RFC 62 Metatype Update

Final

25 Pages

Abstract

This RFC proposes a MetaType service that simplifies the usage of the Metatype specification. The MetaType service reads configuration information from installed bundles (specified as an XML resource) and uses this information to provide the Metatype objects that describe the configuration data as specified by the Metatype specification.

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

`Source code is shown in this typeface.`

0.3 Revision History

The last named individual in this history is currently responsible for this document.

All Page Within This Box

Revision	Date	Comments
Initial	Feb 24 2003	Initial creation Pam Tobias, pam_tobias@us.ibm.com
Ported	Jul 15 2003	Peter Kriens, ported to new template and augmented
3rd Draft	13 Aug 2004	Updated. Does not include all input from e-mails and RFC 69. These were considered but I did not include many things because I did not agree with or understand them. We can now use this document to continue discussion. BJ Hargrave, hargrave@us.ibm.com
4th Draft	1 Sep 2004	Added icon attribute to list of localized values. BJ Hargrave, hargrave@us.ibm.com
Draft 5	5 October 2004	Major update to the RFC. 1. New metatype2 package replaces the old metatype package which is now deprecated. This allows many improvements to the API. 2. New XML schema to support the new API. 3. Javadoc for the new metatype2 API. BJ Hargrave, hargrave@us.ibm.com
Draft 6	20 October 2004	Removed new metatype2 package and reverted to be a metatype for configuration specification. Update the XML Schema and the javadoc. BJ Hargrave, hargrave@us.ibm.com
Draft 7	19 November 2004	Changed the XML schema so that RFC 62 and RFC 94 (both of which deal with configuration) share a common XML schema. Joe Rusnak, jgrusnak@us.ibm.com
Draft 8	7 February 2005	Changed the XML schema to be much simpler. Made a few changes/clarifications to introductory text in Sections 1 and 2. Added a section describing the relationship with RFC 94, and why the two RFCs should share a common XML schema definition. Joe Rusnak, jgrusnak@us.ibm.com
Draft 9	10 February 2005	Fixed an error in the XML schema (missing “localization” attribute). Clarified some “minOccurs” and “maxOccurs” stuff… Renamed two attributes (“ocdref” and “adref”) to clarify they are references. Fixed a mistake in the example XML file. Joe Rusnak, jgrusnak@us.ibm.com

All Page Within This Box

Revision	Date	Comments
Draft 9.1	14 February 2005	Updated the XML schema to include a new attribute used only for RFC 94.
Final Review Draft	20 Mar 2005	Change META-INF to OSGI-INF per Munich CPEG meeting. BJ Hargrave, hargrave@us.ibm.com
FRD+1	21 Mar 2005	Minor changes to the XML schema based on feedback from the RFC 62 and RFC 94 Reference Implementations. Added an explanation of how to handle the “default” attribute when “cardinality” is non-zero. Added an explanantion of how to interpret the “min” and “max” attributes for a String type. Added a “required” attribute to the “AD” element. Joe Rusnak, jgrusnak@us.ibm.com
Final	27 May 2005	No changes. BJ Hargrave, hargrave@us.ibm.com

1 Introduction

This RFC was written because in OSGi R3 it is cumbersome to write bundles that provide meta type information. This is because the OSGi R3 Metatype Specification only provides a number of Java interfaces, leaving the details to the implementations. By providing a central service and an XML format for metatype information, it becomes possible to provide the metatype information as a resource instead of including implementation classes for the Metatype interfaces. The declarative XML approach described in this document is easier than the programmatic approach currently in the OSGi R3 Metatype Specification.

2 Application Domain

The OSGi Metatype Specification allows a bundle to provide type information about data it can consume. This data is a set of key-value pairs. One important (but not the only) application is configuring bundles.

The Configuration Admin Service allows *administrative bundles* to configure other *configurable bundles*. Each bundle can register one or more ManagedService and ManagedServiceFactories (i.e. *configuration targets*) that represent some configuration point. A point is identified with a persistent id (PID).

All Page Within This Box

The administrative bundle can have *a priori* knowledge of the configurable bundles. In that case, it can hard code the user interface for these bundles. However, the OSGi environment is very dynamic, and for this reason the Metatype Specification was created so that the administrative bundle can generate a configuration user interface on the fly based upon metatype information.

The configuration targets can provide type information for the configuration objects they expect to receive. If a configuration target implements the `MetaTypeProvider` interface, a client can use this interface to get the appropriate `ObjectClassDefinition` for the configuration target. This `ObjectClassDefinition` object contains enough information for the administrative bundle to create a UI automatically.

This is just one example of how the Metatype Specification can be used – it is a general purpose mechanism that can be used for purposes other than configuration.

3 Problem Description

In OSGi R3, bundles that want to provide metatype information must carry implementation classes for the Metatype interfaces. In other words, the metatype information must be provided programmatically. These implementation classes must obtain the actual meta type information in an implementation specific way. This is non-trivial and the result has been that few bundles have provided metatype information.

4 Requirements

1. Provide a mechanism that makes it very simple for a bundle to provide metatype information
2. Support the full Metatype Specification, including support for locales.
3. Allow for the separation of the bundle providing the meta type information and the bundle providing the configuration target.
4. Have a common XML schema definition used by both RFC 62 and RFC 94 [5], both of which deal with configuration.

All Page Within This Box

5 Technical Solution

5.1 MetaType Service

The MetaType Service provides a mechanism to read a resource from a bundle containing metatype information and return metatype objects representing that information.

An administrative bundle can call the MetaType Service with a Bundle argument to obtain a MetaTypeInformation object for the bundle. The MetaTypeInformation object provides methods to interrogate the meta type information of the bundle.

If the subject bundle does not contain any meta type documents, then the MetaType Service will query whether the bundle has registered any ManagedService or ManagedServiceFactory services which implement MetaTypeProvider. If so, then the MetaType Service will return a MetaTypeInformation object which wrappers these MetaTypeProvider objects. Thus the MetaType Service can be used to retrieve meta type information for bundles which contain a meta type resource or which provide their own MetaTypeProvider objects.

5.1.1 Example

The MetaTypeService implementation class is the main class. It registers the org.osgi.service.metatype.MetaTypeService service and has a method to get a MetaTypeInformation object for a bundle.

Following is some sample code demonstrating how to print out all the ObjectClassDefinitions and AttributeDefinitions contained in a bundle:

```
Bundle someBundle;

ServiceTracker tracker = new ServiceTracker(context,
   MetaTypeService.class.getName(), null);

tracker.open();

MetaTypeService mts = (MetaTypeService) tracker.getService();

MetaTypeInformation mti = mts.getMetaTypeInformation(someBundle);

String [] pids = mti.getPids();

String [] factoryPids = mti.getFactoryPids();

for (int i=0; i<pids.length; i++) {

     ObjectClassDefinition ocd = mti.getObjectClassDefinition(pids[i], null);

     AttributeDefinitions[] ADs = ocd.getAttributeDefinitions(ALL);

     for (int j=0; j<ADs.length; j++) {
```

All Page Within This Box

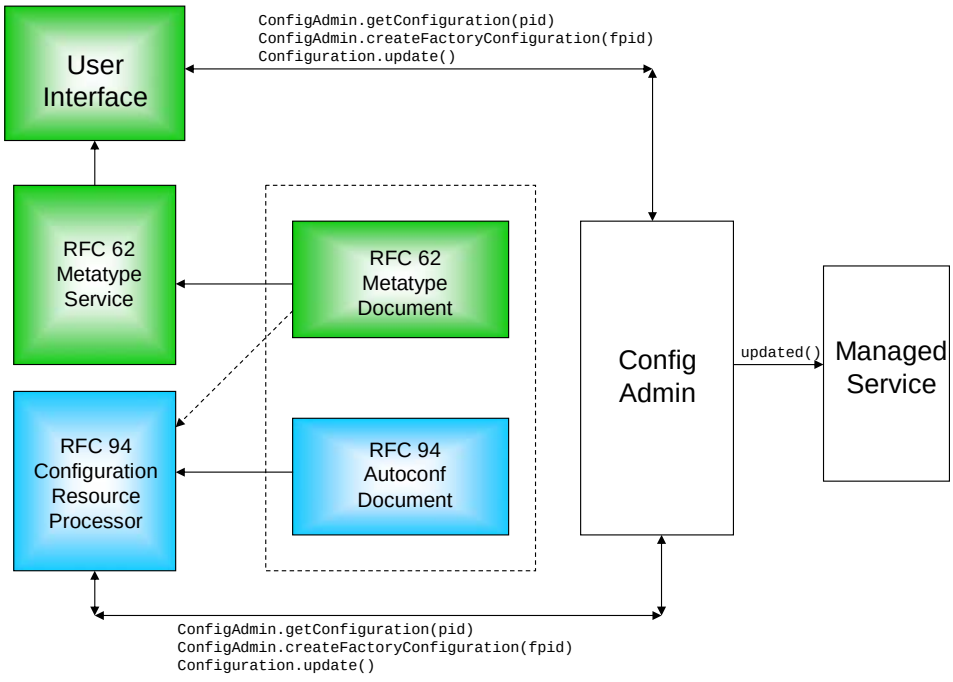
```
            System.out.println("OCD="+ocd.getName()+"; AD="+ADs[j].getName());

        }

    }

    tracker.close();
```

5.2 MetaType Document

The following shows the format of the metatype document. Each configurable bundle should have a OSGI-INF/metatype/ directory in the bundle. All resources in that directory must be meta type documents and the MetaType Service will process each file for meta type information.

5.2.1 Relationship to RFC 94

As shown in the diagram below, the OSGi Metatype Service (RFC 62) and the MEG Configuration Resource Processor (RFC 94) are related. Both use XML documents to provide configuration information to managed services through the Configuration Admin service. Because of their relationship, these two services share a common XML schema definition



As previously discussed in Section 2, the Metatype Service gives a declarative way to specify type information about groups of attributes representing configuration parameters for a managed service. Typically, the file containing the metatype information is a resource that accompanies the bundle providing the managed service.

All Page Within This Box

OSGi MEG RFC 94 defines a Configuration Resource Processor that processes an XML document (AUTOCONF.XML) that also contains type information and values for the configuration parameters. This document might be associated with a MEG deployment package (see RFC 88) that wants to use a managed service, and needs it configured properly. In other words, an AUTOCONF.XML file is not a resource in the bundle providing the managed service. As shown in the diagram above, an AUTOCONF.XML file might be completely self-contained, incorporating both type definition and values. On the other hand, an AUTOCONF.XML file might simply provide the values for configuration parameters, and use the type definitions provided by a Metatype Service document (this is depicted by the dotted arrow in the diagram).

Some of the elements and attributes in the shared XML schema (Section 5.2.3) are used only by RFC 94, some are used only by RFC 62, and some are used by both.

5.2.2 Localization

The MetaType Service will support localization of the name, icon, description and label attributes using the same mechanisms described in RFC 74 [3]. The ***localization*** attribute of the ***metatype*** element can be used to specify a different localization file base name. If this attribute is not specified, then the defaults as specified in RFC 74 are used.

The MetaType Service must examine the bundle and its fragments to locate all localization resources for the localization basename. From that list, the MetaType Service can derive the list of locales which are available for the meta type information. This list can then be returned by MetaTypeInformation.getLocales.

5.2.3 XML Schema

It is not intended that a metatype document be validated against the schema by the MetaType Service. The schema can be used by tools and external management systems.

The master version of the XML Schema is in membercvs [4]. The XML Namespace for meta type documents is:

```
http://www.osgi.org/xmlns/metatype/v1.0.0
```

The XML grammar specified by the schema can be easily parsed by very small XML parsers. For example, the JSR 172 (J2ME Web Services) XML parser is very small (~50KB) and namespace aware and can parse component descriptions.

The portion of the schema used by RFC 62 is shown in the bold text below. Any other parts of the schema that are encountered should be ignored by an RFC 62 implementation.

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.osgi.org/xmlns/metatype/v1.0.0"
   xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0">

   <complexType name="MetaData">
      <sequence>
         <element name="OCD" type="metatype:OCD" minOccurs="0" maxOccurs="unbounded" />
         <element name="Designate" type="metatype:Designate" minOccurs="1" maxOccurs="unbounded" />
      </sequence>
      <attribute name="localization" type="string" use="optional" />
   </complexType>

   <complexType name="OCD">
      <sequence>
```

All Page Within This Box

```
      <element name="AD" type="metatype:AD" minOccurs="1" maxOccurs="unbounded" />
      <element name="Icon" type="metatype:Icon" minOccurs="0" maxOccurs="1" />
   </sequence>
   <attribute name="name" type="string" use="required" />
   <attribute name="description" type="string" use="optional" />
   <attribute name="id" type="string" use="required" />
</complexType>

<complexType name="AD">
   <sequence>
      <element name="Option" type="metatype:Option" minOccurs="0" maxOccurs="unbounded" />
   </sequence>
   <attribute name="name" type="string" use="optional" />
   <attribute name="description" type="string" use="optional" />
   <attribute name="id" type="string" use="required" />
   <attribute name="type" type="metatype:Scalar" use="required" />
   <attribute name="cardinality" type="integer" use="optional" default="0" />
   <attribute name="min" type="string" use="optional" />
   <attribute name="max" type="string" use="optional" />
   <attribute name="default" type="string" use="optional" />
   <attribute name="required" type="boolean" use="optional" default="true" />
</complexType>

<complexType name="Object">
   <sequence>
      <element name="Attribute" type="metatype:Attribute" minOccurs="0" maxOccurs="unbounded" />
   </sequence>
   <attribute name="ocdref" type="string" use="required" />
</complexType>

<complexType name="Attribute">
   <sequence>
      <element name="Value" type="string" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="adref" type="string" use="required" />
   <attribute name="content" type="string" use="optional" />
</complexType>

<complexType name="Designate">
   <sequence>
      <element name="Object" type="metatype:Object" minOccurs="1" maxOccurs="1" />
   </sequence>
   <attribute name="pid" type="string" use="required" />
   <attribute name="factory" type="boolean" use="optional" default="false" />
   <attribute name="bundle" type="string" use="optional" />
   <attribute name="optional" type="boolean" default="false" use="optional" />
   <attribute name="merge" type="boolean" default="false" use="optional" />
</complexType>

<simpleType name="Scalar">
   <restriction base="string">
      <enumeration value="String" />
      <enumeration value="Long" />
      <enumeration value="Double" />
      <enumeration value="Float" />
      <enumeration value="Integer" />
      <enumeration value="Byte" />
      <enumeration value="Char" />
      <enumeration value="Boolean" />
      <enumeration value="Short" />
   </restriction>
</simpleType>

<complexType name="Option">
   <attribute name="label" type="string" use="required" />
   <attribute name="value" type="string" use="required" />
</complexType>

<complexType name="Icon">
```

```
      <attribute name="resource" type="string" use="required" />
      <attribute name="size" type="integer" use="required" />
   </complexType>

   <element name="MetaData" type="metatype:MetaData" />

</schema>
```

5.2.4 Example Resource

5.2.4.1 OSGI-INF/metatype/metatype.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd ">

  <OCD id="ocd1" name="%ocdname" description="%ocddescription">
   <AD id="" name="%adname" type="String" cardinality="0" default="default value"
description="%addescription">
      <Option label="%label1" value="value 1" />
      <Option label="%label2" value="value 2" />
   </AD>
   <Icon size="16" resource="%iconfile" />
  </OCD>

  <Designate pid="org.osgi.example.somebundle.pid" factory="false">
   <Object ocdref="ocd1"/>
  </Designate>

</metatype:MetaData>
```

5.2.4.2 OSGI-INF/l10n/bundle_en_US.properties

```
ocdname=Object Class Definition Name
ocddescription=Object Class Definition Description
adname=Attribute Definition Name
addescription=Attribute Definition Description
label1=Choice number 1
label2=Choice number 2
iconfile=icons_en_US/x.ico
```

5.2.5 Some Details Relating to the Schema

5.2.5.1 Interpreting the “default” attribute when the “cardinality” element is non-zero

When the “cardinality” element on the “AD” element is non-zero, the “default” attribute (if present) is interpreted as a comma-delimited list of values. For example,

```
default=”1.1,2.2,3.3”
```

denotes a set of 3 values: 1.1, 2.2, and 3.3. Whether this is an array or a vector is determined by whether the “cardinality” attribute is greater than zero or less than zero.

All Page Within This Box

If one or more of the items is a String that contains a comma, the backslash (\) is used as an escape character. A comma is represented as "\,". The backslash itself is represented as two backslashes. So, for example, the set of strings "a,b" , "b,c", "c\" and "d" would be represented as:

```
default=""a\,b,b\,c,c\\,d"
```

5.2.5.2 Interpreting the "min" and "max" elements when the "type" attribute is "String"

The "min" and "max" elements should be interpreted as minimum and maximum lengths when the "type" attribute of the AD element is "String". (They have the obvious numeric interpretation for numeric types like "Integer", "Float", etc.).

6 Javadoc

6.1 Package org.osgi.service.metatype

The OSGi Metatype Package.

See:

Description

Interface Summary	
AttributeDefinition	An interface to describe an attribute.
MetaTypeInformation	A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle.
MetaTypeProvider	Provides access to metatypes.
MetaTypeService	The MetaType Service can be used to obtain meta type information for a bundle.
ObjectClassDefinition	Description for the data type information of an objectclass.

6.2 Package org.osgi.service.metatype Description

The OSGi Metatype Package. Specification Version 1.1.

Bundles wishing to use this package must list the packagein the `Import-Package` header of the bundle's manifest.For example:

```
Import-Package: org.osgi.service.metatype; specification-version=1.1
```

All Page Within This Box

6.3 org.osgi.service.metatype
Interface MetaTypeService

public interface **MetaTypeService**

The MetaType Service can be used to obtain meta type information for a bundle. The MetaType Service will examine the specified bundle for meta type documents and to create the returned MetaTypeInformation object.

If the bundle does not contain any meta type documents, then the MetaType Service will query whether the bundle has registered any `ManagedService` or `ManagedServiceFactory` services which implement `MetaTypeProvider`. If so, then the MetaType Service will return a `MetaTypeInformation` object which wrappers these `MetaTypeProvider` objects. Thus the MetaType Service can be used to retrieve meta type information for bundles which contain a meta type resource or which provide their own `MetaTypeProvider` objects.

Method Summary

`MetaTypeInformation`	**`getMetaTypeInformation`**`(org.osgi.framework.Bundle bundle)` Return the MetaType information for the specified bundle.

Method Detail

6.3.1 getMetaTypeInformation

`public MetaTypeInformation` **`getMetaTypeInformation`**`(org.osgi.framework.Bundle bundle)`

Return the MetaType information for the specified bundle.

Parameters:

`bundle` - The bundle for which meta type information is requested.

Returns:

MetaTypeInformation object for the specified bundle.

6.4 org.osgi.service.metatype
Interface MetaTypeInformation

All Superinterfaces:

MetaTypeProvider

public interface **MetaTypeInformation**
extends MetaTypeProvider

A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle.

All Page Within This Box

Method Summary	
org.osgi.framework.Bundle	**getBundle**() Return the bundle for which this object provides metatype information.
java.lang.String[]	**getFactoryPids**() Return the Factory PIDs (for ManagedServices) for which ObjectClassDefinition information is available.
java.lang.String[]	**getPids**() Return the PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Methods inherited from interface org.osgi.service.metatype.MetaTypeProvider
getLocales, getObjectClassDefinition

Method Detail

6.4.1 getPids

public java.lang.String[] **getPids**()

Return the PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Returns:

Array of PIDs.

6.4.2 getFactoryPids

public java.lang.String[] **getFactoryPids**()

Return the Factory PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Returns:

Array of Factory PIDs.

6.4.3 getBundle

public org.osgi.framework.Bundle **getBundle**()

Return the bundle for which this object provides metatype information.

Returns:

Bundle for which this object provides metatype information.

6.5 org.osgi.service.metatype

Interface MetaTypeProvider

All Known Subinterfaces:

MetaTypeInformation

All Page Within This Box

public interface **MetaTypeProvider**

Provides access to metatypes.

Method Summary

`java.lang.String[]`	**`getLocales`**`()` Return a list of available locales.
`ObjectClassDefinition`	**`getObjectClassDefinition`**`(java.lang.String id, java.lang.String locale)` Returns an object class definition for the specified id localized to the specified locale.

Method Detail

6.5.1 getObjectClassDefinition

`public ObjectClassDefinition` **`getObjectClassDefinition`**`(java.lang.String id, java.lang.String locale)`

Returns an object class definition for the specified id localized to the specified locale.

The locale parameter must be a name that consists of `language[ "_" country[ "_" variation] ]` as is customary in the `Locale` class. This `Locale` class is not used because certain profiles do not contain it.

Parameters:
`id` - The ID of the requested object class. This can be a pid or factory pid returned by getPids or getFactoryPids.
`locale` - The locale of the definition or `null` for default locale.
Returns:
A `ObjectClassDefinition` object.
Throws:
`java.lang.IllegalArgumentException` - If the id or locale arguments are not valid

6.5.2 getLocales

`public java.lang.String[]` **`getLocales`**`()`

Return a list of available locales. The results must be names that consists of language [_ country [_ variation]] as is customary in the `Locale` class.

Returns:
An array of locale strings or `null` if there is no locale specific localization can be found.

6.6 org.osgi.service.metatype
Interface ObjectClassDefinition

All Page Within This Box

public interface **ObjectClassDefinition**

Description for the data type information of an objectclass.

Field Summary

`static int`	**`ALL`** Argument for `getAttributeDefinitions(int)`.
`static int`	**`OPTIONAL`** Argument for `getAttributeDefinitions(int)`.
`static int`	**`REQUIRED`** Argument for `getAttributeDefinitions(int)`.

Method Summary

`AttributeDefinition[]`	**`getAttributeDefinitions`**`(int filter)` Return the attribute definitions for this object class.
`java.lang.String`	**`getDescription`**`()` Return a description of this object class.
`java.io.InputStream`	**`getIcon`**`(int size)` Return an `InputStream` object that can be used to create an icon from.
`java.lang.String`	**`getID`**`()` Return the id of this object class.
`java.lang.String`	**`getName`**`()` Return the name of this object class.

Field Detail

6.6.1 REQUIRED

`public static final int` **`REQUIRED`**

Argument for `getAttributeDefinitions(int)`.

`REQUIRED` indicates that only the required definitions are returned. The value is 1.

See Also:
Constant Field Values

6.6.2 OPTIONAL

`public static final int` **`OPTIONAL`**

Argument for `getAttributeDefinitions(int)`.

`OPTIONAL` indicates that only the optional definitions are returned. The value is 2.

See Also:

All Page Within This Box

Constant Field Values

6.6.3 ALL

`public static final int` **`ALL`**

Argument for `getAttributeDefinitions(int)`.

`ALL` indicates that all the definitions are returned. The value is -1.

See Also:
Constant Field Values

Method Detail

6.6.4 getName

`public java.lang.String` **`getName`**`()`

Return the name of this object class. The name may be localized.

Returns:
The name of this object class.

6.6.5 getID

`public java.lang.String` **`getID`**`()`

Return the id of this object class.

`ObjectDefintion` objects share a global namespace in the registry. They share this aspect with LDAP/X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify object classes. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a java class name (reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined object classes already have an OID associated. It is strongly advised to define the object classes from existing LDAP schemes which will give the OID for free. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns:
The id of this object class.

6.6.6 getDescription

`public java.lang.String` **`getDescription`**`()`

Return a description of this object class. The description may be localized.

Returns:
The description of this object class.

6.6.7 getAttributeDefinitions

`public AttributeDefinition[]` **`getAttributeDefinitions`**`(int filter)`

Return the attribute definitions for this object class.

Return a set of attributes. The filter parameter can distinguish between `ALL`,`REQUIRED` or the `OPTIONAL` attributes.

Parameters:
`filter` - `ALL`,`REQUIRED`,`OPTIONAL`
Returns:
An array of attribute definitions or `null` if no attributes are selected

---

6.6.8 getIcon

```
public java.io.InputStream getIcon(int size)
                           throws java.io.IOException
```

Return an `InputStream` object that can be used to create an icon from.

Indicate the size and return an `InputStream` object containing an icon. The returned icon maybe larger or smaller than the indicated size.

The icon may depend on the localization.

Parameters:
`size` - Requested size of an icon, e.g. a 16x16 pixels icon then size = 16
Returns:
An InputStream representing an icon or `null`
Throws:
`java.io.IOException`

6.7 org.osgi.service.metatype
Interface AttributeDefinition

---

public interface **AttributeDefinition**

An interface to describe an attribute.

An `AttributeDefinition` object defines a description of the data type of a property/attribute.

Field Summary

`static int`	**`BIGDECIMAL`** **Deprecated.** *Since 1.1*
`static int`	**`BIGINTEGER`** **Deprecated.** *Since 1.1*
`static int`	**`BOOLEAN`** The `BOOLEAN` (11) type.
`static int`	**`BYTE`** The `BYTE` (6) type.
`static int`	**`CHARACTER`** The `CHARACTER` (5) type.

All Page Within This Box

static int	**DOUBLE** The DOUBLE (7) type.
static int	**FLOAT** The FLOAT (8) type.
static int	**INTEGER** The INTEGER (3) type.
static int	**LONG** The LONG (2) type.
static int	**SHORT** The SHORT (4) type.
static int	**STRING** The STRING (1) type.

Method Summary

int	**getCardinality**() Return the cardinality of this attribute.
java.lang.String[]	**getDefaultValue**() Return a default for this attribute.
java.lang.String	**getDescription**() Return a description of this attribute.
java.lang.String	**getID**() Unique identity for this attribute.
java.lang.String	**getName**() Get the name of the attribute.
java.lang.String[]	**getOptionLabels**() Return a list of labels of option values.
java.lang.String[]	**getOptionValues**() Return a list of option values that this attribute can take.
int	**getType**() Return the type for this attribute.
java.lang.String	**validate**(java.lang.String value) Validate an attribute in String form.

Field Detail

6.7.1 STRING

public static final int **STRING**

The STRING (1) type.

All Page Within This Box

Attributes of this type should be stored as `String`, `Vector` with `String` or `String[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.2 LONG

`public static final int` **`LONG`**

The `LONG` (2) type. Attributes of this type should be stored as `Long`, `Vector` with `Long` or `long[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.3 INTEGER

`public static final int` **`INTEGER`**

The `INTEGER` (3) type. Attributes of this type should be stored as `Integer`, `Vector` with `Integer` or `int[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.4 SHORT

`public static final int` **`SHORT`**

The `SHORT` (4) type. Attributes of this type should be stored as `Short`, `Vector` with `Short` or `short[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.5 CHARACTER

`public static final int` **`CHARACTER`**

The `CHARACTER` (5) type. Attributes of this type should be stored as `Character`, `Vector` with `Character` or `char[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.6 BYTE

`public static final int` **`BYTE`**

The `BYTE` (6) type. Attributes of this type should be stored as `Byte`, `Vector` with `Byte` or `byte[]` objects, depending on the `getCardinality()` value.

See Also:
Constant Field Values

6.7.7 DOUBLE

All Page Within This Box

`public static final int` **`DOUBLE`**

The `DOUBLE` (7) type. Attributes of this type should be stored as `Double`, `Vector` with `Double` or `double[]` objects, depending on the `getCardinality()` value.

See Also:

Constant Field Values

6.7.8 FLOAT

`public static final int` **`FLOAT`**

The `FLOAT` (8) type. Attributes of this type should be stored as `Float`, `Vector` with `Float` or `float[]` objects, depending on the `getCardinality()` value.

See Also:

Constant Field Values

6.7.9 BIGINTEGER

`public static final int` **`BIGINTEGER`**

Deprecated. *Since 1.1*

The `BIGINTEGER` (9) type. Attributes of this type should be stored as `BigInteger`, `Vector` with `BigInteger` or `BigInteger[]` objects, depending on the `getCardinality()` value.

See Also:

Constant Field Values

6.7.10 BIGDECIMAL

`public static final int` **`BIGDECIMAL`**

Deprecated. *Since 1.1*

The `BIGDECIMAL` (10) type. Attributes of this type should be stored as `BigDecimal`, `Vector` with `BigDecimal` or `BigDecimal[]` objects depending on `getCardinality()`.

See Also:

Constant Field Values

6.7.11 BOOLEAN

`public static final int` **`BOOLEAN`**

The `BOOLEAN` (11) type. Attributes of this type should be stored as `Boolean`, `Vector` with `Boolean` or `boolean[]` objects depending on `getCardinality()`.

See Also:

Constant Field Values

Method Detail

6.7.12 getName

`public java.lang.String` **`getName`**`()`

Get the name of the attribute. This name may be localized.

Returns:

The localized name of the definition.

All Page Within This Box

6.7.13 getID

`public java.lang.String` **`getID`**`()`

Unique identity for this attribute. Attributes share a global namespace in the registry. E.g. an attribute `cn` or `commonName` must always be a `String` and the semantics are always a name of some object. They share this aspect with LDAP/X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify an attribute. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a Java class name (reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined attributes already have an OID. It is strongly advised to define the attributes from existing LDAP schemes which will give the OID. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns:

The id or oid

6.7.14 getDescription

`public java.lang.String` **`getDescription`**`()`

Return a description of this attribute. The description may be localized and must describe the semantics of this type and any constraints.

Returns:

The localized description of the definition.

6.7.15 getCardinality

`public int` **`getCardinality`**`()`

Return the cardinality of this attribute. The OSGi environment handles multi valued attributes in arrays ([]) or in `Vector` objects. The return value is defined as follows:

```
x = Integer.MIN_VALUE    no limit, but use Vector
x < 0                    -x = max occurrences, store in Vector
x > 0                     x = max occurrences, store in array []
x = Integer.MAX_VALUE    no limit, but use array []
x = 0                     1 occurrence required
```

6.7.16 getType

`public int` **`getType`**`()`

Return the type for this attribute.

Defined in the following constants which map to the appropriate Java type.
`STRING`,`LONG`,`INTEGER`, `CHAR`,`BYTE`,`DOUBLE`,`FLOAT`, `BOOLEAN`.

6.7.17 getOptionValues

`public java.lang.String[]` **`getOptionValues`**`()`

Return a list of option values that this attribute can take.

If the function returns `null`, there are no option values available.

All Page Within This Box

Each value must be acceptable to validate() (return "") and must be a `String` object that can be converted to the data type defined by getType() for this attribute.

This list must be in the same sequence as `getOptionLabels()`. I.e. for each index i in `getOptionValues`, i in `getOptionLabels()` should be the label.

For example, if an attribute can have the value male, female, unknown, this list can return `new String[] { "male", "female", "unknown" }`.

Returns:
A list values

6.7.18 getOptionLabels

`public java.lang.String[] getOptionLabels()`

Return a list of labels of option values.

The purpose of this method is to allow menus with localized labels. It is associated with `getOptionValues`. The labels returned here are ordered in the same way as the values in that method.

If the function returns `null`, there are no option labels available.

This list must be in the same sequence as the `getOptionValues()` method. I.e. for each index i in `getOptionLabels`, i in `getOptionValues()` should be the associated value.

For example, if an attribute can have the value male, female, unknown, this list can return (for dutch) `new String[] { "Man", "Vrouw", "Onbekend" }`.

Returns:
A list values

6.7.19 validate

`public java.lang.String validate(java.lang.String value)`

Validate an attribute in `String` form. An attribute might be further constrained in value. This method will attempt to validate the attribute according to these constraints. It can return three different values:

```
null                 no validation present
""                  no problems detected
"..."               A localized description of why the value is wrong
```

Parameters:
`value` - The value before turning it into the basic data type
Returns:
`null`, "", or another string

6.7.20 getDefaultValue

`public java.lang.String[] getDefaultValue()`

All Page Within This Box

Return a default for this attribute. The object must be of the appropriate type as defined by the cardinality and `getType()`. The return type is a list of `String` objects that can be converted to the appropriate type. The cardinality of the return array must follow the absolute cardinality of this type. E.g. if the cardinality = 0, the array must contain 1 element. If the cardinality is 1, it must contain 0 or 1 elements. If it is -5, it must contain from 0 to max 5 elements. Note that the special case of a 0 cardinality, meaning a single value, does not allow arrays or vectors of 0 elements.

Returns:

Return a default value or `null` if no default exists.

7 Considered Alternatives

7.1 New metatype2 Package

The org.osgi.service.metatype package had been deprecated and replaced by the new org.osgi.service.metatype2 package. This was suggested due to the changes to the interfaces and the fact that these interfaces were expected to be implemented by bundles. Just changing the interfaces in the metatype package would break existing bundles which implement those interfaces.

This was removed when the RFC was scaled back to just support meta typing for configuration. A separate RFC will be submitted to address additional meta typing needs such as Control Unit (RFC 82) and Diagnostics (RFC 77).

8 Security Considerations

The MetaTypeService can be used by any bundle to read the meta type information from any other bundle.

9 Document Support

9.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. RFC 74 Manifest Localization, http://membercvs.osgi.org/rfcs/rfc0074/rfc-0074-Manifest%20Localization.doc

[4]. RFC 62 XML Schema, http://membercvs.osgi.org/cgi-bin/cvsweb.cgi/xmlns/metatype/metatype.xsd?cvsroot=/cvshome/build

[5]. RFC 94 MEG Deployment Configuration, http://membercvs.osgi.org/rfcs/rfc0094/rfc-0094-MEG%20Deployment%20Configuration.doc

9.2 Author's Address

Name	BJ Hargrave
Company	IBM
Address	11501 Burnet Rd, Austin, TX 78758 USA
Voice	+1 512 838 9938
e-mail	hargrave@us.ibm.com

9.3 Acronyms and Abbreviations

MS – Managed Service

MSF – Managed Service Factory

PID – Persistent IDentifier

9.4 End of Document

All Page Within This Box