Java 2.4 RTF Final Report - ptc/99-03-03

Jeff Mischkinsky, Java 2.4 RTF Chair
March 19, 1999

This report memorializes the actions and resolutions of issues decided by the Java 2.4 Revision Task Force.

The members of the RTF were:
Mary Leland (HP) [mleland@fpk.hp.com]
Colm Caffery(IONA) [ccaffrey@iona.com]
Jeff Mischkinsky (INPRISE), Chair [jeffm@inprise.com]
Simon Nash (IBM) [nash@hursley.ibm.com]
David Heisser (Sun) [dheisser@eng.sun.com]
Chris Jacobi(Xerox) [jacobi@parc.xerox.com] resigned after Vote 1

The Task force was chartered on Sep. 18, 1998. The Task Force considered all of the 18 issues that were outstanding as of close of business March 5, 1999, well after the closing comment date of Jan 8, 1999. The Task Force concluded its work on March 19, 1999.

Of the 18 outstanding issues, 17 were successfully resolved.
Issue 1751 was deferred and left to a future RTF to consider.

The Task Force conducted almost all of its formal work via email and votes on ballots that were distributed via email and on the omg web site. The Task Force held only one formal meeting on Nov. 19, 1998. The minutes and formal actions for that meeting are available on the omg web site as document # ptc/98-11-03. The primary action with respect to issue resolution was to vote on the partial resolution of 1897.

The base document (the Java language mapping chapter) against which the Task Force started its work was OMG document # ptc/98-10-14. As work proceeded and interim drafts were produced applying the results of issue resolutions, resolutions were written against the updated draft chapter (which was made available on the OMG server in the /pub/orbrev/drafts directory. The major effect of this was to change references to the section numbers in later resolutions.

The Chair will complete editing of the base document (CORBA CORE Chapter 25) and make a review copy available as OMG document # ptc/99-03-04.

For the record, votes which passed for which resolutions of the various issues are listed below:
98-11-19 Meeting: 1897 (except for package name)
Vote 1: 1993 2255 2256 2462 1897(package name to be in CORBA_2_3)
Vote 2: 1750 1752 1942 2079 2228 2233 2462
defer 1751
Vote 3: 2514
Vote 4: 2551
Vote 5: 1980 proposal not passed
Vote 6: 1941 1964 2096
Vote 7: 1980

The Chair would personally like to thank the Task Force members and other significant contributors for their hard work and cooperation

Respectfully submitted,

Jeff Mischkinsky, (jeffm@inprise.com, jeff_mischkinsky@omg.org)
Java 2.4 RTF Chair
March 19, 1999

Consolidated Record of Votes

(YES means YES to All)

Voter	Vote 1	Vote 2	Vote 3	Vote 4	Vote 5	Vote 6	Vote 7
Mary Leland (HP)	YES	YES	YES	YES	ABS	YES	YES
Colm Caffery (IONA)	YES	YES	ABS	YES	YES	YES	YES
Jeff Mischkinsky (INPRISE)	YES	YES	YES	YES	YES	YES	YES
Simon Nash (IBM)	YES 1897,1993 ABS 2255, 2256	YES	YES	YES	NO	YES	YES
David Heisser (Sun)	YES	YES	ABS	YES	NO	YES	YES
Chris Jacobi (Xerox)	NV	Resigned					

Issues Addressed and Resolutions

Issue 1750: 2 IDL->Java issues on the singleton ORB (01) (java-rtf)

Click here for this issue's archive.
Nature: Uncategorized Issue
Severity:
Summary: - There may be other work in progress on this, but I just noticed an inconsistency. CORBA 2.2 24.18.8 states variously that, in an applet context, the singleton ORB may only be used for creating TypeCodes and Anys, and later that it may only be used for creating TypeCodes. I assume that the former is correct, perhaps the latter should be amended.
Resolution: spec already says this
Revised Text:none
Actions taken: Close, no action
July 29, 1998: received issue

Issue 1751: 2 IDL->Java issues on the singleton ORB (02) (java-rtf)

Nature: Uncategorized Issue
Severity:
Summary: How is an ORB to determine whether or not it is in an applet context? The nearest that I can see is to check for the presence of a SecurityManager. If we are in an applet context, then there is a SecurityManager present, but the inverse (that the presence of a SecurityManager implies that we are in an applet) is not true. It is however the case that whenever a SecurityManager is present, potentially untrusted code is present, so the same constraints on the singleton ORB are probably appropriate. Therefore, I propose that the specification be changed to state that if System.getSecurityManager() returns a non-null result, the singeton ORB should be constrained to creating TypeCodes and Anys.
Resolution:
Revised Text:
Actions taken:Defer
July 29, 1998: received issue

Issue 1752: mapping of IDL enumerations to java (java-rtf)

Click here for this issue's archive.
Source: GemStone Systems (Mr. Bruce Schuchardt, bruce@gemstone.com)
Nature: Revision
Severity: Significant
Summary: he current mapping of IDL enumerations to Java relies on there being one and only one instance of an enum. The instance is held as a static variable in the enumeration class. Since there is only

one instance, identity comparisons are used in comparing enums and the enumeration value class does not implement equality or hash methods based on the value of the enum.
Resolution: this is the way it is supposed to behave
Revised Text:
Actions taken:Close, no action
July 29, 1998: received issue

Issue 1897: Evolving the org.omg.* APIs (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Mr. Simon C. Nash, nash@hursley.ibm.com)
Nature: Revision
Severity:
Summary: Now that the org.omg.* APIs are built into the JDK core (from JDK 1.2 onwards), there is an issue with how these could be evolved to support changes to the OMG spec that happen between JDK major releases. The following is a proposal for how this could be handled.
Resolution:
1. Change the IDL/Java mapping of enums so that
   a. the generated Java classes are not final
   b. the from_int method is not final, and
   c. the constructor is protected instead of private.

   Rationale:
   This change has previously been discussed by this RTF. It is needed in this context because of the need to include some Java classes mapped from IDL enums (e.g., TCKind) in the core JDK. If new members are added to such an enum later, there needs to be a way to subclass the generated Java class to produce a subclass that corresponds to the new IDL (or PIDL) enum.

2. Change the Object, ObjectImpl, and Delegate classes to
   a. add new methods
      org.omg.CORBA.Object _get_interface_def()
   to Object and ObjectImpl and
      org.omg.CORBA.Object get_interface_def(org.omg.CORBA.Object self)
   to Delegate.

   b. deprecate the existing methods
      org.omg.CORBA.InterfaceDef _get_interface()
   on Object and ObjectImpl and
      org.omg.CORBA.InterfaceDef get_interface(org.omg.CORBA.Object self)
   on Delegate.

   Rationale:
   The new methods do not refer to Interface Repository types in their signatures. However, at runtime they still return the same IR objects as the deprecated methods. Since this signature change is a binary incompatible change, and since Java methods cannot be overloaded on their return types, the new methods have different names than the

deprecated methods. This allows an easier path for user migration than if the method signatures had been updated "in place".

Removing IR types from the signatures allows the new methods to be included in the core JDK without also having to include all the IR classes in the core JDK. With likely changes in this area in the near future because of the Components RFP, it is felt to be unwise to put the IR into core in JDK 1.2.

3. Change the ORB class to
a. add a new method
public NVList create_operation_list(org.omg.CORBA.Object oper)

b. deprecate the existing method
public NVList create_operation_list(org.omg.CORBA.OperationDef oper)

Rationale:
The reason for this change is as for item 2 above. Since Java supports overloading on argument types, there is no need to change the method name in this case.

4. Move some methods needed by Objects By Value from the ORB, InputStream and
OutputStream classes to new subclasses (that will not be part of the core JDK 1.2). The names of these new classes are:
org.omg.CORBA_2_3.ORB
org.omg.CORBA_2_3.portable.InputStream
org.omg.CORBA_2_3.portable.OutputStream

and the methods affected are:
ORB.get_value_def
ORB.register_value_factory
ORB.unregister_value_factory
ORB.lookup_value_factory
InputStream.read_Value
InputStream.read_Abstract
OutputStream.write_Value
OutputStream.write_Abstract
OutputStream.start_block
OutputStream.end_block

Revised Text:
Actions taken:
August 28, 1998: received issue

Issue 1941: Problem mapping "global" types in Java mapping (java-rtf)

Click here for this issue's archive.
Source: Inprise Corporation (Mr. George M. Scot, gscott@inprise.com gscott@visigenic.com)
Nature: Uncategorized Issue
Severity:
Summary: The current IDL/Java mapping cannot correctly map IDL "global" types correctly in all cases.
Resolution:
Revised Text:
Actions taken: Close, no action
September 9, 1998: received issue

Issue 1942: Need overloaded write_Value method on OutputStream (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Mr. Simon C. Nash, nash@hursley.ibm.com)
Nature: Uncategorized Issue
Severity:
Summary: We have come across a case in the Java-to-IDL mapping where we need another overloaded form of the write_Value method on the org.omg.CORBA.portable.OutputStream class.
Resolution: already resolved
Revised Text:
Actions taken:Close, no action
September 10, 1998: received issue

Issue 1964: Creating TypeCodes safely in java (java-rtf)

Click here for this issue's archive.
Source: Inprise Corporation (Mr. George M. Scott, gscott@inprise.com gscott@visigenic.com)
Nature: Uncategorized Issue
Severity:
Summary: There are problems with creating recursive TypeCodes containing values.Our proposal is really quite simple. We want to add a new class to the org.omg.CORBA.portable package defined as follows: package org.omg.CORBA.portable; final public class TypeCodeLock { } All generated type() methods in Helper's could synchronize on the TypeCodeLock's class object to grab a global lock which is used during TypeCode creation to guarnatee the atomicity of TypeCode creation. Note, this is essentialy what is done in C++ where TypeCodes are typically initialized in a global static initalizer which is done from a single thread. We are just allowing the typecodes to be created lazily.
Resolution:
Revised Text:
Actions taken: Close, no action
September 16, 1998: received issue

Issue 1980: Updated proposal for the OBV Java mapping (java-rtf)

Click here for this issue's archive.
Source: Inprise Corporation (Mr. George M. Scot, gscott@inprise.com gscott@visigenic.com)
Nature: Revision
Severity:
Summary: updated proposal for the OBV Java mapping which addresses a number of issues which have been raised by us and others. This proposal is based on a number of discussions and previous proposals and current OMG submissions (thanks to Simon Nash and Bernard Normier). As with previous poposals, these are not currently formal proposals, just working drafts. ;-) Here are the current issues: 1. Java valuetypes cannot unmarshal recursive references to themselves. This is the same problem that occurs with custom valuetypes. 2. The current language mapping mixes both generated code with user written code in the same source file. This poses a very complex "tool" issue for IDL compilers which is unnecessarily complex. 3. Java valuetypes need a way to unmarshal the state of their base class. 4. The addition of the new Helper interface adds ~400 bytes to every Helper class, of which there are about 250 in a typical ORB implementation. Which is an overhead of about 100k just to support an optimization of a corner case in RMI/IIOP where an RMI type happens to contain an IDL type. This doesn't even begin to address the bloat that would occur to user code as well as any additional CORBA services. The space/time tradeoff here appears to have gone the wrong way. 5. The ValueHelper interface contains the method get_safe_base_ids, which is inconsistent with current OBV terminology. 6. The marshaling of boxed types should be considered carefully, because of the special casing required for boxed strings, arrays, and sequences. 7. The compiler should provide compile time enforcement of init() declarations.

Resolution: The solution is modify the mapping so as to use an easier deal with form of a 2 class mapping,
much like C++, to add facilites that will fix the bugs that don't allow recursive structures to be marshaled and unmarshaled, reduce the "bloat", handle all the boxed values correctly, etc.

This proposal, coupled with the proposal to fix Issue 1981, which deals with the "init" portion of the issue at the IDL level, fixes all the known problems with the OBV/Java mapping.
Revised Text:

The changes are relative to version of Chapter 25 which has had Issue 1897 Evolving the org.omg.* APIs
applied to it (available as ftp://ftp.omg.org/orbrev/drafts/idljava_2_4v1.2.pdf)

Add in section 25.4.1.4 after ObjectHolder

```
final public class ValueBaseHolder {
  public org.omg.CORBA.portable.ValueBase value;
  public ValueBaseHolder() {}
  public ValueBaseHolder(org.omg.CORBA.portable.ValueBase initial) {...}
  public void _read(org.omg.CORBA.portable.InputStream is) {...}
  public void _write(org.omg.CORBA.portable.OutputStream os){...}
  public org.omg.CORBA.TypeCode _type(){...}
}
```

Replace section 25.5.1 with:
25.5.1 Generic BoxedValueHelper Interface

```
package org.omg.CORBA.portable;
```

```
public interface BoxedValueHelper {
  java.io.Serializable read_value(InputStream is);
  void write_value(OutputStream os, java.io.Serializable value);
  java.lang.String get_id();
}
```

In 25.5.2 delete the 2nd and 5th paragraphs.

In 25.5.2 delete the specification of the "generated Java helper (value types)

In 25.5.2, in the specification of the "generated Java helper (non value types):

a. delete the "(non value types)" from the comment

b. Add the following methods to the end of the generated <typename>Helper class specification:

```
// for each initializer in non abstract value type (see section 25.5.2.1)
public static <typename> <initializername> (org.omg.CORBA.ORB orb, [<initializer arguments>]) {...}
```

c. Remove the paragraph following that starts with: "For any user defined, value type ... "

Add a new section 25.5.2.1 "Value type initializer convenience functions" as follows:

For each initializer in a value type declaration, a corresponding static convenience method is generated in the helper class for the value type. The name of this method is the name of the initializer. This method takes an orb instance and all the arguments specified in the initializer argument list. The implementation of each of these methods will locate a <typename>ValueFactory (see section xxx) and call the identically named method on the ValueFactory passing in the supplied arguments.

Replace all of section 25.13 Mapping for Value Type with:

25.13.1 Supporting interfaces for value types

25.13.1.1 ValueBase interface

```
package org.omg.CORBA.portable;

public interface ValueBase extends IDLEntity {
    String[] _truncatable_ids();
}
```

All values implement ValueBase either directly (for boxed primitives - see section 25.14.1), or indirectly by implementing either the StreamableValue or CustomValue interface (see below).

25.13.1.2 StreamableValue interface

```
package org.omg.CORBA.portable;
public interface StreamableValue extends Streamable, ValueBase {
```

```
    }
```

All non-boxed IDL valuetypes that are not custom marshalled, implement this interface.

25.13.1.3 CustomMarshal interface

```
package org.omg.CORBA;
public interface CustomMarshal {
    public void marshal (org.omg.CORBA.DataOutputStream os);
    public void unmarshal (org.omg.CORBA.DataInputStream is);
}
```

Implementors of custom marshalled values implement the above interface to provide custom marshalling.

25.13.1.4 CustomValue interface

```
package org.omg.CORBA.portable;
public interface CustomValue extends ValueBase, org.omg.CORBA.CustomMarshal {
    }
```

All custom value types generated from IDL implement this interface.

25.13.1.5 ValueFactory interface

```
package org.omg.CORBA.portable;

public interface ValueFactory {
    java.io.Serializable read_value(InputStream is);
}
```

The ValueFactory interface is the native mapping for the IDL type ***CORBA::ValueFactory***. The ValueFactory's **read_value()** method is called by the ORB runtime in the process of unmarshaling a valuetype. A user must implement this method as part of implementing a type specific ValueFactory. In this implementation, the user must call java.io.Serializable is.read_value(java.io.Serializable) with a blank valuetype to use for unmarshalling. The value returned by the stream is the same value passed in with all the data unmarshalled.

25.13.2 Basics for stateful value types

A concrete value type (i.e. one that is not declared as abstract) is mapped to an abstract Java class with the same name, and a factory Java interface with the suffix "**ValueFactory**" appended to the value type name. In addition, a helper class with the suffix "**Helper**" appended to the value type name and a holder class with the suffix "**Holder**" appended to the value type name shall be generated.

The specification of the generated holder class is as follows:

```
public final class <typename>Holder implements org.omg.CORBA.portable.Streamable {
    public <typename> value;
    public <typename>Holder () {}
    public <typename>Holder (final <typename> initial) {
        value = initial;
    }
```

```
    public void _read (final org.omg.CORBA.portable.InputStream input) {...}
    public void _write (final org.omg.CORBA.portable.OutputStream output) {...}
    public org.omg.CORBA.TypeCode _type () {...}
}
```

The value type's mapped Java abstract class contains instance variables that correspond to the fields in the state definition in the IDL declaration. The order and name of the Java instance variables shall be the same as the correspondng IDL state fields. Fields that are identified as public in the IDL are mapped to public instance variables. Fields that are identified as private in the IDL are mapped to protected instance variables in the mapped Java class.

The Java class for the value type extends either **org.omg.CORBA.portable.CustomValue** or **org.omg.CORBA.portable.StreamableValue,** depending on whether it is declared as custom in IDL or not, respectively.

The generated Java class shall provide implementation of the **ValueBase** interface for this value type.

The value type's generated value factory interface extends **org.omg.CORBA.portable.ValueFactory** and contains one method corresponding to each initializer declared in the IDL. The name of the method is the same as the name of the initializer, and the initializer arguments are mapped in the same way as ***in*** parameters are for IDL operations. The implementor shall provide a factory class with implementations for the methods in the generated value factory interface. When no initializers are declared in IDL, then the value type's value factory is eliminated from the mapping and the implementor shall simply implement **org.omg.CORBA.portable.ValueFactory** to provide the method body for **read_value()**.

The inheritance scheme and specifics of the mapped class depend upon the inheritance and implementation characteristics of the value type and are described in the following subsections.

The mapped Java class contains abstract method definitions which correspond to the operations and attributes defined on the value type in IDL.

An implementor of the value type extends the generated Java class to provide implementation for the operations and attributes declared in the IDL, including those for any derived or supported value types or interfaces.

25.13.2.1 Inheritance from values

- Value types that do not inherit from other values or interfaces:

For non custom values, the generated Java class also implements the **StreamableValue** interface and provides appropriate implementation to marshal the state of the object. For custom values, the generated class extends **CustomValue** but does not provide an implementation for the **CustomMarshal** methods.

- inheritance from other stateful values

The generated Java class extends the Java class to which the inherited value type is mapped

- inheritance from abstract values

The generated Java class implements the Java interface to which the inherited abstract value is mapped(see section 25.13.3).

- supported interfaces

The Java class implements the **Operations** Java interface of all the interfaces(if any) that it supports. (Note that the operations interface for abstract interfaces does not have the "Operations" suffix, see section 25.12.1.1). The implementation of the supported interfaces of the value type shall use the tie mechanism, to tie to the value type implementation.

25.13.3 Abstract Value Types
An abstract value type maps to a Java interface that implements ValueBase and contains all the operations and
attributes specified in the IDL, mapped using the normal rules for mapping operations and atttributes.

Abstract value types cannot be implemented directly. They must only be inherited by other stateful value types or abstract value types.

25.13.4 CORBA::ValueBase
CORBA::ValueBase is mapped to java.io.Serializable.

The **get_value_def()** operation is not mapped to any of the classes associated with a value type in Java. Instead it appears as an operation on the ORB pseudo object in Java(see "public static org.omg.CORBA.Object get_value_def(String repId)" in section 25.19.10).
Note: This implies fixing up the referenced text in !too
25.19.10 to have the correct signature

25.13.5 Examples

In 25.13. 4 Example A

In the IDL
change **init(in long w);**
to: factory **create(in long w);**

Replace the generated Java by:

// generated Java

```
package ExampleA;

public abstract class WeightedBinaryTree implements
org.omg.CORBA.portable.StreamableValue {
 // instance variables
 protected int weight;
 protected ExampleA.WeightedBinaryTree left;
 protected ExampleA.WeightedBinaryTree right;

 abstract public int[] preOrder ();
 abstract public int[] postOrder ();

 public org.omg.CORBA.TypeCode _type () {...}

   public void _read (final org.omg.CORBA.portable.InputStream _input) {
      // read state information using the wire format
     ...
    }
    public void _write (final org.omg.CORBA.portable.OutputStream _output) {
     ....
    }

    public java.lang.String[] _truncatable_ids () {...}
}

public final class WeightedBinaryTreeHelper {
  < ed note: put all the helper methods here>

   public static WeightedBinaryTree create(ORB orb, int w) {
      ...
   }
}

final public class WeightedBinaryTreeHolder implements org.omg.CORBA.portable.Streamable {

   ...
<Note: the code for this isn't changing so just leave the old code there>
}

public interface WeightedBinaryTreeValueFactory extends
org.omg.CORBA.portable.ValueFactory {
   public ExampleA.WeightedBinaryTree create (int w);
}
```

In 25.13.5 Example B

Keep the IDL and the Java mapping of the Printer interface as is.
Replace the generated Java for the WeightedBinaryTree with the following:

```
// generated java
package ExampleB;

public abstract class WeightedBinaryTree implements
org.omg.CORBA.portable.StreamableValue, PrinterOperations {
    protected int weight;
    protected ExampleB.WeightedBinaryTree left;
    protected ExampleB.WeightedBinaryTree right;
    abstract public int[] preOrder ();
    abstract public int[] postOrder ();

    public org.omg.CORBA.TypeCode _type () {
      ...
    }
    public void _read (final org.omg.CORBA.portable.InputStream _input) {
      ...
    }
    public void _write (final org.omg.CORBA.portable.OutputStream _output) {
      ...
    }
    public java.lang.String[] _truncatable_ids () {
      ...
    }
}

public final class WeightedBinaryTreeHelper {
    .... //  helper methods...
< ed note: put all the helper methods here>

}

public final class WeightedBinaryTreeHolder {
 ... <note this hasn't changed, fill in old one here>
}

// user written code for default ValueFactory
public class WeightedBinaryTreeDefaultFactory implements
org.omg.CORBA.portable.ValueFactory {
    public java.io.Serializable read_value (org.omg.CORBA.portable.InputStream is) {
    //user implements code
    }
}
```

Add a new 25.13.5 Example C

```
// IDL
typedef sequence<unsigned long> WeightedSeq;

module ExampleC {
```

```
    custom valuetype WeightedBinaryTree {
      private unsigned long weight;
      private WeightedBinaryTree left;
      private WeightedBinaryTree right;
      factory create(in long w);
      WeightedSeq preOrder();
      WeightedSeq postOrder();
    };
};
```

// generated Java
package ExampleC;

abstract public class WeightedBinaryTree implements org.omg.CORBA.portable.CustomValue {...}
public final class WeightedBinaryTreeHelper {...}
public final class WeightedBinaryTreeHolder {...}

[editing note: fill in the ... as an aid to the reader]

25.13.6 Keep as is currently in the specification

25.13.7 ValueFactory and Marshaling

Replace second bullet in factory lookup algorithm by:

- If this is not successful and the repository id is a standard IDL repository id that starts with "IDL:",then extract the class name from the repository id by stripping of the "IDL:" header and ":<major>.<minor>" version information trailer and replacing all "/"s with "."s. Then attempt to load a value factory by appending a "**DefaultFactory**" suffix to the above class name.

Replace third bullet in factory lookup algorithm by:
- If this is not successful and the repository id is a standard RMI repository id that begins with "RMI:", then extract the class name from the repository id by stripping of the "RMI:" header and the ":<hashcode>:[<suid>]" trailer and applying all necessary conversions(see section 26.****). The **ValueHandler** interface is used to read in the value, if it does not implement IDL Entity.

Replace paragraph following the bullets by:

The IDL native type **ValueFactory** is mapped in Java to **org.omg.CORBA.portable.ValueFactory**.

In 25.14 Mapping for Value Box Type, replace the 3rd paragraph with the following:

A boxed value needs to be treated differently than regular values in Java. Boxed values don't have factories and don't implmenent either the **StreamableValue** or **CustomValue** interfaces, so their

marshalling and unmarshalling is performed by a boxed value helper object. In all cases, code can be generated to unmarshal the boxed type. No user code is required for boxed values.

The **BoxedValueHelper** interface is implemented by all generated Helper classes for boxed valuetypes. The inherited **read_value()** method is called by the ORB runtime in the process of unmarshalling a boxed valuetype. This is required for types that are immutable either in content (eg, string), or size (eg, sequences). The **write_value()** method call is used for marshalling the value box.

There are two general cases to consider. value boxes of primitive Java types and value boxes for entities that are mapped to java classes.

In Section 25.14.1

Replace the first Java class <box_name> to be:

```
public class <box_name> implements ValueBase {
   public <mapped_primitive_Java_type> value;
   public <box_name>(<mapped_primitive_Java_type> initial)
      { value = initial; }
   private static String[] _ids = { <box_name>Helper.id() };
   public String[] _truncatable_ids()
      { return _ids; }
}
```

Replace the third Java class <box_name>Helper to be:

```
final public class <box_name>Helper implements org.omg.CORBA.portable.BoxedValueHelper {
     public <box_name> read_value(InputStream is) {...}
     public write_value(OutputStream is, <box_name> value) {....}
     public String get_id() { ... }
     .... // other helper methods
}
```

In Section 25.14.1.1 Primitive Type example:

Replace the MyLong class by:

```
public class MyLong implements ValueBase {
   public int value;
   public MyLong(int initial) { value = initial; }
   private static String[] _ids = { MyLongHelper.id() };
   public String[] _truncatable_ids() { return _ids; }
}
```

Replace MyLongHelper by:

```
final public class MyLongHelper implements org.omg.CORBA.portable.BoxedValueHelper {
     public MyLong read_value(InputStream is) {...}
     public write_value(OutputStream is, MyLong value) {....}
     public String get_id() { ... }
     .... // other helper methods
}
```

Add to end of paragraph on 25.14.2

```
//IDL
 valuetype <box_name> <IDLtype>;

final public class <box_name>Helper implements org.omg.CORBA.portable.BoxedValueHelper {
     public <IDLType> read_value(InputStream is) {...}
     public write_value(OutputStream is, <IDLType> value) {....}
     .... // other helper methods
}

final public class <box_name>Holder implements org.omg.CORBA.portable.Streamable {
    public <mapped_java_class> value;
    ...
}
```

Add a new section 25.14.3 Examples

```
// IDL

module A {
 valuetype BoxedString string;
};

// generated Java
package A;
public final class BoxedStringHelper implements org.omg.CORBA.portable.BoxedValueHelper {
     private static final BoxedStringHelper _instance = new BoxedStringHelper();
     public static java.lang.String read (final org.omg.CORBA.portable.InputStream _input) {
       if (!(_input instanceof org.omg.CORBA_2_3.portable.InputStream)) {
            throw new org.omg.CORBA.BAD_PARAM();
       }
       return
(java.lang.String)((org.omg.CORBA_2_3.portable.InputStream)_input).read_value(_instance);
  }

public static void write (final org.omg.CORBA.portable.OutputStream _output, final
java.lang.String value) {
       if (!(_output instanceof org.omg.CORBA_2_3.portable.OutputStream)) {
            throw new org.omg.CORBA.BAD_PARAM();
       }
       ((org.omg.CORBA_2_3.portable.OutputStream)_output).write_value(value, _instance);
  }
```

```
  public static void insert (org.omg.CORBA.Any any, java.lang.String value) {...}
  public static java.lang.String extract (org.omg.CORBA.Any any) {...}
  public static org.omg.CORBA.TypeCode type () {...}
  public static java.lang.String id () {...}

public java.io.Serializable read_value (org.omg.CORBA.portable.InputStream _input) {
       java.lang.String result;
       result = _input.read_string();
       return (java.io.Serializable)result;
  }

public void write_value (org.omg.CORBA.portable.OutputStream _output, java.io.Serializable
value) {
       if (!(value instanceof java.lang.String)) {
            throw new org.omg.CORBA.MARSHAL();
       }
       java.lang.String valueType = (java.lang.String)value;
       _output.write_string(valueType);
}
public java.lang.String get_id () {
    return id();
  }
}

public final class BoxedStringHolder implements org.omg.CORBA.portable.Streamable {....}
```

Section 25.14.3.1 Example B

```
// IDL
struct idlStruct {
    short x;
};

module A {
    valuetype BoxedStruct idlStruct;
};

// generated Java
package A;
public final class BoxedStructHelper implements org.omg.CORBA.portable.BoxedValueHelper {
     private static final BoxedStructHelper _instance = new BoxedStructHelper();
     public static idlStruct read (final org.omg.CORBA.portable.InputStream _input) {
          if (!(_input instanceof org.omg.CORBA_2_3.portable.InputStream)) {
               throw new org.omg.CORBA.BAD_PARAM();
          }
        return
(idlStruct)((org.omg.CORBA_2_3.portable.InputStream)_input).read_value(_instance);
  }
```

```
public static void write (final org.omg.CORBA.portable.OutputStream _output, final idlStruct
value) {
     if (!(_output instanceof org.omg.CORBA_2_3.portable.OutputStream)) {
       throw new org.omg.CORBA.BAD_PARAM();
      }
      ((org.omg.CORBA_2_3.portable.OutputStream)_output).write_value(value, _instance);
 }

 public static void insert (final org.omg.CORBA.Any any, final idlStruct value) {...}

 public static idlStruct extract (final org.omg.CORBA.Any any) {...}

 public static org.omg.CORBA.TypeCode type () {...}

 public static java.lang.String id () {...}

public java.io.Serializable read_value (final org.omg.CORBA.portable.InputStream _input) {
      final idlStruct result;
      result = idlStructHelper.read(_input);
      return (java.io.Serializable)result;
 }

public void write_value (final org.omg.CORBA.portable.OutputStream _output, final
java.io.Serializable value) {
      if (!(value instanceof idlStruct)) {
           throw new org.omg.CORBA.MARSHAL();
      }
      idlStruct valueType = (idlStruct)value;
      idlStructHelper.write(_output, valueType);
 }

public java.lang.String get_id () {
   return id();
 }
}

public final class BoxedStructHolder implements org.omg.CORBA.portable.Streamable {
 ....
}
```

In Section 25.19.10 ORB in the definition of the **org.omg.CORBA_2_3.ORB** class:

Change the signature of **register_value_factory** to:

```
 public org.omg.CORBA.portable.ValueFactory register_value_factory(String id,
           org.omg.CORBA.portable.ValueFactory factory);
```

Change the return type **lookup_value_factory()**
  from: **org.omg.CORBA.portable.ValueHelper**

to: **org.omg.CORBA.portable.ValueFactory**

In Section 25.21.4

In the definition of the **org.omg.CORBA_2_3.portable.InputStream**:

Change both declarations of **read_Value** to **read_value**

In the 2nd overloaded read_Value change the parameter type
from: **ValueHelper helper**
to: **java.lang.String rep_id**

Change both declarations of **read_Abstract** to **read_abstract_interface**

Add the following new methods:

```
public java.io.Serializable read_value(org.omg.CORBA.portable.BoxedValueHelper factory) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

public java.io.Serializable read_value(java.io.Serializable) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

In the definition of the **org.omg.CORBA_2_3.portable.OutputStream**:

Change both declarations of **write_Value** to **write_value**

In the 2nd overloaded write_Value change the 2nd parameter type
from: **ValueHelper value**
to: **java.lang.String rep_id**

Change the declaration of **write_Abstract** to **write_abstract_interface**

Delete the **start_block** and **end_block** methods

Add the following new method:

```
public void write_value(java.io.Serializable value, org.omg.CORBA.portable.BoxedValueHelper
factory) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Actions taken: Close and incorporate changes
September 18, 1998: received issue
February 26, 1999: moved from obv_rtf to java rtf

Issue 1993: create_lname & create_lname_context (java-rtf)

Click here for this issue's archive.
Source: Applied Testing and Technology (Mr. James Pasley, james@aptest.ie)
Nature: Uncategorized Issue
Severity:
Summary: The naming service states that LName and LNameComponent are created in C/C++ using the create_lname and create_lname_context functions respectively. No mention is made as to how this is achieved in Java. Should it be possible to simply create an instance of the class using new, or is there a need for a factory class?
Resolution: These functions (library) has been deprecated by the newly adopted Interoperable Naming Service.
Revised Text:
Actions taken: Close, no action
September 23, 1998: received issue

Issue 2079: Potential problem with Java Mapping ORB init for Applet (java-rtf)

Click here for this issue's archive.
Nature: Uncategorized Issue
Severity:
Summary: While working on the revised interoperable naming specification, a java issue came up with regard to arguments to CORBA ORB_init. The PIDL reads as: module CORBA { typedef string ORBid; typedef sequence arg_list; ORB ORB_init(inout arg_list argv, in ORBid orb_identifier); } However the java mapping for init in an applet is: // public static ORB init(Applet app, Properties props); Using a property list for the argument list results in considerably different behavior than the PIDl sequence definition, mainly no preservation of sequence order and no repeating elements (properties).
Resolution: This is the proper design behavior. The use of sequence in PIDL does not necessarily imply any specific semantics with regards to such things. (There is no primitive set concept which could be used in PIDL. Each language mapping maps PIDL as it sees fit. The current specification is the natural mapping for Java.
Revised Text: none
Actions taken:Close, no action
October 14, 1998: received issue

Issue 2096: Issue with Mapping for Constants within an interface (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Kim Rochat, krochat@austin.ibm.com)
Nature: Uncategorized Issue
Severity:
Summary: In Chapter 25 (IDL/Java) of CORBA V2.3-1.3 (Aug 1998), section 25.5.1, "Constants Within An Interface", the text says "Constants declared within an IDL interface are mapped to fields in the Java interface corresponding to the IDL interface". The problem is that there are now two Java

interfaces produced for an IDL interface, the Operations interface and the Signature interface. The current client programming model for accessing constants within an interface is to say "t.c" (instead of "tOperations.c"). Therefore, in order to avoid changing the client programming mode, I propose that section 25.5.1 be revised to say that constants within an IDL interface "are mapped to fields in the Java signature interface..."
Resolution: This is an editorial issue, that got missed when updating the spec to take into account the split into the signature and operations interfaces. The editor has already made the change.
Revised Text:
Actions taken: Close, no action
October 19, 1998: received issue

Issue 2228: typedefs in IDl when generating Java code (java-rtf)

Click here for this issue's archive.
Nature: Uncategorized Issue
Severity:
Summary: A question came up regarding the use of typedefs in idl when generating Java >code. Since Java has no support for typedefs the alias is not available to >any Java code other than the idl generated code. This seems to be a flaw in >the design of the idl to Java mapping specification.
Resolution: The helper is generated for all aliases, it can be used if needed.
Revised Text:none
Actions taken:Close, no action
November 25, 1998: received issue

Issue 2233: Helper "narrow" exception clarification (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Kim Rochat, krochat@austin.ibm.com)
Nature: Uncategorized Issue
Severity:
Summary: Referring to 23_java-2_3base.pdf, section 23.11.1, page 23-26, second paragraph, "The IDL exception CORBA::BAD_PARAM is thrown if the narrow fails." This statement can be interpreted in two different ways. Does it mean that BAD_PARAM is thrown no matter what goes wrong in the narrow call, or that BAD_PARAM is thrown only when the parameter doesn't support the interface being narrowed to?
Resolution: The latter. A different exception shall be raised to indicate other errors.
Revised Text: In Section 23.12.1 Basics:
Replace the last sentence of the 3rd paragraph:
The IDL exception CORBA::BAD_PARAM is thrown if the narrow fails.
with:
The IDL exception CORBA::BAD_PARAM shall be thrown if the narrow fails because the object reference does not support the requested type. A different system exception shall be raised to indicate other kinds of errors. Trying to narrow a null will always succeed with a return value of null.
Actions taken: Close, incorporate text
December 1, 1998: received issue

Issue 2251: Java LocalStub (java-rtf)

Click here for this issue's archive.
Source: Object-Oriented Concepts (Mr. Matthew Newhook, matthew@ooc.com)
Nature: Uncategorized Issue
Severity:
Summary: From IDL/Java specification (draft version 3.4 - August 9): The remote stub must be named _Stub where is the IDL interface name this stub is implementing. The local stub must be named _LocalStub whereis the same IDL interface name. Local stubs are defined to be direct subclasses of remote stubs. and: The _is_local() method is provided so Helper classes may determine if a particular object is implemented by a local servant and if so create a local stub, rather than a remote stub. This function would typically be called from within the narrow() method of a Helper class. The _is_local() method may only return true if the servant incarnating the object is located in the same Java VM. The method may return false if the servant is not local or the ORB does not support local stubs for that particular servant. The default behavior of _is_local() is to return false. The design described in the specification seems to make it very difficult, if not impossible, to do the following: - Create a proxy that refers to some remote servant - Upon a request to the remote servant, a LocationForward is generated that refers to a colocated servant.
Resolution:
Merge the two stub classes together as one class with as minimal impact on the APIs and performance.
Revised Text:
Replace the first five paragraphs of section 25.21.5.1 "Stub/Skeleton Architecture" with the following text:

"The mapping defines a single stub which may be used for both local and remote invocation. Local invocation provides higher performance for collocated calls on Servants located in the same process as the client. Local invocation is also required for certain IDL types which contain parameter types which cannot be marshalled remotely. Remote invocation is used to invoke operations on objects which are located in an address space separate from the client.

While a stub is using local invocation it shall provide complete location transparency. To provide the correct semantics, compliant programs shall comply with the parameter passing semantics defined in Section 25.11.2, "Parameter Passing Modes". When using local invocation the stub shall copy all valuetypes passed to them, either as in parameters, or as data within in parameters, and shall pass the resulting copies to the Servant in place of the originals. The valuetypes shall be copied using the same deep copy semantics as would result from GIOP marshaling and unmarshaling.

The following sections describe the characteristics of the stubs and skeletons. The examples are based on the following IDL:"

In Section 25.21.5.1.1 "Stub Design" replace the second paragraph with the following:

"The stub shall be named _<in**terface_name>Stub** where **<interface_name>** is the IDL interface name this stub is implementing.

Stubs shall support both local invocation and remote invocation, except in the following cases:

1. The stub is implementing an IDL interface which may only be invoked locally (e.g. ***PortableServer::POA***). In this case, the stub may choose to implement only local invocation.

"

In Section 25.21.5.1.1, remove the fourth paragraph.

Replace the example in Section 25.21.5.1.3 "Stream-based Stub example" with the following:

```
package Example;

public class _AnInterfaceStub extends org.omg.CORBA.portable.ObjectImpl implements
AnInterface {

  public java.lang.String[] _ids () {
    return __ids;
  }

  private static java.lang.String[] __ids = {
    "IDL:Example/AnInterface:1.0"
  };

  final public static java.lang.Class _opsClass = Example.AnInterfaceOperations.class;

  public int length (java.lang.String s) throws  Example.AnException {
    while(true) {
      if(!this._is_local()) {
        try {
            org.omg.CORBA.portable.OutputStream _output = this._request("length", true);
            _output.write_string(s);
            org.omg.CORBA.portable.InputStream _input = this._invoke(_output);
            return  _input.read_long();
        }
        catch (org.omg.CORBA.portable.RemarshalException _exception) {
            continue;
        }
        catch (org.omg.CORBA.portable.ApplicationException _exception) {
            java.lang.String _exception_id = _exception.getId();
            if (_exception_id.equals(Example.AnExceptionHelper.id())) {
                throw Example.AnExceptionHelper.read(_exception.getInputStream());
            }
            throw new org.omg.CORBA.UNKNOWN("Unexpected User Exception: " +
_exception_id);
        }
        finally {
```

```
            this._releaseReply(_input);
        }
      }
      else {
        org.omg.CORBA.portable.ServantObject _so = _servant_preinvoke("length", _opsClass);
        if (_so == null) {
            continue;
        }
        Example.AnInterfaceOperations _self = (Example.AnInterfaceOperations)_so.servant;
        try {
            return _self.length(s);
        }
        finally {
            _servant_postinvoke(_so);
        }
      }
    }
  }
}
```

In Section 25.21.5.2 "Stub and Skeleton Class Hierarchy", Figure 25-1,
remove the class "**_FooLocalStub**".

In Section 25.21.5.3.1 "Streaming Stub APIs" replace the second paragraph
with the following:

"The method **_invoke()** is called to invoke an operation. The stub
provides an OutputStream that was previously returned from a
_request() call. The method **_invoke()** returns an InputStream which
contains the marshalled reply. The **_invoke()** method may throw only one of the following:
an ApplicationException, a RemarshalException, or a CORBA system
exception. An ApplicationException shall be thrown to indicate the target
has raised a CORBA user exception during the invocation. The stub may
access the **InputStream** of the **ApplicationException** to unmarshal the
exception data. A **RemarshalException** shall be thrown if the stub was
redirected to a different target object and remarshalling is
necessary, this is normally due to a GIOP object forward or locate
forward messages. In this case, the stub shall then attempt to reinvoke the request on
behalf of the client after verifying the target is still remote by
invoking **_is_local()** (see section 25.21.5.3.2). If _is_local() returns True, then an attempt to reinvoke the request
using the Local Invocation APIs shall be made. If a CORBA system
exception is thrown, then the exception shall be passed on directly to the
user."

Rename section 25.21.5.3.2 "Local Stub APIs" to "Local Invocation APIs".
Replace the first three paragraphs with the following:

"Local invocation is supported by the following methods and classes:

The **_is_local()** method is provided so stubs may determine if a particular object is implemented by a local servant and hence local invocation APIS may be used. The **_is_local()** method shall return true if the servant incarnating the object is located in the same process as the stub and they both share the same ORB instance. The **_is_local()** method returns false otherwise. The default behavior of **_is_local()** is to return false.

The **_servant_preinvoke()** method is invoked by a local stub to obtain a Java reference to the servant which should be used for this request. The method takes a string containing the operation name and a Class object representing the expected type of the servant as parameters and returns a ServantObject object (Note, ORB vendors may subclass the ServantObject object to return additional request state that may be required by their implementations). The operation name corresponds to the operation name as it would be encoded in a GIOP request. The expected type shall be the Class object associated with the operations class of the stub's interface (e.g. A stub for an interface Foo, would pass the Class object for the FooOperations interface). The method shall return a null value if the servant is not local or the servant has ceased to be local as a result of the call (i.e, due to a ForwardRequest from a POA ServantManager). The method shall throw ***CORBA::BAD_PARAM*** if the servant is not of the expected type. If a ServantObject object is returned, then the servant field shall have been set to an object of the expected type (Note, the object may or may not be the actual servant instance). The local stub may cast the servant field to the expected type, and then invoke the operation directly. The ServantRequest object is valid for only one invocation, and cannot be used for more than one invocation."

Actions taken:Close, and incorporate text.
December 10, 1998: received issue

Issue 2255: Proposal for persistent valuetypes (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Mr. Simon C. Nash, nash@hursley.ibm.com)
Nature: Uncategorized Issue
Severity:
Summary: Here is my proposal for how to solve the persistent valuetype problem. It allows valuetypes to be used for PSS with getter and setter access to the data, without changing the OBV Java language bindings or adding new keywords or pragmas to IDL. Any additions to IDL that may be desirable for improved functionality (e.g, query and finder support) could be designed jointly by the components submission group and the PSS group as part of some future RTF activity. The proposal is very simple. It is that persistent valuetypes must be abstract valuetypes with attributes. The attributes would be used to specify the persistent state of the object.
Resolution: Persistent valuetypes are part of a proposed submission to the PSS RFP which has not yet been adopted. If and when such a submission is adopted as an OMG specification, it MAY be

appropriate to resubmit this, or a similar issue. Until then, even the filing of this issue is premature at best.
Revised Text:
Actions taken: Close, no action.
December 15, 1998: received issue

Issue 2256: PSS requiremnets for the ObV/Java mapping (java-rtf)

Click here for this issue's archive.
Source: IONA (Mr. Bernard Normier, bnormier@iona.com)
Nature: Uncategorized Issue
Severity:
Summary: The latest ObV/Java mapping, including George's latest proposal, is not usable for values with a persistent implementation. I'd really like to solve this issue with the Java & ObV RTF, rather than solve it through the adoption of a new spec that updates the Java mapping. ==== Issue - ObV/Java maps value types's data members to Java data members - providing a persistent implementation for such data members would require byte-code post-processing or pre-processing, or yet-to-be-specd-and-agreed JVM hooks; we do not see this as viable/ realistic options.
Resolution: Persistent valuetypes are part of a proposed submission to the PSS RFP which has not yet been adopted. If and when such a submission is adopted as an OMG specification, it MAY be appropriate to resubmit this, or a similar issue. Until then, even the filing of this issue is premature at best.
Revised Text:
Actions taken: Close, no action
December 15, 1998: received issue

Issue 2462: BAD_PARAM issue (java-rtf)

Click here for this issue's archive.
Source: International Business Machines (Kim Rochat, krochat@austin.ibm.com)
Nature: Uncategorized Issue
Severity:
Summary: This issue is with the 1998-08-22 "Mapping of OMG IDL to Java". ptc/1998-08-22 (the IDL/Java mapping), section 25.3.5, says that if a bounded string is too long, MARSHAL is thrown. Meanwhile, ptc/1998-12-04 (2.3a Core chapters 1-15), section 3.17.1, says that MARSHAL is to be used for "A request or reply is structurally invalid ...indicates a bug in the runtime". Meanwhile, BAD_PARAM says "a parameter passed to a call is out of range or otherwise illegal..." Based on these descriptions, I propose that BAD_PARAM should be thrown in Java when a string exceeds its bound rather than MARSHAL.
Resolution: Make it so.
Revised Text: In section 25.4.5 Strings replace the 2 occurrences of MARSHAL with BAD_PARAM.
Actions taken:Close, incorporate changes
February 22, 1999: received issue

Issue 2514: PortableServer Servant issue (java-rtf)

Click here for this issue's archive.
Source: Inprise Corporation (Mr. George M. Scot, gscott@inprise.com gscott@visigenic.com)
Nature: Uncategorized Issue
Severity:
Summary: The currently specified definition of the org.omg.CORBA.PortableServer.Servant class does not correctly implement the behavior specified. In particular, the POA returned by PortableServer::Current::get_POA is always used during the processing of _this_object() even if _this_object() is not being called on the Servant which is currently being invoked as part of the request. This contradicts the behavior defined in the first bullet item for _this_object as defined in section 25.19.2.1 "Mapping of PortableServer::Servant". Also, the overall behavior of _this() may need some tweaking (see my other email "Behavior of POA Servant's _this()"), which may require us to revisit the definition of the Servant class.
Resolution:

The current definition of **org.omg.PortableServer.Servant** does not correctly implement the specification defined for **Servant** as defined by the Java mapping. In particular, it does not correctly implement the semantics of bullet item number one defined for **_this_object()**. Furthermore, the current definition of Servant hard codes certain functionality and prevents vendors from making fixes or otherwise changing the implementation.

To fix this problem, a delegation model for the Servant class similar to that used by stubs is proposed in order to allow vendors full control over the way that certain Servant operations are handled. We believe this is necessary to correctly implement the **_this_object()** method for which a correct implementation using the currently defined public POA APIs is somewhat problematic. The proposal also allows vendors flexibility in the implementation of the methods way which may allow for better optimization opporutunities.

The proposed changes have minimum impact on end user applications as well as minimal impact on ORB vendors (the Visibroker ORB was converted to the new design in less than a day including time to implement **_this_object()**).

The only API change end users will see is the removal of the method:

void _orb(org.omg.CORBA.ORB orb);

This method does not work well with the new delegation model, and it has been our experience that this method is rarely, if ever, used. The only need a user ever has to set the ORB on a Servant is for the case of implicit activation, and the prefered way is to use the method:

org.omg.CORBA.Object _this_object(org.omg.CORBA.ORB orb);

The **_this_object()** method is still provided, but now sets the delegate instead of setting a locally stored ORB variable. We believe this method is required and also commonly used so should remain part of the user API.

The proposal defines a new **Delegate** interface which is defined in a new **package org.omg.PortableServer.portable** and mirrors the design pattern used by the Stub classes for the Java mapping. Unlike stubs, however, the Servant must have a way of setting its delegate given an instance of the ORB in order to support implicit activation via **_this()**. To accomplish this our proposal includes the addition of a new method on the ORB class:

public void set_delegate(java.lang.Object object);

This method will actually appear on **org.omg.CORBA_2_3.ORB** due to the **org.omg.CORBA.ORB** class being frozen in the Java 2 core. This method takes an object which is known to the ORB to follow the delegation model and sets it delegate. In this version of the proposal the only type that an ORB is required to support setting the delegate is **org.omg.PortableServer.Servant**. However, we expect in the future it may be possible to make use of this method to set delegates on stubs which have been reincarnated through Java serialization.

Note, we also chose to make the **Delegate** an interface rather than an abstract class to provide for greater flexibility in the implementation.

This proposal as it fixes an important piece of POA functionality and is required for correct operation of the POA.

George Scott, gscott@inprise.com
Revised Text:

In Section 25.19.10 "ORB". Add the following new method to **org.omg.CORBA_2_3.ORB**:

```
public abstract class ORB extends org.omg.CORBA.ORB {
    abstract public void set_delegate(java.lang.Object wrapper);
}
```

Add the following paragraph following the class definition for org.omg.CORBA_2_3.ORB:

Add a new section: 25.19.10.1 set_delegate at the end of section 25.19.10

"The **set_delegate()** method supports the Java ORB portability interfaces by providing a method for classes which support ORB portability through delegation to set their delegate. This is typically required in cases where instances of such classes were created by the application programmer rather than the ORB runtime. The wrapper parameter is the instance of the object on which the ORB must set the delegate. The mechanism to set the delegate is specific to the class of the wrapper instance. The **set_delegate()** method shall support setting delegates on instances of the following Java classes:

- org.omg.PortableServer.Servant

If the wrapper paramter is not an instance of a class for which the ORB
can set the delegate, the CORBA::BAD_PARAM exception shall be thrown."

In Section 25.20.2.1 "Mapping of PortableServer::Servant", reformat the methods and group them to clarify their use as follows:

Add the comment below and group the following 5 methods
// Convenience methods for application programmer
final public org.omg.CORBA.Object _this_object() {...}
final public org.omg.CORBA.Object _this_object(ORB orb) {...}
final public org.omg.CORBA.ORB _orb() {...}
final public POA _poa() {...}
final public byte[] _object_id() {...}

Add the comment below and group the following 4 methods:
// Methods which may be overriden by the application programmer
public POA _default_POA() {...}
public boolean _is_a(String repository_id) {...}
public boolean _non_existent() {...}
public org.omg.CORBA.InterfaceDef _get_interface() {...}

Add the comment below and group the following 1 methods:
// Methods for which the skeleton or application programmer must
// provide an implementation
abstract public String[] _all_interfaces(POA poa, byte[] objectId);

Delete the following method:
final public void _orb(ORB orb) {...}

Replace the following paragraph which reads as "The Servant class is an abstract Java class which serves as the base classfor all POA Servant implementations." with the following 3 paragraphs:

"The Servant class is an abstract Java class which serves as the base classfor all POA Servant implementations. It provides a number of methods
which may be invoked by the application programmer, as well as methods
which are invoked by the POA itself and may be overridden by the user to
control aspects of Servant behavior.

With the exception of the **_all_interfaces()** and **_this_object(ORB orb)**
methods, all methods defined on the **Servant** class may only be invoked
after the **Servant** has been associated with an ORB instance.
Attempting to invoke the methods on a **Servant** which has not been
associated with an ORB instance shall result in a **CORBA::BAD_INV_ORDER**
exception being raised. A **Servant** is associated with an ORB instance
via one of the following means:
1. Through a call to **_this_object(ORB orb)** passing an ORB instance as

parameter. The **Servant** will become associated with the specified ORB instance.
2. By explicitly activating a Servant with a POA by calling either **POA::activate_object** or **POA::activate_object_with_id**. Activating a **Servant** in this fashion will associate the Servant with the ORB instance which contains the POA on which the Servant has been activated.
3. By returning a **Servant** instance from a ServantManager. The Servant returned from **PortableServer::ServantActivator::incarnate()** or **PortableServer::ServantLocator::preinvoke()** will be associated with the ORB instance which contains the POA on which the ServantManager is installed.
4. By installing the Servant as a default servant on a POA. The Servant will become associated with the ORB instance which contains the POA for which the Servant is acting as a default servant.

It is not possible to associate a Servant with more than one ORB instance at a time. Attempting to associate a Servant with more than one ORB instance will result in undefined behavior.

Replace Section 25.20.2.1.2 _orb with:

The **_orb()** method is a convenience method which returns the instance of the ORB which is currently associated with the Servant."

In Section 25.20.2.1.1 _this_object
Replace the fourth bullet item with the following text:

" * The **_this_object(ORB orb)** method first associates the **Servant** with the specified ORB instance and then invokes **_this_object()** as normal."

In Section 25.21.2 "Overall Architecture" Add the following bullet after the "Portable Delegate" bullet:

" * Portable Servant Delegate - provides the vendor specific implementation of PortableServer::Servant"

In Section 25.21.2.1 "Portability Package" Change the first sentence to the following:

"The APIs needed to implement portability are found in the **org.omg.CORBA.portable** and **org.omg.PortableServer.portable** packages."

Add the following new sections after section 25.21.6 "Delegate":
(ORB becomes 25.21.9)

"
25.21.7 "Servant"

The Servant class is the base class for all POA-based implementations.

It delegates all functionality to the Delegate interface defined in section 25.21.6.

```
package org.omg.PortableServer;

import org.omg.CORBA.ORB;
import org.omg.PortableServer.portable.Delegate;

abstract public class Servant {

  private transient Delegate _delegate = null;

  final public Delegate _get_delegate() {
    if (_delegate == null) {
      throw new org.omg.CORBA.BAD_INV_ORDER("The Servant has not been associated with
an ORB instance");
    }
    return _delegate;
  }

  final public void _set_delegate(Delegate delegate) {
    _delegate = delegate;
  }

  final public org.omg.CORBA.Object _this_object() {
    return _get_delegate().this_object(this);
  }

  final public org.omg.CORBA.Object _this_object(ORB orb) {
    try {
      ((org.omg.CORBA_2_3.ORB)orb).set_delegate(this);
    }
    catch(ClassCastException e) {
      throw new org.omg.CORBA.BAD_PARAM("POA Servant requires an instance of
org.omg.CORBA_2_3.ORB");
    }
    return _this_object();
  }

  // access to the ORB
  final public ORB _orb() {
    return _get_delegate().orb(this);
  }

  // convenience methods to the POA Current
  final public POA _poa() {
    return _get_delegate().poa(this);
  }
```

```
  final public byte[] _object_id() {
    return _get_delegate().object_id(this);
  }

  // Methods which may be overriden by the user
  public POA _default_POA() {
    return _get_delegate().default_POA(this);
  }

  public boolean _is_a(String repository_id) {
    return _get_delegate().is_a(this, repository_id);
  }

  public boolean _non_existent() {
    return _get_delegate().non_existent(this);
  }

  public org.omg.CORBA.InterfaceDef _get_interface() {
    return _get_delegate().get_interface(this);
  }

  // methods for which the user must provide an implementation
  abstract public String[] _all_interfaces(POA poa, byte[] objectId);
}
```

25.21.8 "Servant Delegate"

The Delegate interface provides the ORB vendor specific implementation of **PortableServer::Servant**.

```
package org.omg.PortableServer.portable;

import org.omg.PortableServer.Servant;
import org.omg.PortableServer.POA;

public interface Delegate {
  org.omg.CORBA.ORB orb(Servant self);
  org.omg.CORBA.Object this_object(Servant self);
  POA poa(Servant self);
  byte[] object_id(Servant self);
  POA default_POA(Servant self);
  boolean is_a(Servant self, String repository_id);
  boolean non_existent(Servant self);
  org.omg.CORBA.InterfaceDef get_interface(Servant self);
}
```

Rename Section 25.21.6 "Delegate" to "Stub Delegate".

Actions taken: Close, incorporate text

March 5, 1999: received issue