall ever be considered equal to `null`.

Inequalities expressed relative to null shall never evaluate to true—no value is greater than or less than `null`.

7.6.5.3 Grammar Extensions

The Parameter production in the grammar given in [DDS] Section A.1 shall be redefined as follows:

```
Parameter ::=
          | CHARVALUE
          | FLOATVALUE
          | STRING
          | ENUMERATEDVALUE
          | BOOLEANVALUE
          | NULLVALUE
          | PARAMETER
          .
```

(New tokens have been highlighted in **blue**.)

The `BOOLEANVALUE` token shall be either `true` or `false` (case-insensitive).

The `NULLVALUE` token shall always be `null`.

Disposition: **Resolved**

OMG Issue No: 17019

Title: Truncation of text in Figure 3

Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The first rectangular symbol in Figure 3 contains truncated text - Type Representation.

Resolution:

Fix the truncated text.

Revised Text:

Fix the truncated text.

Disposition: **Resolved**

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 17020

Title: Non-standard UML notation on figures

Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The following figures exhibit non-standard notation for specialized elements:

- Figure 4
- Figure 9
- Figure 11
- Figure 14
- Figure 16
- Figure 17
- Figure 20

The textual notation showing generalized elements on specialized ones needs to either be hidden, or have the generalized element explicitly shown on the diagram (with generalization relationship between the elements showing).

Discussion:

To hide the generalized element names on specialized ones in Enterprise Architect, double-click on a blank area of the diagram to open the Diagram Properties Dialog. On the "Diagram" tab deselect the "Show additional Parents" option.

Note that Figures 6–8, 10, 12, 13, 15, and 38 exhibit the same problem as well.

Resolution:

Figures 4, 6–9, 12, 13, 15, 16, 17, 20, and 38: Simply hide the additional parents, as described above. *Note* that if the resolution to issue #16561 is accepted, the modification to figure 38 will be irrelevant, because the corresponding section of the document will be removed.

Figures 10, 11, and 14: Add class “Namespace” to the diagram.

Revised Text:

See above.

Disposition: **Resolved**

Rick Warren 2/13/12 9:36 AM
Deleted: Disposition: Under Discussion

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Deleted: Proposed

Rick Warren 2/13/12 9:36 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 17021

Rick Warren 2/13/12 9:36 AM
Deleted: Disposition: Under Discussion

Title: Conflicting data types applied to several Attributes, Operations and Parameters in UML notation

Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The following figures exhibit inconsistent application of data types to attributes, operations and parameters:

- Figure 4
- Figure 9
- Figure 10
- Figure 13
- Figure 14
- Figure 16
- Figure 26
- Figure 30
- Figure 33
- Figure 36
- Figure 37
- Figure 38
- Figure 39

Certain attributes/parameters have 'int' or 'string' applied which conflict the data types defined within the aforementioned model, and the PrimitiveTypes defined in UML. What is the intent with these? Which set of primitive data types are intended for application here (IDL? Java, C++, UML, XTypes?)

For example, the 'Enumeration' class in Figure 10 defines a 'bit-bound' property of type int.

Resolution:

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Use DDS-XTypes types consistently. Note that Figures 9 and 16 are already correct—they use the DDS-XTypes-derived basic types, not the UML-built-in ones.

Note that if the resolution to issue #16561 is accepted, the modification to figure 38 will be irrelevant, because the corresponding section of the document will be removed.

Revised Text:

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Update the figures enumerated above to show DDS-XTypes-derived primitive types consistently.

Disposition: **Resolved**

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Rick Warren 2/13/12 9:37 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 17022

Rick Warren 2/13/12 9:37 AM
Deleted: Disposition: Under Discussion

Title: Invalid data types applied to `AnnotationDescriptor::value`, `DynamicData::descriptor`, `DynamicData::value`

Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The above-mentioned attributes (Figures 28 and 37) have types defined in the form "xxx --> yyy" which has no meaning in UML. Please clarify and correct.

Discussion:

Note that figure 26, “Dynamic Data and Dynamic Type”, displays the properties of the `DynamicData` class and hence also needs to be updated.

Resolution:

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Change the type of `AnnotationDescriptor::value to Map<String, String>`.

Unify the property `DynamicData::descriptor` with the existing association to `MemberDescriptor`, and qualify this association by `MemberId`.

Change the property `DynamicData::value` to an association to `Type`, and qualify this association by `MemberId`.

Revised Text:

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

In section 7.5.2, “Dynamic Language Binding”, in figure 26, “Dynamic Data and Dynamic Type”, display the model changes described above and add `Type` to the diagram.

In section 7.5.2.3, “`AnnotationDescriptor`”, in the table “`AnnotationDescriptor` properties and operations”, replace the text “`string<Char8,256>` → `string<Char8,256> [0..1]`” with “`Map<String<Char8,256>, String<Char8,256>>`”.

In figure 28 immediately below, change the type of `value` to `Map`. Also add two constraints to `AnnotationDescriptor`:

```
{ value.element_type = String }
{ value.key_element_type = String }
```

In section 7.5.2.11, “Dynamic Data”, in figure 27, “Dynamic Data and Dynamic Data Factory”, display the model changes described above. Also add `Type` to the figure.

Disposition: **Resolved**

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Rick Warren 2/13/12 9:37 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 17023

Title: Circular reference in definition of specialized Primitive Types

Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The specialized Primitive Types defined for `Int`, `UInt`, `Float`, `Boolean`, etc. Each defines a property 'value' that is typed by the property's owner thus resulting in a circular reference in the model.

- Figure 6
- Figure 7
- Figure 8

These are specializations of the `PrimitiveType` abstract class. Changing the UML instance of the `PrimitiveType` definition to a `UML::PrimitiveType` will automatically convey the semantic that an instance of the specialized `PrimitiveType` is the value itself.

Discussion:

To do this in Enterprise Architect, recreate the `PrimitiveType` abstract definition as an instance of the 'Primitive' element found in the 'Class' toolbox.

Unfortunately, `PrimitiveType` and the classes that specialize it cannot be changed to specialize `UML::PrimitiveType`, because a primitive cannot extend a class or even specialize an interface. Therefore, these types would be effectively removed from the Type classifier hierarchy, which would break other parts of the model.

Resolution:

Remove the `value` members.

Revised Text:

Update figures 6, 7, and 8 to display the aforementioned model changes.

Disposition: **Resolved**

Rick Warren 2/13/12 9:37 AM
Deleted: Disposition: Under Discussion

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Rick Warren 2/13/12 9:37 AM
Deleted: Proposed

Rick Warren 2/13/12 9:37 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

OMG Issue No: 17090

Title: Inconsistent generated code for `DynamicType::descriptor`

Rick Warren 2/13/12 9:37 AM
Deleted: Disposition: Under Discussion

Source:

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

Severity: Minor

Summary:

`DynamicType` has a property "descriptor" of type (read-only) `TypeDescriptor`. This property is mapped to IDL as a `readonly` attribute. When code is generated for this attribute in C or C++, the corresponding getter is not consistent with the other getters in the same class, either in terms of its style or in terms of its memory management. (The other getters are all defined explicitly as methods in IDL.)

Discussion:

The same issue applies to `DynamicTypeBuilder` (see the resolution to issue #16240) and `DynamicTypeMember`.

Resolution:

Rick Warren 2/13/12 9:38 AM
Deleted: Proposed

Remove these attributes and add explicit getter methods, like:

```
DDS::ReturnCode_t get_descriptor(
        inout TypeDescriptor descriptor);
```

This method overwrites a user-provided object, just like `DynamicType::get_member_by_name` and `get_annotation`.

Revised Text:

Rick Warren 2/13/12 9:38 AM
Deleted: Proposed

In section 7.5.2.8, "`DynamicType`", in the table of properties and operations, remove the row for the property "`descriptor`". Add the following new rows:

`get_descriptor`		`DDS::ReturnCode_t`
	`inout descriptor`	`TypeDescriptor`

Rename section 7.5.2.8.2, "Property: `descriptor`", to "Operation: `get_descriptor`". Replace the body of this section as follows:

> This operation provides a summary of the state of this type. It overwrites the state of an application-provided object.

If the argument is nil, this operation shall fail with `RETCODE_BAD_PARAMETER`.

Apply the same set of changes to section 7.5.2.9, "`DynamicTypeBuilder`" (see issue #16240).

Apply the same set of changes to section 7.5.2.6, "`DynamicTypeMember`", with one difference: the type of the descriptor is `MemberDescriptor`, not `TypeDescriptor`.

In the IDL file and the corresponding Annex, replace `DynamicTypeMember::descriptor` with the following method definition:

```
DDS::ReturnCode_t get_descriptor(
        inout MemberDescriptor descriptor);
```

Likewise, replace `DynamicType::descriptor` and `DynamicTypeBuilder::descriptor` with the following method definition (see issue #16240):

```
DDS::ReturnCode_t get_descriptor(
        inout TypeDescriptor descriptor);
```

Disposition: **Resolved**

Rick Warren 2/13/12 9:38 AM
Deleted: Proposed

Rick Warren 2/13/12 9:38 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

Disposition: Deferred

Rick Warren 2/13/12 9:38 AM
Deleted: Disposition: Under Discussion

OMG Issue No: 15946

Title: All IDL should use local interfaces

Source:

Remedy IT (Mr. Johnny Willemsen, jwillemsen@remedy.nl)

Nature: Clarification

Severity: Minor

Summary:

All IDL should be using local interfaces, not regular interfaces, please add the keyword local to all interfaces.

Discussion:

The only non-local interfaces in the DDS-XTypes API are specializations of “generic” interfaces defined by the DDS specification. It is not appropriate to change these to local interfaces until the corresponding change is made in the DDS specification itself. An issue has already been filed for the DDS RTF to make this change.

Resolution:

Defer this issue until the corresponding one is addressed in DDS itself.

Disposition: **Deferred**

OMG Issue No: 17018

Title: Rendering of UML diagrams in each of the figures is of poor graphical quality

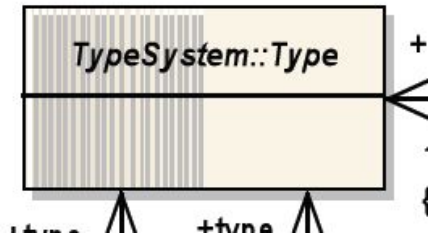
Source:

Sparx Systems (Sam Mancarella, sam.mancarella@sparxsystems.com)

Severity: Support Text

Summary:

The gradient shading rendered in each of the elements on a low-color image results in elements rendered as follows:



The diagrams should be rendered either by using a higher-color format (such as PNG), or without the gradient rendering enabled.

Discussion:

To turn off the gradient shading in Enterprise Architect, go to the "Tools->Options" dialog. Under the "Diagram->Appearance" group:

1) Deselect the "Show Gradient Fill for Paper Color" option
2) Set the "Gradient Fill Direction for an Element" to "<none>"

Resolution:

Make these changes in the EAP file so that as diagrams are updated, this problem will be fixed. In the mean time, defer this issue. Do not preemptively regenerate every diagram at this time.

Disposition: **Deferred**

Rick Warren 2/13/12 9:49 AM
Deleted: Proposed

Rick Warren 2/13/12 9:49 AM
Deleted: Proposed

Rick Warren 2/13/12 9:49 AM
Formatted: Default Paragraph Font, Font:Arial, Bold

Disposition: Closed, no change

Rick Warren 2/13/12 9:49 AM
Deleted: Disposition: Under Discussion

OMG Issue No: 15981

Title: C++ mapping MAP type needs member access semantics specification

Source:

Remedy IT (Mr. Martin Corino, mcorino@remedy.nl)

Nature: Clarification

Severity: Significant

Summary:

The C++ mapping for the new MAP type needs a precise specification of member (or key) access semantics and memory management rules like the CORBA C++ mapping describes for the SEQUENCE type on pages 36-42 (par. 4.15) of the C++ language mapping, version 1.2, formal/2008-01-09.

Discussion:

[Rick] I don't understand what the specification is missing. If using the equivalent IDL, the IDL-to-C++ mapping defines the semantics. And if using `std::map`, that container defines its own semantics.

The spec says in 7.5.1.3.1: "An IDL map type shall be transformed into an instantiation of the `std::map` template such that the `Key` parameter is the C++ type corresponding to the IDL key element type and the `T` parameter is the C++ type corresponding to the IDL value element type."

For example, IDL `map<long, string>`: the mapping for `long` in C++ is in 7.5.1.1.2: `DDS::Int32`, `CORBA::Long`, or `int32_t` are all legal options. The mapping for `string` is not overridden; the previously defined language binding remains in force: "This mapping reuses the OMG-standard IDL-to-language mappings..." at the beginning of 7.5.1.

[Martin] This certainly clears things up for me.

Resolution:

Close this issue without making any change.

Disposition: **Closed, no change**

Disposition: Duplicate/merged

OMG Issue No: 16238

Title: Type signature fields in built-in topic data shouldn't use unbounded strings

Source:

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

Severity: Minor

Summary:

The `type_name` members of `TopicBuiltinTopicData`, `PublicationBuiltinTopicData`, and `SubscriptionBuiltinTopicData` are bounded at 256 characters. The `equivalent_type_name` and `base_type_name` members store sequences of type names, but the strings in those sequences are unbounded. The flexibility to store strings of any length is unnecessary, because no string can appear there that doesn't also appear in a `type_name` somewhere, and is therefore of 256 characters or less.

This issue is minor in that it does not impact correctness. However, it does impact clarity to a small extent and can impact the efficiency of some implementations.

Resolution:

This issue is obsolete in the face of issue #16561: the resolution of that issue eliminates the type signature concept altogether. Therefore, this issue should be merged with that one.

Disposition: **Merged with issue #16561**

Rick Warren 2/13/12 9:34 AM
Deleted: Proposed

Rick Warren 2/13/12 10:31 AM
Deleted: proposed

Rick Warren 2/13/12 10:31 AM
Deleted: would

Rick Warren 2/13/12 9:34 AM
Deleted: Proposed

Rick Warren 2/13/12 9:34 AM
Formatted: Default Paragraph Font

Rick Warren 2/13/12 9:34 AM
Deleted:
Disposition: Under Discussion