Service Registry 3.1 Developer's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–4638–10
February 2007

070226@16599

Contents

Preface

The *Service Registry 3.1 Developer's Guide* describes how to use the Java™ API for XML Registries (JAXR) to query Service Registry ("the Registry") and to publish content to it.

Service Registry is an ebXML Registry: a federated registry and repository that manages all types of electronic content described by standard and extensible metadata. It provides federated, secure information management of Service Oriented Architecture (SOA) and other content and metadata. It supports the ebXML Registry 3.0 and UDDI 3.0 registry protocols.

Who Should Use This Book

The *Developer's Guide* is intended for applications programmers who plan to develop JAXR clients that search the Registry and that publish content to the Registry.

This guide assumes you are familiar with the following:

- The Java programming language
- The basic concepts of the ebXML Registry and Repository specifications

Before You Read This Book

You should be familiar with the basic concepts of these specifications:

- *ebXML Registry Information Model Version 3.0*
- *ebXML Registry Services and Protocols Version 3.0*

As you develop code, you can use the Web Console provided with the Service Registry software to verify that your code is working correctly. Read the *Service Registry 3.1 User's Guide* to familiarize yourself with the Web Console.

Service Registry is a component of Sun Java Enterprise System ("Java ES"), a software infrastructure that supports enterprise applications distributed across a network or Internet environment. You should be familiar with the Java ES* documentation at `http://docs.sun.com/coll/1286.2`.

How This Book Is Organized

The contents of this book are as follows:

Chapter 1 provides a brief overview of JAXR.

Chapter 2 describes the first steps to follow to implement a JAXR client that can perform queries and updates to the Service Registry.

Chapter 3 describes the interfaces and methods JAXR provides for querying a registry.

Chapter 4 describes how to publish objects to the Registry.

Chapter 5 describes how to perform operations on objects in the registry, such as deleting objects and changing their state.

Chapter 6 describes how to develop Java client programs that enable you to use UDDI queries to search the Registry.

Chapter 7 describes solutions to some problems that you can encounter when using JAXR with Service Registry.

Appendix A lists constants that you can use to search for objects by their unique identifiers.

Service Registry Documentation Set

The Service Registry documentation set is available at `http://docs.sun.com/app/docs/coll/1314.2`. To learn about Service Registry, refer to the books listed in the following table.

TABLE P–1 Service Registry Documentation

Document Title	Contents
Service Registry 3.1 Release Notes	Contains the latest information about Service Registry, including known problems.
Service Registry 3.1 Administration Guide	Describes how to configure Service Registry after installation and how to use the administration tool provided with the Registry. It also describes how to perform other administrative tasks.
Service Registry 3.1 User's Guide	Describes how to use the Service Registry Web Console to search Service Registry and to publish data to it.

TABLE P–1 Service Registry Documentation *(Continued)*

Document Title	Contents
Service Registry 3.1 Developer's Guide	Describes how to use the Java API for XML Registries (JAXR) to search Service Registry and to publish data to it.

Related Books

When you install Service Registry, it is deployed to the Sun Java System Application Server. For information about administering Application Server, refer to *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

The Java ES documentation set describes deployment planning and system installation. The URL for system documentation is `http://docs.sun.com/coll/1286.2`. For an introduction to Java ES, refer to the books in the order in which they are listed in the following table.

TABLE P–2 Java Enterprise System Documentation

Document Title	Contents
Sun Java Enterprise System 5 Release Notes for UNIX *Sun Java Enterprise System 5 Release Notes for Microsoft Windows*	Contains the latest information about Java ES, including known problems. In addition, components have their own release notes listed in the Release Notes Collection (`http://docs.sun.com/coll/1315.2`).
Sun Java Enterprise System 5 Technical Overview	Introduces the technical and conceptual foundations of Java ES. Describes components, the architecture, processes, and features.
Sun Java Enterprise System Deployment Planning Guide	Provides an introduction to planning and designing enterprise deployment solutions based on Java ES. Presents basic concepts and principles of deployment planning and design, discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Java ES.
Sun Java Enterprise System 5 Installation Planning Guide	Helps you develop the implementation specifications for the hardware, operating system, and network aspects of your Java ES deployment. Describes issues such as component dependencies to address in your installation and configuration plan.
Sun Java Enterprise System 5 Installation Guide for UNIX *Sun Java Enterprise System 5 Installation Guide for Microsoft Windows*	Guides you through the process of installing Java ES. Also shows how to configure components after installation, and verify that they function properly.

TABLE P–2 Java Enterprise System Documentation *(Continued)*

Document Title	Contents
Sun Java Enterprise System 5 Installation Reference for UNIX	Gives additional information about configuration parameters, provides worksheets to use in your configuration planning, and lists reference material such as default directories and port numbers on the Solaris Operating System and Linux operating environment.
Sun Java Enterprise System 5 Upgrade Guide for UNIX	Provides instructions for upgrading to Java ES 5 from previously installed versions.
Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows	
Sun Java Enterprise System 5 Monitoring Guide	Gives instructions for setting up the Monitoring Framework for each product component and using the Monitoring Console to view real-time data and create monitoring rules.
Sun Java Enterprise System Glossary	Defines terms that are used in Java ES documentation.

The URL for all documentation about Java ES and its components is `http://docs.sun.com/prod/entsys.5`.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P–3 Default Paths and File Names

Placeholder	Description	Default Value
ServiceRegistry-base	Represents the base installation directory for Service Registry.	Solaris OS: `/opt/SUNWsrvc-registry` Linux and HP-UX systems: `/opt/sun/srvc-registry`
RegistryDomain-base	Represents the directory where the Application Server domain for Service Registry is located and where the Service Registry database is located.	Solaris OS: `/var/opt/SUNWsrvc-registry` Linux and HP-UX systems: `/var/opt/sun/srvc-registry`
Ant-base	Represents the directory where the Java ES version of the Ant tool is located.	Solaris OS: `/usr/sfw/bin` Linux and HP-UX systems: `/opt/sun/share/bin`

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–4 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–5 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–6 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com℠ web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for “broker,” type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-4638.

◆ ◆ ◆ CHAPTER 1

Overview of JAXR

This section provides a brief overview of the Java™ API for XML Registries (JAXR). The section covers the following topics:

About Registries and Repositories

An XML *registry* is an infrastructure that enables the building, deployment, and discovery of web services. It is a neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions. A registry is available to organizations as a shared resource, normally in the form of a web-based service.

Currently, several specifications for XML registries exist. These specifications include

- The ebXML Registry and Repository standard, which is sponsored by the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport (U.N./CEFACT). *ebXML* stands for Electronic Business using eXtensible Markup Language.
- The Universal Description, Discovery, and Integration (UDDI) protocol, which is developed by a vendor consortium.

A *registry provider* is an implementation of a registry that conforms to a specification for XML registries.

While a UDDI registry stores information about businesses and the services they offer, an ebXML registry has a much wider scope. It is a *repository* as well as a registry. A repository stores arbitrary content as well as information about that content. In other words, a repository

stores data as well as metadata. The ebXML Registry standard defines an interoperable Enterprise Content Management (ECM) API for web services.

An ebXML registry and repository is to the web what a relational database is to enterprise applications: it provides a means for web services and web applications to store and share content and metadata.

An ebXML registry can be part of a registry *federation*, an affiliated group of registries. For example, the health ministry of a country in Europe could operate a registry, and that registry could be part of a federation that included the registries of other European health ministries.

Service Registry implements version 3.0 of the ebXML Registry and Repository specification. The specification is in two parts:

- *ebXML Registry Services and Protocols Version 3.0* ("ebXML RS") defines the services and protocols for an ebXML Registry.
- *ebXML Registry Information Model Version 3.0* ("ebXML RIM") defines the types of metadata and content that can be stored in an ebXML Registry.

Service Registry is based on an open source registry project developed at SourceForge.net. The web site for the SourceForge project contains a Wiki with additional information about ebXML registries, including an overview.

- `http://ebxmlrr.sourceforge.net/wiki/`
- `http://ebxmlrr.sourceforge.net/wiki/index.php/Overview`

About JAXR

JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. A unified JAXR information model describes content and metadata within XML registries.

JAXR gives developers the ability to write registry client programs that are portable across various target registries. JAXR also enables value-added capabilities beyond those of the underlying registries.

The current version of the JAXR specification includes detailed bindings between the JAXR information model and the ebXML Registry specifications. You can find the latest version of the JAXR specification at `http://java.sun.com/xml/downloads/jaxr.html`. The API documentation for JAXR is part of the API documentation for Java 2 Platform, Enterprise Edition (J2EE platform) (`http://java.sun.com/j2ee/1.4/docs/api/index.html`).

Service Registry includes a JAXR provider that implements the level 1 capability profile, which allows full access to ebXML registries. The ebXML specifications and the JAXR specification are not in perfect alignment, because the ebXML specifications have advanced beyond the JAXR specification. For this reason, the JAXR provider for the Registry includes some additional

implementation-specific interfaces, classes, and methods that implement the ebXML specifications. These additional interfaces, classes, and methods are likely to be included in the next version of the JAXR specification.

JAXR Architecture

The high-level architecture of JAXR consists of the following parts:

- A *JAXR client*: This is a client program that uses the JAXR API to access a registry through a JAXR provider.
- A *JAXR provider*: This is an implementation of the JAXR API that provides access to a specific registry provider or to a class of registry providers that are based on a common specification. This guide does not describe how to implement a JAXR provider.

A JAXR provider implements two main packages:

- `javax.xml.registry`, which consists of the API interfaces and classes that define the registry access interface.
- `javax.xml.registry.infomodel`, which consists of interfaces that define the information model for JAXR. These interfaces define the types of objects that reside in a registry and how they relate to each other. The basic interface in this package is the `RegistryObject` interface.

The most basic interfaces in the `javax.xml.registry` package are

- `Connection`. The `Connection` interface represents a client session with a registry provider. The client must create a connection with the JAXR provider in order to use a registry.
- `RegistryService`. The client obtains a `RegistryService` object from its connection. The `RegistryService` object in turn enables the client to obtain the interfaces it uses to access the registry.

The primary interfaces, also part of the `javax.xml.registry` package, are

- `QueryManager` and `BusinessQueryManager`, which allow the client to search a registry for information in accordance with the `javax.xml.registry.infomodel` interfaces. An optional interface, `DeclarativeQueryManager`, allows the client to use SQL syntax for queries. The ebXML provider for the Registry implements `DeclarativeQueryManager`.
- `LifeCycleManager` and `BusinessLifeCycleManager`, which allow the client to modify the information in a registry by either saving the information (updating it) or deleting it.

For more details, and for a figure that illustrates the relationships among these interfaces, see the API documentation for the `javax.xml.registry` package at `http://java.sun.com/j2ee/1.4/docs/api/javax/xml/registry/package-summary.html`.

When an error occurs, JAXR API methods throw a `JAXRException` or one of its subclasses.

Many methods in the JAXR API use a `Collection` object as an argument or a returned value. Use of a `Collection` object allows operations on several registry objects at a time.

Figure 1–1 illustrates the architecture of JAXR. For the Registry, a JAXR client uses the capability level 0 and level 1 interfaces of the JAXR API to access the JAXR provider, which is an ebXML provider. The JAXR provider in turn accesses the Registry, an ebXML registry.

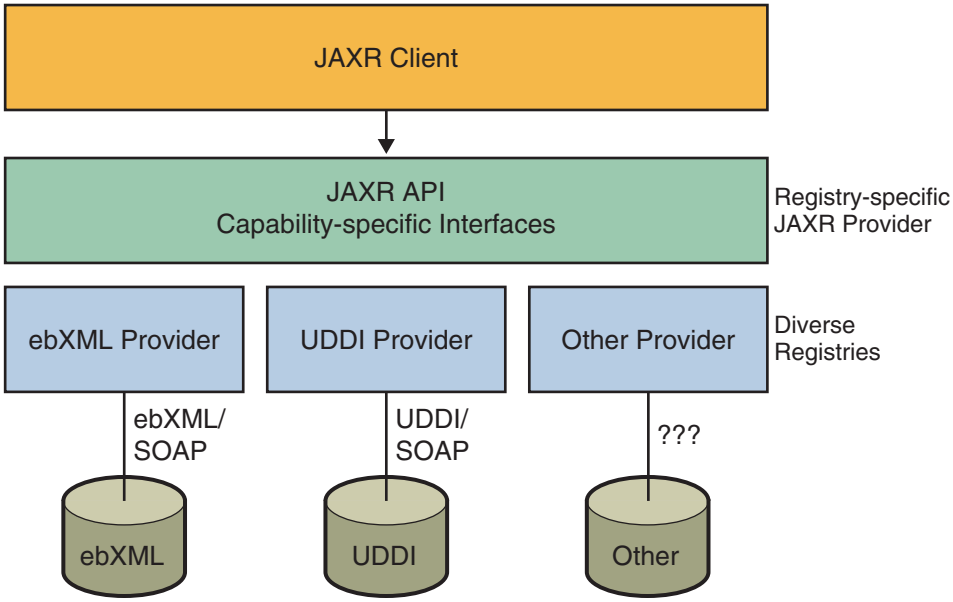

FIGURE 1–1 JAXR Architecture

About the Examples

Many sample client programs that demonstrate JAXR features are described in this manual. If you installed the developer bundle for the registry, a zip file containing these examples is in the file *ServiceRegistry-base*`/samples.zip`.

Copy this zip file to any convenient location on your file system. After you unzip the file, the example source code is in the directory *INSTALL*`/registry-samples`, where *INSTALL* is the directory where you unzipped the examples.

Each example or group of examples has a `build.xml` file that allows you to compile and run each example using the Ant tool. Each `build.xml` file has a `compile` target and one or more targets that run the example or examples. Some of the run targets take command-line arguments.

On a system where the Service Registry developer bundle is installed, the `ant` command is in the following directory:

On Solaris OS: `/usr/sfw/bin`

On Linux and HP-UX systems: `/opt/sun/share/bin`

This manual refers to this directory as *Ant-base*.

The `ant` command requires the `JAVA_HOME` environment variable to be set. Ordinarily, you set this variable to the following value:

```
/usr/jdk/entsys-j2se
```

Before you run the examples, you must edit the `build.properties` file in the directory *INSTALL*`/registry-samples/common`. This file is used by the Ant targets that run the examples.

The other properties file, `JAXRExamples.properties`, is a resource bundle that is used by the examples themselves. It contains strings that you can modify at any time. The Ant targets that run the examples always use the latest version of the file.

In addition, a `targets.xml` file in the *INSTALL*`/registry-samples/common` directory defines the classpath for compiling and running the examples. It also contains a `clean` target that deletes the `build` directory created when each example is compiled. You do not need to edit this file.

Note – You can find additional JAXR examples in the form of Junit tests at the SourceForge project web site. You can browse these examples online or follow the download instructions.

▼ To Edit the `build.properties` File

1. **Set the property `registry.home` to the directory where the Registry is installed.**

   On Solaris OS: `/opt/SUNWsrvc-registry`

   On Linux and HP-UX systems: `/opt/sun/srvc-registry`

2. **Set the property `share.dir` to the directory where the Java ES shared components are located.**

   On Solaris OS: `/usr/share`

   On Linux and HP-UX systems: `/opt/sun/share`

3. **Set the property `proxyHost` to the name of the system through which you access the Internet, if you are behind a firewall.**

   If you are not sure what the value should be, consult your system administrator or another person with that information. The `proxyPort` value is set to 8080, the typical value; change this value if necessary.

4. **Edit the properties `query.url` and `publish.url` to specify the URL of the Registry.**

   The file provides a default setting of `localhost:6480` for the host and port. Change this setting to another host or port if the Registry is installed on a remote server or at a non-default port.

5. **Edit the `alias` and `password` properties to specify the values that are required for publishing to the Registry. Make these edits after you use the User Registration Wizard of the Web Console. See "Getting Access to the Registry" on page 23 for details.**

◆ ◆ ◆ CHAPTER 2

Setting Up a JAXR Client

This chapter describes the first steps to follow to implement a JAXR client that can perform queries and updates to Service Registry. A JAXR client is a client program that uses the JAXR API to access registries.

This chapter covers the following topics:

Starting the Registry

To start the Registry, you start the container where the Registry is installed, the Sun Java System Application Server.

If the Registry is not already running, start it or ask your system administrator to do so. See "To Stop and Restart the Application Server Domain for the Registry" in *Service Registry 3.1 Administration Guide* for instructions.

Getting Access to the Registry

Any user of a JAXR client can perform queries on the Registry for objects that are not restricted by an access control policy. A user must, however, obtain permission from the Registry for the following actions:

- To add data to the Registry
- To update Registry data
- To perform queries for restricted objects

The Registry uses client-certificate authentication for user access.

To create a user that can submit data to the Registry, use the User Registration Wizard of the Web Console. The Web Console is part of the Registry software. For details on using the wizard to obtain a user name and password as well as a certificate that authorizes you to use the Registry, see "Creating a User Account" in *Service Registry 3.1 User's Guide*. You can also use an existing certificate that you obtained from a certificate authority.

Before you can publish to the Registry, you must move the certificate from the `.p12` file that you downloaded to a JKS keystore file. The keystore file must reside at the following location in your home directory: `$HOME/soar/3.0/jaxr-ebxml/security/keystore.jks`. The example programs include an Ant target that performs this task. For details, see "To Create a Keystore for Your Certificate" on page 24.

After you create a user account and a keystore, edit the `build.properties` file. See "To Edit the Security Settings of the `build.properties` File" on page 25 for details.

▼ To Create a Keystore for Your Certificate

To create a JKS keystore for your certificate, you use the Ant target `move-keystore`, which is defined in the file *INSTALL*`/registry-samples/common/targets.xml`. This targets file is used by all the `build.xml` files in the example directories.

Note – The Admin Tool `keystoreMover` command performs the same function as this Ant target. See "keystoreMover" in *Service Registry 3.1 Administration Guide* for details.

The `move-keystore` target uses a property named `keystoreFile` that is defined in the file *INSTALL*`/registry-samples/common/build.properties`. Do not change the definition of this property. The `move-keystore` target also specifies a keystore password of `ebxmlrr`. This value is used in the `storepass` property of the file `build.properties`.

1 **Go to any of the example directories except `common`.**

For example, you might use the following command:

```
cd registry-samples/search-id
```

2 **Run the following command (all on one line):**

Ant-base`/ant move-keystore -Dp12path=`*path-of-p12-file* `-Dalias=`*your-user-name* `-Dpassword=`*your-password*

Use a command like the following:

```
Ant-base/ant move-keystore -Dp12path=/home/myname/testuser.p12 -Dalias=testuser
-Dpassword=testuser
```

To see a syntax reminder for this target, use the command *Ant-base*`/ant -projecthelp`.

▼ To Edit the Security Settings of the `build.properties` File

1. **Open the file** *INSTALL*`/registry-samples/common/build.properties` **in a text editor.**
2. **Find the following lines:**

   ```
   alias=
   keypass=
   ```

3. **For the value of the** `alias` **property, specify the alias that you provided to the User Registration Wizard.**
4. **For the value of the** `keypass` **property, specify the password that you provided to the User Registration Wizard.**
5. **Save and close the file.**

Establishing a Connection to the Registry

The first task that a JAXR client must complete is to establish a connection to a registry. Establishment of a connection involves the following tasks:

- "Obtaining a Connection Factory" on page 25
- "Creating a Connection" on page 25

Obtaining a Connection Factory

A client creates a connection from a connection factory. To obtain an instance of the abstract class `ConnectionFactory`, the client calls the `getConnectionFactory` method in the JAXR provider's `JAXRUtility` class.

```
import org.freebxml.omar.client.xml.registry.util.JAXRUtility;
...
ConnectionFactory factory = JAXRUtility.getConnectionFactory();
```

Creating a Connection

To create a connection, a client first creates a set of properties that specify the URL or URLs of the registry or registries to be accessed. The following code provides the URLs of the query service and publishing service for the Registry if the Registry is deployed on the local system.

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
    "http://localhost:6480/soar/registry/soap");
props.setProperty("javax.xml.registry.lifeCycleManagerURL",
    "http://localhost:6480/soar/registry/soap");
```

The client then obtains the connection factory as described in "Obtaining a Connection Factory" on page 25, sets its properties, and creates the connection. The following code fragment performs these tasks:

```
ConnectionFactory factory = JAXRUtility.getConnectionFactory();
factory.setProperties(props);
Connection connection = factory.createConnection();
```

The `makeConnection` method in the sample programs shows the steps used to create a JAXR connection.

Table 2–1 lists and describes the two properties that you can set on a connection. These properties are defined in the JAXR specification.

TABLE 2–1 Standard JAXR Connection Properties

Property Name and Description	Data Type	Default Value
`javax.xml.registry.queryManagerURL` Specifies the URL of the query manager service within the target registry provider.	String	None
`javax.xml.registry.lifeCycleManagerURL` Specifies the URL of the life-cycle manager service within the target registry provider (for registry updates).	String	Same as the specified `queryManagerURL` value

Obtaining and Using a `RegistryService` Object

After creating the connection, the client uses the connection to obtain a `RegistryService` object and then the interface or interfaces that the client will use:

```
RegistryService rs = connection.getRegistryService();
DeclarativeQueryManager bqm = rs.getDeclarativeQueryManager();
BusinessLifeCycleManager blcm = rs.getBusinessLifeCycleManager();
```

Typically, a client obtains two objects from the `RegistryService` object: a query manager and a life cycle manager. The query manager is either a `DeclarativeQueryManager` object or a `BusinessQueryManager` object; both of these implement the base interface `QueryManager`. The life cycle manager is a `BusinessLifeCycleManager` object, which implements the base interface `LifeCycleManager`. If the client is using the Registry for simple queries only, it might need to obtain only a query manager.

If your program uses implementation-specific features of the Service Registry JAXR provider, you need to use the implementation-specific version of the respective query manager or life cycle manager: `BusinessQueryManagerImpl`, `DeclarativeQueryManagerImpl`, or `BusinessLifeCycleManagerImpl`.

◆ ◆ ◆ CHAPTER 3

Querying a Registry

This chapter describes the interfaces and methods that JAXR provides for querying a registry. The chapter covers the following topics:

Basic Query Methods

The simplest way for a client to use a registry is to query the registry for information about the objects and data it contains. The `QueryManager`, `BusinessQueryManager`, and `RegistryObject` interfaces support a number of finder and getter methods. These methods allow clients to search for data by using the JAXR information model. Many of the finder methods return a `BulkResponse`. A `BulkResponse` is a collection of objects that meets a set of criteria that are specified in the method arguments.

The most general of these methods are as follows:

- `getRegistryObject` and `getRegistryObjects`. When used with an argument, these `QueryManager` methods return one or more objects based on their object type or unique identifier. Without an argument, the `getRegistryObjects` method returns the objects owned by the caller. For information on unique identifiers, see "Finding Objects by Unique Identifier" on page 34.
- `findObjects`, an implementation-specific `BusinessQueryManager` method that returns a list of all objects of a specified type that meet the specified criteria.

Other finder methods allow you to find specific kinds of objects supported by the JAXR information model. A UDDI registry supports a specific hierarchy of objects: organizations, which contain users, services, and service bindings. In contrast, an ebXML registry permits the storage of freestanding objects of various types that can be linked to each other in various ways. Other objects are not freestanding but are always attributes of another object.

The `BusinessQueryManager` finder methods are useful primarily for searching UDDI registries. The more general `findObjects` method and the `RegistryObject` getter methods are more appropriate for Service Registry.

To execute queries for unrestricted objects (see "Getting Access to the Registry" on page 23), you do not need to log in to the Registry. By default, an unauthenticated user has the identity of the user named "Registry Guest."

JAXR Information Model Interfaces

Table 3–1 lists the main interfaces supported by the JAXR information model. All these interfaces extend the `RegistryObject` interface. The table indicates objects specific to the Service Registry implementation of JAXR.

For more details, and for a figure that illustrates the relationships among these interfaces, see the API documentation for the `javax.xml.registry.infomodel` package at `http://java.sun.com/j2ee/1.4/docs/api/javax/xml/registry/infomodel/package-summary.html`.

TABLE 3–1 JAXR `RegistryObject` Subinterfaces

Interface Name	Description
`AdhocQuery`	(Implementation-specific) Represents an ad hoc query expressed in a query syntax. `AdhocQuery` objects are stored in the Registry and are used for discovery of registry objects. `AdhocQuery` objects are similar in purpose to the concept of stored procedures in relational databases.

TABLE 3–1 JAXR `RegistryObject` Subinterfaces *(Continued)*

Interface Name	Description
`Association`	Defines a relationship between two objects. Getter and finder methods: `RegistryObject.getAssociations`, `BusinessQueryManager.findAssociations`, `BusinessQueryManager.findCallerAssociations`.
`AuditableEvent`	Provides a record of a change to an object. A collection of `AuditableEvent` objects constitutes an object's audit trail. Getter method: `RegistryObject.getAuditTrail`.
`Classification`	Classifies an object by using a `ClassificationScheme`. Getter method: `RegistryObject.getClassifications`.
`ClassificationScheme`	Represents a taxonomy used to classify objects. In an internal `ClassificationScheme`, all taxonomy elements are defined in the registry as `Concept` instances. In an external `ClassificationScheme`, the values are not defined in the registry as `Concept` instances but instead are referenced by their `String` representations. Finder methods: `BusinessQueryManager.findClassificationSchemes`, `BusinessQueryManager.findClassificationSchemeByName`.
`Concept`	Represents a taxonomy element and its structural relationship with other elements in an internal `ClassificationScheme`. Called a `ClassificationNode` in the ebXML specifications. Finder methods: `BusinessQueryManager.findConcepts`, `BusinessQueryManager.findConceptByPath`.
`ExternalIdentifier`	Provides additional information about an object by using `String` values within an identification scheme (an external `ClassificationScheme`). Examples of identification schemes are DUNS numbers and Social Security numbers. Getter method: `RegistryObject.getExternalIdentifiers`.
`ExternalLink`	Provides a URI for content that resides outside the registry. Getter method: `RegistryObject.getExternalLinks`.
`ExtrinsicObject`	Provides metadata that describes submitted content whose type is not intrinsically known to the registry and that therefore must be described by means of additional attributes, such as MIME type. No specific getter or finder methods.
`Federation`	(Implementation-specific) Represents an affiliated group of registries. No specific getter or finder methods.

TABLE 3–1 JAXR `RegistryObject` Subinterfaces *(Continued)*

Interface Name	Description
`Notification`	(Implementation-specific) Represents a notification from the registry regarding an event that matches a `Subscription`. No specific getter or finder methods.
`Organization`	Provides information about an organization. May have a parent, and may have one or more child organizations. Always has a User object as a primary contact, and may offer Service objects. Finder method: `BusinessQueryManager.findOrganizations`.
`Registry`	(Implementation-specific) Represents a registry. No specific getter or finder methods.
`RegistryPackage`	Represents a logical grouping of registry objects. A `RegistryPackage` may have any number of `RegistryObjects`. Getter and finder methods: `RegistryObject.getRegistryPackages`, `BusinessQueryManager.findRegistryPackages`.
`Service`	Provides information on a service. May have a set of `ServiceBinding` objects. Finder method: `BusinessQueryManager.findServices`.
`ServiceBinding`	Represents technical information on how to access a `Service`. Getter and finder methods: `Service.getServiceBindings`, `BusinessQueryManager.findServiceBindings`.
`SpecificationLink`	Provides the linkage between a `ServiceBinding` and a technical specification that describes how to use the service by using the `ServiceBinding`. Getter method: `ServiceBinding.getSpecificationLinks`.
`Subscription`	(Implementation-specific) Defines a `User`'s interest in certain types of `AuditableEvent` objects. No specific getter or finder methods.
`User`	Provide information about registered users within the registry. `User` objects are affiliated with `Organization` objects. Getter methods: `Organization.getUsers`, `Organization.getPrimaryContact`.

Table 3–2 lists the other interfaces supported by the JAXR information model. These interfaces provide attributes for the main registry objects. These interfaces do not extend the `RegistryObject` interface.

TABLE 3–2 JAXR Information Model Interfaces Used as Attributes

Interface Name	Description
`EmailAddress`	Represents an email address. A `User` can have an `EmailAddress`. Getter method: `User.getEmailAddresses`.
`InternationalString`	Represents a `String` that can be internationalized into several locales. Contains a `Collection` of `LocalizedString` objects. The name and description of a `RegistryObject` are `InternationalString` objects. Getter methods: `RegistryObject.getName`, `RegistryObject.getDescription`.
`Key`	An object that identifies a `RegistryObject`. Contains a unique identifier value that must be a unique URN, such as a DCE 128 UUID (Universal Unique IDentifier). Getter method: `RegistryObject.getKey`.
`LocalizedString`	A component of an `InternationalString` that associates a `String` with its `Locale`. Getter method: `InternationalString.getLocalizedStrings`.
`PersonName`	Represents a person's name. A `User` has a `PersonName`. Getter method: `User.getPersonName`.
`PostalAddress`	Represents a postal address. An `Organization` or `User` can have one or more `PostalAddress` objects. Getter methods: `Organization.getPostalAddress`, `OrganizationImpl.getPostalAddresses` (implementation-specific), `User.getPostalAddresses`.
`Slot`	Provides a dynamic way to add arbitrary attributes to `RegistryObject` instances. Getter methods: `RegistryObject.getSlot`, `RegistryObject.getSlots`.
`TelephoneNumber`	Represents a telephone number. An `Organization` or a `User` can have one or more `TelephoneNumber` objects. Getter methods: `Organization.getTelephoneNumbers`, `User.getTelephoneNumbers`.

Finding Objects by Unique Identifier

Every object in the Registry has two identifiers, a unique identifier (also called a `Key`) and a logical identifier. Often, the unique identifier is the same as the logical identifier. However, when an object exists in more than one version, the unique identifiers are different for each version, but the logical identifier remains the same. (See "Retrieving the Version of an Object" on page 54.)

If you know the value of the unique identifier for an object, you can retrieve the object by calling the `QueryManager.getRegistryObject` method with the `String` value as an argument. For example, if `bqm` is your `BusinessQueryManager` instance and `idString` is the `String` value, the following line of code retrieves the object:

```
RegistryObject obj = bqm.getRegistryObject(idString);
```

After you have the object, you can obtain its type, name, description, and other attributes.

Finding objects by identifier is the most efficient way to retrieve objects from the Registry.

Finding Objects by Unique Identifier: Example

For an example of finding objects by unique identifier, see `JAXRSearchById.java` in the directory *INSTALL*`/registry-samples/search-id/src`, which searches for objects that have a specified unique identifier.

▼ To Run the `JAXRSearchById` Example

1. **Go to the directory** *INSTALL*`/registry-samples/search-id`**.**

2. **Type the following command:**

   Ant-base`/ant run -Did=`*urn-value*

   For example, if you specify the following ID, you retrieve information on the `ObjectType` classification scheme.

   ```
   urn:oasis:names:tc:ebxml-regrep:classificationScheme:ObjectType
   ```

Finding Objects by Name

To search for objects by name, you normally use a combination of find qualifiers and name patterns. Find qualifiers affect sorting and pattern matching. Name patterns specify the strings to be searched. The `BusinessQueryManagerImpl.findObjects` method takes a collection of `FindQualifier` objects as its second argument and takes a collection of name patterns as its third argument. The method signature is as follows:

```
public BulkResponse findObjects(java.lang.String objectType,
        java.util.Collection findQualifiers,
        java.util.Collection namePatterns,
        java.util.Collection classifications,
        java.util.Collection specifications,
        java.util.Collection externalIdentifiers,
        java.util.Collection externalLinks)
    throws JAXRException
```

For the first argument, the object type, you normally specify one of a set of string constants that are defined in the `LifeCycleManager` interface.

You can use wildcards in a name pattern. Use percent signs (`%`) to specify that the search string occurs at the beginning, middle, or end of the object name. Here are some examples:

- Specify **`nor%`** to return strings that start with `Nor` or `nor`, such as `North` and `northern`.
- Specify **`%off%`** to return strings that contain the string `off`, such as `Coffee`.
- Specify **`%ica`** to return strings that end with `ica`, such as `America`.

You can also use an underscore (`_`) as a wildcard to match a single character. For example, the search string `_us_` would match objects named `Aus1` and `Bus3`.

For example, the following code fragment finds all the organizations in the Registry whose names begin with a specified string, `searchString`, and sorts them in alphabetical order.

```
// Define find qualifiers and name patterns
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection namePatterns = new ArrayList();
namePatterns.add(searchString + "%");

// Find organizations with name that starts with searchString
BulkResponse response =
     bqm.findObjects("Organization", findQualifiers,
         namePatterns, null, null, null, null);
Collection orgs = response.getCollection();
```

The `findObjects` method is not case-sensitive, unless you specify `FindQualifier.CASE_SENSITIVE_MATCH`. In the previous fragment, the first argument could be either `"Organization"` or `"organization"`, and the name pattern matches names regardless of case.

The following code fragment performs a case-sensitive search for all registry objects whose names contain the string `searchString` and sorts the objects in alphabetical order.

```
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.CASE_SENSITIVE_MATCH);
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection namePatterns = new ArrayList();
namePatterns.add("%" + searchString + "%");

// Find objects with name that contains searchString
BulkResponse response =
     bqm.findObjects("RegistryObject", findQualifiers,
         namePatterns, null, null, null, null);
Collection objects = response.getCollection();
```

To locate a particular object, search by unique identifier if possible. Searching by name is less efficient and more likely to lead to errors, since names are not unique.

Finding Objects by Name: Example

For an example of finding objects by name, see `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

▼ To Run the `JAXRSearchByName` Example

1. **Go to the directory** *INSTALL*`/registry-samples/search-name`**.**
2. **Type the following command, specifying a string value:**

   Ant-base`/ant run -Dname=`*string*

   The program performs a case-insensitive search, returning all objects whose names contain the specified string. The program also displays the object's classifications, external identifiers, external links, slots, and audit trail.

Finding Objects by Type

To find all objects of a specified type, specify only the first argument of the `BusinessQueryManagerImpl.findObjects` method and, optionally, a collection of `FindQualifier` objects. For example, if `typeString` is a string whose value is `LifeCycleManager.SERVICE`, the following code fragment finds all services in the Registry and sorts them in alphabetical order.

```
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.SORT_BY_NAME_ASC);

BulkResponse response = bqm.findObjects(typeString,
     findQualifiers, null, null, null, null, null);
```

You cannot use wildcards in the first argument to `findObjects`.

Finding Objects by Type: Example

For an example of finding objects by type, see `JAXRSearchByObjectType.java` in the directory *INSTALL*`/registry-samples/search-object-type/src`.

▼ To Run the `JAXRSearchByObjectType` Example

1. **Go to the directory** *INSTALL*`/registry-samples/search-object-type`**.**
2. **Type the following command, specifying a string value:**

   Ant-base`/ant run -Dtype=`*type-name*

   Specify the exact name of the type, not a wildcard, as in the following command line:

   Ant-base`/ant run -Dtype=federation`

   The `JAXRSearchByObjectType` program passes the object type as a string argument to `QueryManager.findObjects` in order to accept user input as simply as possible. However, developers should use the constants defined by the `LifeCycleManager` interface.

   The program performs a case-insensitive search, returning all objects whose type is *type-name*and displaying their names, descriptions, and unique identifiers. At the end, it displays the number of objects found.

Finding Objects by Classification

To find objects by classification, you first establish the classification within a particular classification scheme. Then you specify the classification as an argument to the `BusinessQueryManagerImpl.findObjects` method.

To establish the classification within a particular classification scheme, you first find the classification scheme. Then you create a `Classification` object to be used as an argument to the `findObjects` method or another finder method.

The following code fragment finds all organizations that correspond to a particular classification within the ISO 3166 country codes classification system that is maintained by the International Organization for Standardization (ISO). See `http://www.iso.org/iso/en/prods-services/iso3166ma/index.html` for details. This classification scheme is provided in the sample database that is included with the Registry.

```
String schemeId = "urn:freebxml:registry:demo:schemes:iso-ch:3166:1999";
ClassificationScheme cScheme =
     (ClassificationScheme) bqm.getRegistryObject(schemeId);

Classification classification =
     blcm.createClassification(cScheme, "United States", "US");
Collection classifications = new ArrayList();
classifications.add(classification);
// perform search
BulkResponse response = bqm.findObjects(LifeCycleManager.ORGANIZATION,
    null, null, classifications, null, null, null);
Collection orgs = response.getCollection();
```

The ebXML Registry Information Model Specification requires a set of canonical classification schemes to be present in an ebXML registry. Each scheme also has a set of required concepts (which are called `ClassificationNode` objects in the ebXML specifications). The primary purpose of the canonical classification schemes is not to classify objects but to provide enumerated types for object attributes. For example, the `EmailType` classification scheme provides a set of values for the `type` attribute of an `EmailAddress` object.

Table 3–3 lists and describes these canonical classification schemes.

TABLE 3–3 Canonical Classification Schemes

Classification Scheme	Description
`AssociationType`	Defines the types of associations between `RegistryObjects`.
`ContentManagementService`	Defines the types of content management services.

TABLE 3–3 Canonical Classification Schemes *(Continued)*

Classification Scheme	Description
`DataType`	Defines the data types for attributes in classes defined by the specification.
`DeletionScopeType`	Defines the values for the `deletionScope` attribute in the `RemoveObjectsRequest` protocol message.
`EmailType`	Defines the types of email addresses.
`ErrorHandlingModel`	Defines the types of error handling models for content management services.
`ErrorSeverityType`	Defines the different error severity types encountered by the registry during processing of protocol messages.
`EventType`	Defines the types of events that can occur in a registry.
`InvocationModel`	Defines the different ways that a content management service may be invoked by the registry.
`NodeType`	Defines the different ways in which a `ClassificationScheme` may assign the value of the code attribute for its `ClassificationNodes`.
`NotificationOptionType`	Defines the different ways in which a client may be notified by the registry of an event within a `Subscription`.
`ObjectType`	Defines the different types of `RegistryObjects` a registry may support.
`PhoneType`	Defines the types of telephone numbers.
`QueryLanguage`	Defines the query languages supported by a registry.
`ResponseStatusType`	Defines the different types of status for a `RegistryResponse`.
`StatusType`	Defines the different types of status for a `RegistryObject`.
`SubjectGroup`	Defines the groups that a `User` may belong to for access control purposes.
`SubjectRole`	Defines the roles that may be assigned to a `User` for access control purposes.

To find objects that use the canonical classification schemes and their concepts, you can look up the objects by using string constants that are defined in the package `org.freebxml.common.CanonicalConstants`. The constants are listed in "Constants for Classification Schemes" on page 94.

First, you look up the classification scheme by using the value of its unique identifier:

```
String schemeId =
    CanonicalConstants.CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectRole;
ClassificationScheme cScheme =
    (ClassificationScheme) bqm.getRegistryObject(schemeId);
String schemeName = getName(cScheme);
```

Then you look up the concept in the same way and create a classification from it:

```
String concId =
    CanonicalConstants.CANONICAL_SUBJECT_ROLE_ID_RegistryAdministrator;
Concept concept = (Concept) bqm.getRegistryObject(concId);
Classification classification =
    blcm.createClassification(concept);
```

Finally, you search for objects in the same way you do with a non-canonical classification scheme:

```
Collection classifications = new ArrayList();
classifications.add(classification);
BulkResponse response = bqm.findObjects("RegistryObject",
    null, null, classifications, null, null, null);
Collection objects = response.getCollection();
```

For a sample program that displays all the canonical classification schemes and their concepts, see `JAXRGetCanonicalSchemes.java` in the directory *INSTALL*`/registry-samples/classification-schemes/src`.

▼ To Run the `JAXRGetCanonicalSchemes` Example

1. **Go to the directory** *INSTALL*`/registry-samples/classification-schemes`**.**
2. **Type the following command:**

   Ant-base`/ant get-schemes`

Finding Objects by Classification: Examples

For examples of finding objects by classification, see `JAXRSearchByClassification.java` and `JAXRSearchByCountryClassification.java` in the directory *INSTALL*`/registry-samples/search-classification/src`. The first example searches for objects that use the canonical classification scheme `SubjectRole`, while the other example searches for organizations that use a geographical classification.

The program `JAXRSearchByCountryClassification.java` uses the `OR_ALL_KEYS` field of `FindQualifier` to find organizations that use either of two geographical classifications. By default, finder methods look for objects that have all the specified classifications.

▼ To Run the `JAXRSearchByClassification` and `JAXRSearchByCountryClassification` Examples

Before You Begin To obtain results from the `JAXRSearchByCountryClassification` example, you must publish an object that uses the specified classifications. Run the example in "Adding Classifications: Example" on page 69 first.

1. **Go to the directory** *INSTALL*`/registry-samples/search-classification`**.**
2. **Type either of the following commands:**

   Ant-base`/ant search-class`
   Ant-base`/ant search-geo`

   The `search-class` target typically returns one result. The `search-geo` target returns results if you have run the `run` target in "Adding Classifications: Example" on page 69.

Finding Objects by External Identifier

Finding objects by external identifier is similar to finding objects by classification. You first find the classification scheme, then create an `ExternalIdentifier` object to be used as an argument to the `BusinessQueryManagerImpl.findObjects` method or another finder method.

The following code fragment finds all registry objects that contain the Sun Microsystems stock ticker symbol as an external identifier. You need to create an external classification scheme named `NASDAQ` for this example to work. See "Adding External Identifiers to Objects" on page 69 for details on how to perform this task.

The collection of external identifiers is supplied as the next-to-last argument of the `findObjects` method.

```
String schemeId = "urn:devguide:samples:ClassificationScheme:NASDAQ";
ClassificationScheme cScheme = (ClassificationScheme)
    bqm.getRegistryObject(schemeId);

ExternalIdentifier extId =
    blcm.createExternalIdentifier(cScheme, "%Sun%",
        "SUNW");
Collection extIds = new ArrayList();
extIds.add(extId);
// perform search
BulkResponse response = bqm.findObjects("RegistryObject",
    null, null, null, null, extIds, null);
Collection objects = response.getCollection();
```

Finding Objects by External Identifier: Example

For an example of finding objects by external identifier, see `JAXRSearchByExternalIdentifier.java` in the directory *INSTALL*`/registry-samples/search-external-identifier/src`, which searches for objects that use the `NASDAQ` classification scheme.

▼ To Run the `JAXRSearchByExternalIdentifier` Example

Before You Begin To obtain results from this example, first run the `publish-object` example described in "Adding Classifications: Example" on page 69.

1. **Go to the directory** *INSTALL*`/registry-samples/search-external-identifier`**.**
2. **Type the following command:**

   Ant-base`/ant run`

Finding Objects by External Link

Finding objects by external link does not require the use of a classification scheme, but it does require you to specify a valid URI. The arguments to the `createExternalLink` method are a URI and a description.

If the link you specify is outside your firewall, you must also specify the system properties `http.proxyHost` and `http.proxyPort` when you run the program so that JAXR can determine the validity of the URI.

The following code fragment finds all organizations that have a specified `ExternalLink` object.

```
ExternalLink extLink =
     blcm.createExternalLink("http://java.sun.com/",
         "Sun Java site");

Collection extLinks = new ArrayList();
extLinks.add(extLink);
BulkResponse response = bqm.findObjects(LifeCycleManager.ORGANIZATION,
     null, null, null, null, null, extLinks);
Collection objects = response.getCollection();
```

Finding Objects by External Link: Example

For an example of finding objects by external link, see `JAXRSearchByExternalLink.java` in the directory *INSTALL*`/registry-samples/search-external-link/src`, which searches for

objects that have a specified external link. The `http.proxyHost` and `http.proxyPort` properties are specified in the `run` target in the `build.xml` file. Make sure you have set these properties as described in "To Edit the `build.properties` File" on page 21.

▼ To Run the `JAXRSearchByExternalLink` Example

Before You Begin To obtain results from this example, first run the `publish-object` example described in "Adding Classifications: Example" on page 69.

1. **Go to the directory** *INSTALL*`/registry-samples/search-external-link`**.**
2. **Type the following command:**

   Ant-base`/ant run`

Finding Objects You Published

You can retrieve all objects that you published to the Registry. Alternatively, you can narrow the search to retrieve only the objects that you published that are of a particular object type. To retrieve all the objects that you have published, use the no-argument version of the `QueryManager.getRegistryObjects` method. The name of this method is misleading, because the method returns only objects that you have published, not all registry objects.

For example, if `bqm` is your `BusinessQueryManager` instance, use the following line of code:

```
BulkResponse response = bqm.getRegistryObjects();
```

To retrieve all the objects of a particular type that you published, use `QueryManager.getRegistryObjects` with a constant argument that specifies the type:

```
BulkResponse response = bqm.getRegistryObjects(LifeCycleManager.SERVICE);
```

The `QueryManager.getRegistryObjects` method is case-sensitive.

The sample programs `JAXRGetMyObjects` and `JAXRGetMyObjectsByType` show how to use these methods.

Finding Objects You Published: Examples

For examples of finding objects you published, see `JAXRGetMyObjects.java` and `JAXRGetMyObjectsByType.java` in the directory *INSTALL*`/registry-samples/get-objects/src`. The first example, `JAXRGetMyObjects.java`, retrieves all objects you have published. The second example, `JAXRGetMyObjectsByType.java`, retrieves all the objects you have published of a specified type.

▼ To Run the `JAXRGetMyObjects` and `JAXRGetMyObjectsByType` Examples

1. **Go to the directory** *INSTALL*`/registry-samples/get-objects`**.**
2. **To find all the objects that you have published, type the following command:**

   Ant-base`/ant get-obj`
3. **To find all the objects that you have published of a specified type, type the following command, where** *type-name* **is case-sensitive:**

   Ant-base`/ant get-obj-type -Dtype=`*type-name*

   The `JAXRGetMyObjectsByType` program passes the object type as a string argument to `QueryManager.getRegistryObjects` in order to accept user input as simply as possible. However, developers should use the constants defined by the `LifeCycleManager` interface.

Retrieving Information About an Object

After you have retrieved the object or objects you are searching for, you can also retrieve the object's attributes and other objects that belong to it:

- Name
- Description
- Type
- Unique identifier and logical identifier
- Classifications
- External identifiers
- External links
- Slots

For an organization, you can also retrieve the following:

- The primary contact, which is a `User` object
- Postal address
- Telephone numbers
- Services

For a service, you can retrieve the service bindings.

For any object, you can also retrieve the audit trail, which contains the events that have changed the object's state, and the version. You can also retrieve an object's version number. If versioning is turned on, the version number is updated whenever a change is made to one of the object's attributes.

Note – At this release of Service Registry, versioning is turned off by default. To enable versioning, an administrator must perform the task described in "Enabling Versioning of Registry Objects" in *Service Registry 3.1 Administration Guide*.)

This section covers the following topics:

Retrieving the Identifier Values for an Object

The unique identifier for an object is contained in a `Key` object. A `Key` is a structure that contains the identifier in the form of an id attribute that is a String value. To retrieve the identifier, call the method `RegistryObject.getKey().getId()`.

The JAXR provider also has an implementation-specific method for retrieving the logical identifier, which is called a `lid`. The `lid` is a `String` attribute of a `RegistryObject`. To retrieve the `lid`, call `RegistryObjectImpl.getLid`. The method has the following signature:

```
public java.lang.String getLid()
    throws JAXRException
```

For an example of the use of this method, see `JAXRSearchOrg.java` in the directory *INSTALL*`/registry-samples/organizations/src`. For more information on this example, see "Retrieving Organization Attributes: Example" on page 50.

Retrieving the Name or Description of an Object

The name and description of an object are both `InternationalString` objects. An `InternationalString` object contains a set of `LocalizedString` objects. The methods `RegistryObject.getName` and `RegistryObject.getDescription` return the `LocalizedString` object for the default locale. You can then retrieve the `String` value of the `LocalizedString` object. The following code fragment uses these methods:

```
String name = ro.getName().getValue();
String description = ro.getDescription().getValue();
```

Call the `getName` or `getDescription` method with a `Locale` argument to retrieve the value for a particular locale.

Many of the examples contain private utility methods that retrieve the name, description, and unique identifier for an object. See, for example, `JAXRGetMyObjects.java` in the directory *INSTALL*`/registry-samples/get-objects/src`.

Retrieving the Type of an Object

If you have searched the Registry without specifying a particular object type, you can retrieve the type of the objects returned by the search. Use the `RegistryObject.getObjectType` method, which returns a `Concept` value. You can then use the `Concept.getValue` method to obtain the `String` value of the object type. The following code fragment uses these methods:

```
Concept objType = object.getObjectType();
System.out.println("Object type is " + objType.getValue());
```

The concept will be one of those in the canonical classification scheme `ObjectType`. For an example of this code, see `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Retrieving the Classifications for an Object

Use the `RegistryObject.getClassifications` method to retrieve a `Collection` of the object's classifications. For a classification, the important attributes are its value and the classification scheme to which it belongs. Often, a classification has no name or description. The following code fragment retrieves and displays an object's classifications.

```
Collection classifications = object.getClassifications();
Iterator classIter = classifications.iterator();
while (classIter.hasNext()) {
    Classification classification =
         (Classification) classIter.next();
    String name = classification.getName().getValue();
    System.out.println("  Classification name is " + name);
    System.out.println("  Classification value is " +
        classification.getValue());
    ClassificationScheme scheme =
         classification.getClassificationScheme();
    System.out.println("  Classification scheme for " +
        name + " is " + scheme.getName().getValue());
}
```

Some of the examples have a `showClassifications` method that uses code similar to this. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Retrieving the External Identifiers for an Object

Use the `RegistryObject.getExternalIdentifiers` method to retrieve a `Collection` of the object's external identifiers. For each identifier, you can retrieve its name, its value, and the classification scheme to which it belongs. For an external identifier, the method that retrieves the classification scheme is `getIdentificationScheme`. The following code fragment retrieves and displays an object's external identifiers.

```
Collection exIds = object.getExternalIdentifiers();
Iterator exIdIter = exIds.iterator();
while (exIdIter.hasNext()) {
    ExternalIdentifier exId =
         (ExternalIdentifier) exIdIter.next();
    String name = exId.getName().getValue();
    System.out.println("  External identifier name is " +
        name);
    String exIdValue = exId.getValue();
    System.out.println("  External identifier value is " +
         exIdValue);
    ClassificationScheme scheme =
         exId.getIdentificationScheme();
    System.out.println("  External identifier " +
         "classification scheme is " +
         scheme.getName().getValue());
}
```

Some of the examples have a `showExternalIdentifiers` method that uses code similar to this. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Retrieving the External Links for an Object

Use the `RegistryObject.getExternalLinks` method to retrieve a `Collection` of the object's external links. For each external link, you can retrieve its name, description, and value. For an external link, the name is optional. The following code fragment retrieves and displays an object's external links.

```
Collection exLinks = obj.getExternalLinks();
Iterator exLinkIter = exLinks.iterator();
while (exLinkIter.hasNext()) {
```

```
    ExternalLink exLink = (ExternalLink) exLinkIter.next();
    String name = exLink.getName().getValue();
    if (name != null) {
        System.out.println("  External link name is " + name);
    }
    String description = exLink.getDescription().getValue();
    System.out.println("  External link description is " +
        description);
    String externalURI = exLink.getExternalURI();
    System.out.println("  External link URI is " +
         externalURI);
}
```

Some of the examples have a `showExternalLinks` method that uses code similar to this. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Retrieving the Slots for an Object

Slots are arbitrary attributes that you can create for an object. Use the `RegistryObject.getSlots` method to retrieve a `Collection` of the object's slots. For each slot, you can retrieve its name, values, and type. The name of a `Slot` object is a `String`, not an `InternationalString`, and a slot has a `Collection` of values. The following fragment retrieves and displays an object's slots:

```
Collection slots = object.getSlots();
Iterator slotIter = slots.iterator();
while (slotIter.hasNext()) {
    Slot slot = (Slot) slotIter.next();
    String name = slot.getName();
    System.out.println("  Slot name is " + name);
    Collection values = slot.getValues();
    Iterator valIter = values.iterator();
    int count = 1;
    while (valIter.hasNext()) {
        String value = (String) valIter.next();
        System.out.println("  Slot value " + count++ +
             ": " + value);
    }
    String type = slot.getSlotType();
    if (type != null) {
        System.out.println("  Slot type is " + type);
}
```

Some of the examples have a `showSlots` method that uses this code. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Retrieving the Attributes of an Organization or User

Every `Organization` object can have one or more postal addresses and one or more telephone numbers in addition to the attributes that are available to all other objects. Every organization also has a `User` object as a primary contact. The organization can have additional affiliated `User` objects.

The attributes for a `User` object include a `PersonName` object, which has a different format from the name of an object. A user can have multiple postal addresses as well as multiple telephone numbers. A user can also have multiple email addresses.

To retrieve the postal address for an organization, call the `Organization.getPostalAddress` method as follows (`org` is the organization):

```
PostalAddress pAd = org.getPostalAddress();
```

After you retrieve the address, you can retrieve the address attributes as follows:

```
System.out.println(" Postal Address:\n  " +
    pAd.getStreetNumber() + " " + pAd.getStreet() +
    "\n  " + pAd.getCity() + ", " +
    pAd.getStateOrProvince() + " " +
    pAd.getPostalCode() + "\n  " + pAd.getCountry() +
    "(" + pAd.getType() + ")");
```

To retrieve the primary contact for an organization, call the `Organization.getPrimaryContact` method as follows (`org` is the organization):

```
User pc = org.getPrimaryContact();
```

To retrieve the postal addresses for a user, call the `User.getPostalAddresses` method and extract the `Collection` values as follows (`pc` is the primary contact):

```
Collection pcpAddrs = pc.getPostalAddresses();
Iterator pcaddIter = pcpAddrs.iterator();
while (pcaddIter.hasNext()) {
    PostalAddress pAd = (PostalAddress) pcaddIter.next();
    /* retrieve attributes */
}
```

To retrieve the telephone numbers for either an organization or a user, call the `getTelephoneNumbers` method. In the following code fragment, `org` is the organization. The code retrieves the country code, area code, main number, and type of the telephone number.

```
Collection orgphNums = org.getTelephoneNumbers(null);
Iterator orgphIter = orgphNums.iterator();
while (orgphIter.hasNext()) {
```

```
    TelephoneNumber num = (TelephoneNumber) orgphIter.next();
    System.out.println(" Phone number: " +
         "+" + num.getCountryCode() + " " +
        "(" + num.getAreaCode() + ") " +
        num.getNumber() + " (" + num.getType() + ")");
}
```

A `TelephoneNumber` can also have an extension, retrievable through the `getExtension` method. If the number can be dialed electronically, it can have a `url` attribute, retrievable through the `getUrl` method.

To retrieve the name of a user, call the `User.getPersonName` method. A `PersonName` has three attributes that correspond to the given name, middle name (or names), and surname of a user. In the following code fragment, `pc` is the primary contact.

```
PersonName pcName = pc.getPersonName();
System.out.println(" Contact name: " +
    pcName.getFirstName() + " " +
    pcName.getMiddleName() + " " +
    pcName.getLastName());
```

To retrieve the email addresses for a user, call the `User.getEmailAddresses` method. An `EmailAddress` has two attributes, the address and its type. In the following code fragment, `pc` is the primary contact.

```
Collection eAddrs = pc.getEmailAddresses();
Iterator eaIter = eAddrs.iterator();
while (eaIter.hasNext()) {
    EmailAddress eAd = (EmailAddress) eaIter.next();
    System.out.println("  Email address: " +
        eAd.getAddress() + " (" + eAd.getType() + ")");
}
```

The attributes for `PostalAddress`, `TelephoneNumber`, `PersonName`, and `EmailAddress` objects are all String values. As noted in "JAXR Information Model Interfaces" on page 30, these objects do not extend the `RegistryObject` interface, so they do not have the attributes of other registry objects.

Retrieving Organization Attributes: Example

For an example of retrieving the attributes of an organization and the `User` that is its primary contact, see `JAXRSearchOrg.java` in the directory *INSTALL*`/registry-samples/organizations/src`, which displays information about an organization whose name contains a specified string.

▼ To Run the `JAXRSearchOrg` Example

1. **Go to the directory** *INSTALL*`/registry-samples/organizations`**.**
2. **Type the following command:**

   Ant-base`/ant search-org -Dorg=`*string*

Retrieving the Services and Service Bindings for an Organization

Most organizations offer services. JAXR has methods that retrieve the services and service bindings for an organization.

A `Service` object has all the attributes of other registry objects. In addition, it normally has *service bindings*, which provide information about how to access the service. A `ServiceBinding` object, along with its other attributes, normally has an access URI. It can also have a specification link, which provides the linkage between a service binding and a technical specification that describes how to use the service through the service binding.

A specification link has the following attributes:

- A specification object, which is typically an `ExtrinsicObject`
- A usage description, which is an `InternationalString` object
- A `Collection` of usage parameters, which are `String` values

You can use the `Service.getProvidingOrganization` method to retrieve the organization that provides a service, and you can use the `ServiceBinding.getService` method to retrieve the service for a service binding.

The following code fragment retrieves the services for the organization `org`. Then it retrieves the service bindings for each service and, for each service binding, its access URI.

```
Collection services = org.getServices();
Iterator svcIter = services.iterator();
while (svcIter.hasNext()) {
    Service svc = (Service) svcIter.next();
    System.out.println(" Service name: " + getName(svc));
    System.out.println(" Service description: " +
        getDescription(svc));

    Collection serviceBindings = svc.getServiceBindings();
    Iterator sbIter = serviceBindings.iterator();
    while (sbIter.hasNext()) {
        ServiceBinding sb = (ServiceBinding) sbIter.next();
        System.out.println("  Binding name: " +
```

```
            getName(sb));
        System.out.println("  Binding description: " +
            getDescription(sb));
        System.out.println("  Access URI: " +
            sb.getAccessURI());
        }
    }
}
```

The example "Retrieving Organization Attributes: Example" on page 50 also displays the services and service bindings for the organizations it finds.

Services often exist independent of an organization. You can search for services directly using the `BusinessQueryManagerImpl.findObjects` method.

Retrieving an Organization Hierarchy

JAXR allows you to group organizations into families. One organization can have other organizations as its children. The child organizations can also have children. Therefore, any given organization can have a parent, children, and descendants.

The `Organization.getParentOrganization` method retrieves an organization's parent. In the following fragment, `chorg` is a child organization.

```
Organization porg = chorg.getParentOrganization();
```

The `Organization.getChildOrganizations` method retrieves a `Collection` of the organization's children. In the following fragment, `org` is a parent organization.

```
Collection children = org.getChildOrganizations();
```

The `Organization.getDescendantOrganizations` method retrieves multiple generations of descendants, while the `Organization.getRootOrganization` method retrieves the parentless ancestor of any descendant.

For an example of retrieving an organization hierarchy, see "Creating and Retrieving an Organization Hierarchy: Examples" on page 78.

Retrieving the Audit Trail of an Object

Whenever an object is published to the Registry, and whenever it is modified in any way, the JAXR provider creates another object, called an `AuditableEvent`. The JAXR provider also creates an `AuditableEvent` object when some objects are modified in certain ways. The JAXR provider adds the `AuditableEvent` object to the audit trail for the published object. The audit trail contains a list of all the events for that object. To retrieve the audit trail, call

`RegistryObject.getAuditTrail`. You can also retrieve the individual events in the audit trail and find out their event types. JAXR supports the event types listed in Table 3–4.

TABLE 3–4 `AuditableEvent` Types

Event Type	Description
`EVENT_TYPE_CREATED`	Object was created and was published to the registry.
`EVENT_TYPE_DELETED`	Object was deleted using one of the `LifeCycleManager` or `BusinessLifeCycleManager` deletion methods.
`EVENT_TYPE_DEPRECATED`	Object was deprecated using the `LifeCycleManager.deprecateObjects` method.
`EVENT_TYPE_UNDEPRECATED`	Object was undeprecated using the `LifeCycleManager.unDeprecateObjects` method.
`EVENT_TYPE_VERSIONED`	A new version of the object was created. If versioning is enabled, this event typically happens when any of the object's attributes changes.
`EVENT_TYPE_UPDATED`	Object was updated.
`EVENT_TYPE_APPROVED`	Object was approved using the `LifeCycleManagerImpl.approveObjects` method (implementation-specific).
`EVENT_TYPE_DOWNLOADED`	Object was downloaded (implementation-specific).
`EVENT_TYPE_RELOCATED`	Object was relocated from another registry (implementation-specific).

The following code fragment retrieves the audit trail for a registry object, displaying the type and timestamp of each event:

```
Collection events = obj.getAuditTrail();
String objName = obj.getName().getValue();
Iterator eventIter = events.iterator();
while (eventIter.hasNext()) {
    AuditableEventImpl ae = (AuditableEventImpl) eventIter.next();
    int eType = ae.getEventType();
    if (eType == AuditableEvent.EVENT_TYPE_CREATED) {
        System.out.print(objName + " created ");
    } else if (eType == AuditableEvent.EVENT_TYPE_DELETED) {
        System.out.print(objName + " deleted ");
    } else if (eType == AuditableEvent.EVENT_TYPE_DEPRECATED) {
        System.out.print(objName + " deprecated ");
    } else if (eType == AuditableEvent.EVENT_TYPE_UNDEPRECATED) {
        System.out.print(objName + " undeprecated ");
    } else if (eType == AuditableEvent.EVENT_TYPE_UPDATED) {
```

```
            System.out.print(objName + " updated ");
        } else if (eType == AuditableEvent.EVENT_TYPE_VERSIONED) {
            System.out.print(objName + " versioned ");
        } else if (eType == AuditableEventImpl.EVENT_TYPE_APPROVED) {
            System.out.print(objName + " approved ");
        } else if (eType == AuditableEventImpl.EVENT_TYPE_DOWNLOADED) {
            System.out.print(objName + " downloaded ");
        } else if (eType == AuditableEventImpl.EVENT_TYPE_RELOCATED) {
            System.out.print(objName + " relocated ");
        } else {
            System.out.print("Unknown event for " + objName + " ");
        }System.out.println(ae.getTimestamp().toString());
    }
```

Some of the examples have a `showAuditTrail` method that uses code similar to this. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

See "Changing the State of Objects in the Registry" on page 85 for information on how to change the state of registry objects.

Retrieving the Version of an Object

If you modify the attributes of a registry object, the Registry may create a new version of the object. For details on how versioning happens, see "Changing the State of Objects in the Registry" on page 85. When you first create an object, the object has a version of 1.1.

Note – At this release, versioning of objects is disabled by default. To enable versioning of objects, an administrator must perform the task described in "Enabling Versioning of Registry Objects" in *Service Registry 3.1 Administration Guide*. The administrator commonly enables versioning for some object types but not for all.

To retrieve the version of an object, use the implementation-specific `getVersionInfo` method for a registry object, which returns a `VersionInfoType` object. The method has the following signature:

```
public VersionInfoType getVersionInfo()
    throws JAXRException
```

For example, to retrieve the version number for the organization `org`, cast `org` to a `RegistryObjectImpl` when you call the method. Then call the `VersionInfoType.getVersionName` method, which returns a `String`.

```
import org.oasis.ebxml.registry.bindings.rim.VersionInfoType;
...
VersionInfoType vInfo =
     ((RegistryObjectImpl)org).getVersionInfo();
if (vInfo != null) {
    System.out.println("Org version: " +
        vInfo.getVersionName());
}
```

Some of the examples use code similar to this. See, for example, `JAXRSearchByName.java` in the directory *INSTALL*`/registry-samples/search-name/src`.

Using Declarative Queries

Instead of the `BusinessQueryManager` interface, you can use the `DeclarativeQueryManager` interface to create and execute queries to the Registry. If you are familiar with SQL, you might prefer to use declarative queries. The `DeclarativeQueryManager` interface depends on another interface, `Query`.

The `DeclarativeQueryManager` interface has two methods, `createQuery` and `executeQuery`. The `createQuery` method takes two arguments, a query type and a string that contains the query. The following code fragment creates an SQL query that asks for a list of all `Service` objects in the Registry. Here, `rs` is a `RegistryService` object.

```
DeclarativeQueryManager qm = rs.getDeclarativeQueryManager();
String qString = "select s.* from Service s";
Query query = qm.createQuery(Query.QUERY_TYPE_SQL, qString);
```

After you create the query, you execute it as follows:

```
BulkResponse response = qm.executeQuery(query);
Collection objects = response.getCollection();
```

You then extract the objects from the response just as you do with ordinary queries.

For more information on SQL query syntax and for examples, see Chapter 6, "Query Management Protocols," of the ebRS 3.0 specification, especially Section 6.6.

Using Declarative Queries: Example

For examples of the use of declarative queries, see `JAXRQueryDeclarative.java` and `JAXRGetAllSchemes.java` in the directory *INSTALL*`/registry-samples/query-declarative/src`. Both examples create and execute a SQL query. The query strings are defined in the `JAXRExamples.properties` file.

The SQL query string for `JAXRQueryDeclarative` is as follows (all on one line):

```
SELECT ro.* from RegistryObject ro, Name nm, Description d
WHERE upper(nm.value) LIKE upper('%free%') AND upper(d.value)
LIKE upper('%free%') AND (ro.id = nm.parent AND ro.id = d.parent)
```

This query finds all objects that have the string `"free"` in both the name and the description attributes.

The SQL query string for `JAXRGetAllSchemes` is as follows:

```
SELECT * FROM ClassScheme s order by s.id
```

This query finds all the classification schemes in the Registry.

▼ To Run the `JAXRQueryDeclarative` Example

1. **Go to the directory** *INSTALL*`/registry-samples/query-declarative`**.**
2. **To run the** `JAXRQueryDeclarative` **example, type the following command:**

   Ant-base`/ant get-free`
3. **To run the** `JAXRGetAllSchemes` **example, type the following command:**

   Ant-base`/ant get-schemes`

Using Iterative Queries

If you expect a declarative query to return a very large result set, you can use the implementation-specific iterative query feature. The `DeclarativeQueryManagerImpl.executeQuery` method can take an argument that specifies a set of parameters. This method has the following signature:

```
public BulkResponse executeQuery(Query query,
        java.util.Map queryParams,
        IterativeQueryParams iterativeParams)
    throws JAXRException
```

You can specify parameters that cause each query to request a different subset of results within the result set. Instead of making one query return the entire result set, you can make each individual query return a manageable set of results.

Suppose you have a query string that you expect to return up to 100 results. You can create a set of parameters that causes the query to return 10 results at a time. First, you create an instance of the class `IterativeQueryParams`, which is defined in the package `org.freebxml.omar.common`.

The two fields of the class are `startIndex`, the starting index of the array, and `maxResults`, the maximum number of results to return. You specify the initial values for these fields in the constructor.

```
int maxResults = 10;
int startIndex = 0;
IterativeQueryParams iterativeQueryParams =
     new IterativeQueryParams(startIndex, maxResults);
```

Execute the queries within a `for` loop that terminates with the highest number of expected results and that advances by the `maxResults` value for the individual queries. Increment the `startIndex` field at each loop iteration.

```
for (int i = 0; i < 100; i += maxResults) {
    // Execute query with iterative query params
    Query query = dqm.createQuery(Query.QUERY_TYPE_SQL,
        queryStr);
     iterativeQueryParams.startIndex = i;
    BulkResponse br = dqm.executeQuery(query, null,
        iterativeQueryParams);
    Collection objects = br.getCollection();
    // retrieve individual objects ...
}
```

The Registry is not required to maintain transactional consistency or state between iterations of a query. New objects might be added to the complete result set between iterations, or existing objects might be removed from the result set. Therefore, you might notice that a result set element is skipped or duplicated between iterations.

Using Iterative Queries: Example

For an example of the use of an iterative query, see `JAXRQueryIterative.java` in the directory *INSTALL*`/registry-samples/query-iterative/src`. This program finds all registry objects whose names match a given string and then iterates through the first 100 of them.

▼ To Run the `JAXRQueryIterative` Example

1. **Go to the directory** *INSTALL*`/registry-samples/query-iterative`**.**
2. **Type the following command, specifying a string value:**

   Ant-base`/ant run -Dname=`*string*

Using Stored Queries

It is possible to invoke queries that are stored in the Registry. A number of predefined queries are already stored in the Registry as `AdhocQuery` objects. To find them, search for objects of that type. Most of these queries are provided in the Web Console. See Chapter 2, "Searching the Registry," in *Service Registry 3.1 User's Guide* for information on how to invoke these queries through the Web Console.

To invoke stored queries through JAXR, you need to know the following:

- The unique identifier for the query. Constants you can use to specify this identifier are in "Constants for Stored Queries" on page 100.
- The parameters to specify to the query. Only the simplest queries (`GetCallersUser` and `FindAllMyObjects`) take no parameters. The easiest way to determine the parameters used by a query is to use the Web Console to find the object and then to examine its details, which include the SQL statement executed by the query.

For example, suppose you wanted to invoke the stored query named Basic Query to search by object type for all organizations in the Registry. If you look at this query in the Web Console, you can see that it takes a parameter named `$objectTypePath`. This means that you specify the classification scheme node as a path structure instead of specifying the concept for the object type directly. The following code first retrieves the constant for the query identifier, then specifies the object type path. The actual `String` value of the object type path is `/urn:oasis:names:tc:ebxml-regrep:classificationScheme:ObjectType/RegistryObject/Organization`.

```
String queryId =
    CanonicalConstants.CANONICAL_QUERY_BasicQuery;
String objectType =
    CanonicalConstants.CANONICAL_CLASSIFICATION_SCHEME_ID_ObjectType;
String objectTypePath =
    "/" + objectType + "/RegistryObject/Organization";
```

Once you have the values for the query identifier and the parameters, you create a `HashMap` to pass these values. First put the parameters in the `HashMap`, and finally the query identifier, using the constant `CANONICAL_SLOT_QUERY_ID` to specify the parameter name. Then you call an implementation-specific form of the `DeclarativeQueryManager.createQuery` method to create the query. Finally, you call an implementation-specific form of the `DeclarativeQueryManager.executeQuery` method to execute the parameterized query.

```
Map parameters = new HashMap();
parameters.put("$objectTypePath", objectTypePath);
parameters.put(CanonicalConstants.CANONICAL_SLOT_QUERY_ID, queryId);
Query query = dqm.createQuery(Query.QUERY_TYPE_SQL);

BulkResponse br = dqm.executeQuery(query, parameters);
```

The signatures of the implementation-specific forms of the `createQuery` and `executeQuery` methods are as follows:

```
public Query createQuery(int queryType)
    throws InvalidRequestException, JAXRException

public BulkResponse executeQuery(Query query, java.util.Map queryParams)
    throws JAXRException
```

Using Stored Queries: Example

For an example of the use of a stored query, see `JAXRQueryStored.java` in the directory *INSTALL*`/registry-samples/query-stored/src`. This example returns all organizations stored in the Registry.

▼ To Run the `JAXRQueryStored` Example

1. **Go to the directory** *INSTALL*`/registry-samples/query-stored`**.**
2. **Type the following command:**

   Ant-base`/ant run`

Using Federated Queries

If the registry you are querying is part of one or more registry federations (see "About Registries and Repositories" on page 17), you can perform declarative queries on all registries in all federations of which your registry is a member, or on all the registries in one federation.

To perform a query on all registries in all federations of which your registry is a member, call the implementation-specific `setFederated` method on a `QueryImpl` object. The method has the following signature:

```
public void setFederated(boolean federated)
    throws JAXRException
```

You call the method as follows:

```
QueryImpl query = (QueryImpl)
    dqm.createQuery(Query.QUERY_TYPE_SQL, qString);
query.setFederated(true);
```

If you know that your registry is a member of only one federation, this method is the only one you need to call before you execute the query.

To limit your query to the registries in one federation, you need to call an additional implementation-specific method, `setFederation`. This method takes as its argument the unique identifier of the federation you want to query:

```
public void setFederation(java.lang.String federationId)
    throws JAXRException
```

Therefore, before you can call this method, you must obtain the unique identifier value. Normally, this value is well-known.

Next, create the query, call `setFederated` and `setFederation`, and execute the query:

```
QueryImpl query = (QueryImpl)
    dqm.createQuery(Query.QUERY_TYPE_SQL, qString);
query.setFederated(true);
query.setFederation(fedId);
response = dqm.executeQuery(query);
```

Using Federated Queries: Example

For an example of the use of a federated query, see `JAXRQueryFederation.java` in the directory *INSTALL*`/registry-samples/query-federation/src`. This example performs two queries, a declarative query and a stored query, on every federation it finds (the database provided with the Registry contains only one). Because the federation stored in the Registry database does not have a well-known identifier value, the example uses `findObjects` to locate the federation.

The declarative query is the query that is performed in "Using Declarative Queries: Example" on page 55. The stored query is the `FindAllMyObjects` query.

Because the federation in the Registry database has no members, this example cannot perform the queries.

▼ To Run the `JAXRQueryFederation` Example

1. **Go to the directory** *INSTALL*`/registry-samples/query-federation`**.**
2. **Type the following command:**

   Ant-base`/ant run`

◆ ◆ ◆ CHAPTER 4

Publishing Objects to the Registry

If a client has authorization to do so, it can submit objects to Service Registry, modify objects, and remove objects. A client uses the `BusinessLifeCycleManager` interface to perform these tasks.

Registries usually allow a client to modify or remove objects only if the objects are being modified or removed by the same user who first submitted them. Access policies can control who is authorized to publish objects and to perform actions on them.

Note – The term *submit* is used by the ebXML Registry specifications; the term *publish* is used by the JAXR API specification. The two terms mean the same thing.

Publishing registry objects involves the following tasks:

- "Authenticating with the Registry" on page 62
- "Creating Objects" on page 63
- "Saving Objects in the Registry" on page 78

Submitting objects is a multi-step task: you create the objects and populate them by setting their attributes, then you save the objects. The objects appear in the registry only after you save them.

You may remember that when you search for objects by classification, external identifier, and the like, you create the classification or other object that you are using in the search. (For an example, see "Finding Objects by Classification" on page 38.) However, you do not save this object. You create the object only for the purposes of the search, after which the object disappears. You do not need authorization from the Registry to create an object, but you must have authorization to save it.

Authenticating with the Registry

The Registry uses certificate authentication, so to submit data to the Registry you must have a certificate. You must also use the User Registration Wizard of the Web Console to create a user who can submit data to the Registry. See "Getting Access to the Registry" on page 23 for details.

Before a client can submit data, the client must send its certificate to the Registry in a set of *credentials*. The following code fragment shows how to perform this task. You need to specify the following required values to obtain credentials:

- The keystore path, the full path to the file, typically `keystore.jks`, in which the certificate key is stored
- The keystore password, typically `ebxmlrr`
- The user name and password that you chose when you registered using the Wizard

Typically, you would retrieve the four required values from property settings, and you would encapsulate much of the code in a method.

```
String keystorePath = "myKeystorePath";
String storepass = "myStorepass";
String alias = "myAlias";
String keypass = myKeypass";

Set credentials = new HashSet();
KeyStore keyStore = KeyStore.getInstance("JKS");
keyStore.load(new BufferedInputStream(
    new FileInputStream(keystorePath)),
     storepass.toCharArray());
X509Certificate cert = (X509Certificate)
     keyStore.getCertificate(alias);
PrivateKey privateKey =
     (PrivateKey) keyStore.getKey(alias, keypass.toCharArray());
credentials.add(new X500PrivateCredential(cert, privateKey,
    alias));
connection.setCredentials(credentials);
```

If the `setCredentials` method succeeds, you are logged in to the Registry and can publish objects.

The sample programs that authenticate with the Registry all call a method named `getCredentialsFromKeystore` that contains this code. The method is defined in the file *INSTALL*`/registry-samples/common/src/RegistryCredentials.java`.

Note – The method in `RegistryCredentials.java` contains commented-out lines that display the values of the keystore file, the keystore password, the user alias, and the user password. If you get an error the first time you try to publish an object, remove the comment characters from these lines to make sure that these properties are set correctly as described in "To Edit the Security Settings of the `build.properties` File" on page 25.

Creating Objects

A client creates an object and populates it with data before publishing it. You can create and publish any of the following types of `RegistryObject`:

- `AdhocQuery` (implementation-specific)
- `Association`
- `ClassificationScheme`
- `Concept`
- `ExternalLink`
- `ExtrinsicObject`
- `Federation` (implementation-specific)
- `Organization`
- `Person` (implementation-specific)
- `RegistryPackage`
- `Service`
- `Subscription`
- `User`

The following types of `RegistryObject` cannot be published separately, but you can create and save these objects as part of another object:

- `Classification` (any `RegistryObject`)
- `ExternalIdentifier` (any `RegistryObject`)
- `ServiceBinding` (`Service`)
- `Slot` (any `RegistryObject`)
- `SpecificationLink` (`ServiceBinding`)

Some objects fall into special categories:

- An `AuditableEvent` is published by the Registry when an object has a change in state.
- A `Notification` is published by the Registry when an `AuditableEvent` that matches a `Subscription` occurs.
- A `Registry` can be published only by a user with the role `RegistryAdministrator`.

The subsections that follow describe first the tasks common to creating and saving all registry objects. The subsections then describe some tasks specific to particular object types.

Using Create Methods for Objects

The `LifeCycleManager` interface supports create methods for all types of `RegistryObject` (except `AuditableEvent` and `Notification`, which can be created only by the Registry itself).

In addition, you can use the `LifeCycleManager.createObject` factory method to create an object of a particular type. This method takes a `String` argument consisting of one of the static fields supported by the `LifeCycleManager` interface. In the following code fragment, `blcm` is the `BusinessLifeCycleManager` object:

```
Organization org = (Organization)
    blcm.createObject(blcm.ORGANIZATION);
```

The object-specific create methods usually take one or more parameters that set some of the attributes of the object. For example, the `createOrganization` method sets the name of the organization:

```
Organization org = blcm.createOrganization("MyOrgName");
```

On the other hand, the `createExtrinsicObject` method normally takes a `DataHandler` argument that sets the repository item for the extrinsic object.

Adding Names and Descriptions to Objects

For all objects, you can set the name and description attributes by calling setter methods. These attributes are of type `InternationalString`. An `InternationalString` includes a set of `LocalizedString` objects that allow users to display the name and description in one or more locales. By default, the `InternationalString` value uses the default locale.

For example, the following fragment creates a description that uses two localized strings. One string is in the language of the default locale. The other string is in Canadian French.

```
InternationalString is =
    blcm.createInternationalString("What We Do");
Locale loc = new Locale("fr", "CA");
LocalizedString ls = blcm.createLocalizedString(loc,
     "ce que nous faisons");
is.addLocalizedString(ls);
org.setDescription(is);
```

Identifying Objects

As stated in "Finding Objects by Unique Identifier" on page 34, every object in the Registry has two identifiers, a unique identifier and a logical identifier. If you do not set these identifiers when you create the object, the Registry generates a unique value and assigns that value to both the unique and the logical identifiers.

Whenever a new version of an object is created, the logical identifier remains the same as the original one, but the Registry generates a new unique identifier by adding a colon and the version number to the unique identifier. See "Retrieving the Version of an Object" on page 54 and "Creating Relationships Between Objects: Associations" on page 81 for more information.

If you are creating long-lived objects, you should establish a hierarchical identification scheme with human-friendly names. Then you can use API methods to set object identifiers. You can find examples of human-friendly identifier names in many of the objects that populate the default Registry database. For example, the organization in this database has the identifier `urn:freebxml:registry:Organization:freebXMLRegistry`.

In the JAXR API, the unique identifier is called a `Key` object. You can use the `LifeCycleManager.createKey` method to create a unique identifier from a `String` object. You can then use the `RegistryObject.setKey` method to set the key.

The logical identifier is called a `lid`. The JAXR provider for the Registry has an implementation-specific method, `RegistryObjectImpl.setLid`, which also takes a `String` argument, for setting this identifier. The method has the following signature:

```
public void setLid(java.lang.String lid)
    throws JAXRException
```

Any identifier that you specify must be a valid, globally unique URN (Uniform Resource Name). When the JAXR API generates a key for an object, the key is in the form of a DCE 128 UUID (Universal Unique IDentifier).

If you set the unique identifier but do not set the `lid`, the Registry assigns the object a `lid` that has the same value as the unique identifier. If you set the `lid` explicitly when you create an object, the value should be the same as that of the unique identifier.

Most of the sample programs do not set identifiers, because the objects they create are not expected to be long-lived or unique objects.

Creating and Using Classification Schemes and Concepts

You can create your own classification schemes and concept hierarchies for classifying registry objects. To do so, follow these steps:

1. Use the `LifeCycleManager.createClassificationScheme` method to create the classification scheme.
2. Use the `LifeCycleManager.createConcept` method to create concepts.
3. Use the `ClassificationScheme.addChildConcept` method to add the concepts to the classification scheme.
4. For a deeper hierarchy, use the `Concept.addChildConcept` method to add child concepts to the concepts.
5. Save the classification scheme.

The `LifeCycleManager.createClassificationScheme` method has several forms. You can specify two arguments, a name and description, as either `String` or `InternationalString` values. For example, to create a classification scheme to describe how books are shelved in a library, you could use the following code fragment:

```
ClassificationScheme cs =
     blcm.createClassificationScheme("LibraryFloors",
         "Scheme for Shelving Books");
```

An alternate form of the `createClassificationScheme` method takes one argument, a `Concept`, and converts the concept to a `ClassificationScheme`.

The `createConcept` method takes three arguments: a parent, a name, and a value. The parent can be either a `ClassificationScheme` or another `Concept`. You can specify a value but no name.

The following code fragment creates a concept for each floor of the library by using a static `String` array that contains the names of the floors. The code fragment then adds the concept to the classification scheme.

```
for (int i = 0; i < floors.length; i++) {
    Concept con = blcm.createConcept(cs, floors[i], floors[i]);
    cs.addChildConcept(con);
    ...
```

For each concept, you can create more new concepts and call `Concept.addChildConcept` to create another level of the hierarchy. When you save the classification scheme, the entire concept hierarchy is also saved.

Creating and Displaying Classification Schemes: Examples

For an example of creating a classification scheme, see `JAXRPublishScheme.java` in the directory *INSTALL*`/registry-samples/classification-schemes/src`. This example creates a classification scheme named `LibraryFloors` and a concept hierarchy that includes each floor of the library and the subject areas that can be found there. It gives the classification scheme a human-friendly unique identifier value so that it can be searched for by identifier rather than by name.

To display the concept hierarchy, use the program `JAXRSearchScheme.java` in the same directory. This example displays the concept hierarchy for the `LibraryFloors` classification scheme.

To delete the classification scheme and concepts, use the program `JAXRDeleteScheme.java` in the same directory.

▼ To Run the `JAXRPublishScheme` Example

1. **Go to the directory** *INSTALL*`/registry-samples/classification-schemes`.
2. **Type the following command:**

   Ant-base`/ant pub-scheme`

▼ To Run the `JAXRSearchScheme` Example

1. **Go to the directory** *INSTALL*`/registry-samples/classification-schemes`.
2. **Type the following command:**

   Ant-base`/ant search-scheme`

▼ To Run the `JAXRDeleteScheme` Example

1. **Go to the directory** *INSTALL*`/registry-samples/classification-schemes`.
2. **Type the following command:**

   Ant-base`/ant del-scheme`

Adding Classifications to Objects

Objects can have one or more classifications based on one or more classification schemes (taxonomies). To establish a classification for an object, the client first locates the taxonomy. The client then creates a classification by using the classification scheme and a concept (a taxonomy element) within the classification scheme.

For information on creating a new classification scheme with a hierarchy of concepts, see "Creating and Using Classification Schemes and Concepts" on page 66. A classification scheme with a concept hierarchy is called an *internal classification scheme*.

To add a classification that uses an existing classification scheme, you usually call the `QueryManager.getRegistryObject` method to find the classification scheme, specifying the unique identifier of the scheme as the argument. For example, the following code fragment searches for the classification scheme that is named `AssociationType`, using the value defined in `CanonicalConstants`:

```
String schemeId =
    CanonicalConstants.CANONICAL_CLASSIFICATION_SCHEME_ID_AssociationType;
ClassificationScheme cScheme = (ClassificationScheme)
     bqm.getRegistryObject(schemeId);
```

After you locate the classification scheme, you call the `LifeCycleManager.createClassification` method, specifying three arguments: the classification scheme and the name and value of the concept.

```
Classification classification =
     blcm.createClassification(cScheme, "Extends", "Extends");
```

An alternative method is to call `BusinessQueryManager.findConcepts` (or `BusinessQueryManagerImpl.findObjects` with a `LifeCycleManager.CONCEPT` argument) to locate the concept you wish to use, and then to call another form of `createClassification`, with the concept as the only argument:

```
Classification classification =
     blcm.createClassification(concept);
```

After creating the classification, you call `RegistryObject.addClassification` to add the classification to the object.

```
object.addClassification(classification);
```

To add multiple classifications, you can create a `Collection`, add the classification to the `Collection`, and call `RegistryObject.addClassifications` to add the `Collection` to the object.

Adding Classifications: Example

For an example of adding classifications to an object, see `JAXRPublishObject.java` in the directory *INSTALL*`/registry-samples/publish-object/src`. This example creates an organization named `GenericOrg` and adds a number of objects to it.

▼ To Run the `JAXRPublishObject` Example

1. **Go to the directory** *INSTALL*`/registry-samples/publish-object`**.**
2. **Type the following command:**

   Ant-base`/ant run`

Adding External Identifiers to Objects

To add an external identifier to an object, follow these steps:

1. Find or create the classification scheme to be used.
2. Create an external identifier using the classification scheme.

To create external identifiers, you use an *external classification scheme*, which is a classification scheme without a concept hierarchy. You specify a name and value for the external identifier.

The database that is supplied with the Registry does not include any external classification schemes. Before you can use an external classification scheme, you must create it, using code like the following:

```
ClassificationScheme extScheme =
     blcm.createClassificationScheme("NASDAQ",
         "OTC Stock Exchange");
String extSchemeId = "urn:devguide:samples:ClassificationScheme:NASDAQ";
Key extSchemeKey = blcm.createKey(extSchemeId);
extScheme.setKey(extSchemeKey);
Collection classSchemes = new ArrayList();
classSchemes.add(extScheme);
BulkResponse response = blcm.saveObjects(classSchemes);
```

To find an existing classification scheme, you typically call the `BusinessQueryManager.getRegistryObject` method, as described in "Adding Classifications to Objects" on page 68.

For example, the following code fragment finds the external classification scheme you just created:

```
String extSchemeId = "urn:devguide:samples:ClassificationScheme:NASDAQ";
ClassificationScheme extScheme =
     bqm.getRegistryObject(extSchemeId);
```

To add the external identifier, you call the `LifeCycleManager.createExternalIdentifier` method, which takes three arguments: the classification scheme and the name and value of the external identifier. Then you add the external identifier to the object.

```
ExternalIdentifier extId =
     blcm.createExternalIdentifier(extScheme, "Sun",
         "SUNW");
object.addExternalIdentifier(extId);
```

The example *INSTALL*`/registry-samples/publish-object/src/JAXRPublishObject.java`, described in "Adding Classifications: Example" on page 69, also adds an external identifier to an object.

Adding External Links to Objects

To add an external link to an object, you call the `LifeCycleManager.createExternalLink` method, which takes two arguments: the URI of the link, and a description of the link. Then you add the external link to the object.

```
String eiURI = "http://java.sun.com/";
String eiDescription = "Java Technology";
ExternalLink extLink =
     blcm.createExternalLink(eiURI, eiDescription);
object.addExternalLink(extLink);
```

The URI must be a valid URI, and the JAXR provider checks its validity. If the link that you specify is outside your firewall, you need to specify the system properties `http.proxyHost` and `http.proxyPort` when you run the program so that JAXR can determine the validity of the URI.

To disable URI validation (for example, if you want to specify a link that is not currently active), call the `ExternalLink.setValidateURI` method before you create the link.

```
extLink.setValidateURI(false);
```

The example *INSTALL*`/registry-samples/publish-object/src/JAXRPublishObject.java`, described in "Adding Classifications: Example" on page 69, also adds an external link to an object. The `build.xml` file for this example specifies the system properties `http.proxyHost` and `http.proxyPort`.

Adding Slots to Objects

Slots are arbitrary attributes, so the API provides maximum flexibility for you to create them. You can provide a name, one or more values, and a type. The name and type are `String` objects.

The name is ordinarily a human-friendly URN. The type is the unique identifier value of a concept in the canonical `DataType` classification scheme; see "Constants for Data Type Concepts" on page 95.

The value or values are stored as a `Collection` of `String` objects, but the `LifeCycleManager.createSlot` method has a form that allows you to specify a single `String` value. For example, the following code fragment creates a slot using a `String` value, then adds the slot to the object.

```
String slotName = "urn:com:acme:organizationalUnit:Branch";
String slotValue = "Paris";
String slotType = CanonicalConstants.CANONICAL_DATA_TYPE_ID_String;
Slot slot = blcm.createSlot(slotName, slotValue, slotType);
org.addSlot(slot);
```

The example *INSTALL*`/registry-samples/publish-object/src/JAXRPublishObject.java`, described in "Adding Classifications: Example" on page 69, also adds a slot to an object.

Creating Extrinsic Objects

As "About Registries and Repositories" on page 17 explains, the Registry includes a repository in which you can store electronic content. For every item that you store in the repository, you must first create an `ExtrinsicObject`. When you save the `ExtrinsicObject` to the Registry, the associated repository item is also saved.

To create an `ExtrinsicObject`, you first need to create a `javax.activation.DataHandler` object for the repository item. The `LifeCycleManager.createExtrinsicObject` method takes a `DataHandler` argument.

Note – You can also use an implementation-specific form of the `createExtrinsicObject` method that takes no arguments. If you use this form, you can create the `DataHandler` object later and use the `ExtrinsicObject.setRepositoryItem` method to specify the repository item. You can also create extrinsic objects that have no associated repository items.

To store a file in the repository, for example, first create a `java.io.File` object. From the `File` object, create a `javax.activation.FileDataSource` object, which you use to instantiate the `DataHandler` object.

```
String filename = "./MyFile.xml";
File repositoryItemFile = new File(filename);
DataHandler repositoryItem =
     new DataHandler(new FileDataSource(repositoryItemFile));
```

Next, call `createExtrinsicObject` with the `DataHandler` as argument:

```
ExtrinsicObject eo =
     blcm.createExtrinsicObject(repositoryItem);
eo.setName("My Graphics File");
```

Set the MIME type of the object to make the object accessible. The default MIME type is `application/octet-stream`. If the file is an XML file, set the MIME type as follows:

```
eo.setMimeType("text/xml");
```

Finally, call the implementation-specific `ExtrinsicObjectImpl.setObjectType` method to store the `ExtrinsicObject` in an appropriate area of the Registry. This method has the following signature:

```
public void setObjectType(Concept objectType)
    throws JAXRException
```

The easiest way to find the appropriate concept for a particular type of file is to use the Explore feature of the Web Console. Look under the `ObjectType` classification scheme for the various types of `ExtrinsicObject` concepts. Specify the ID for the concept as the argument to `getRegistryObject`, then specify the concept as the argument to `setObjectType`.

```
String conceptId =
"urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:ExtrinsicObject:XML";
Concept objectTypeConcept =
     (Concept) bqm.getRegistryObject(conceptId);
((ExtrinsicObjectImpl)eo).setObjectType(objectTypeConcept);
```

The constant that represents this value is `CanonicalConstants.CANONICAL_OBJECT_TYPE_ID_XML`.

Finally, you save the `ExtrinsicObject` to the Registry.

```
Collection extobjs = new ArrayList();
extobjs.add(eo);
BulkResponse response = blcm.saveObjects(extobjs);
```

The `ExtrinsicObject` contains the metadata, and a copy of the file is stored in the repository.

If the Registry does not have a concept for the kind of file that you want to store there, you can create and save the concept yourself.

Creating an Extrinsic Object: Example

For an example of creating an extrinsic object, see `JAXRPublishExtrinsicObject.java` in the directory *INSTALL*`/registry-samples/publish-extrinsic/src`. This example publishes an XML file to the Registry (its own `build.xml` file).

▼ To Run the `JAXRPublishExtrinsicObject` Example

1 **Go to the directory** *INSTALL*`/registry-samples/publish-extrinsic`**.**

2 **Type the following command:**

Ant-base`/ant run`

Creating Services by Publishing WSDL Files

An ebXML Registry includes an XML content cataloging service. This means that when you publish certain kinds of files as `ExtrinsicObject` objects, the registry creates other kinds of objects. The most important use of this capability is the creation of `Service` objects from WSDL files.

If you publish a WSDL file as an `ExtrinsicObject`, the cataloging service creates one or more services and service bindings, along with additional `ExtrinsicObject` objects, based on the content of the WSDL file.

To publish a `Service` based on a WSDL file, create an `ExtrinsicObject` as described in "Creating Extrinsic Objects" on page 71. Use the ID for the WSDL concept and the `text/xml` MIME type.

```
String conceptId =
"urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:ExtrinsicObject:WSDL";
Concept objectTypeConcept =
     (Concept) bqm.getRegistryObject(conceptId);
((ExtrinsicObjectImpl)eo).setObjectType(objectTypeConcept);
eo.setMimeType("text/xml");
```

The constant that represents this value is `CanonicalConstants.CANONICAL_OBJECT_TYPE_ID_WSDL`. However, the interface for WSDL object type concepts is `org.freebxml.omar.common.profile.ws.wsdl.CanonicalConstants`, not `org.freebxml.omar.common.CanonicalConstants`.

After you publish the WSDL file, you will find in the Registry a service and service binding that correspond to the `service` and `portType` elements of the WSDL file, along with additional `ExtrinsicObject` objects of type `Binding` and `PortType`.

Note – To publish a WSDL file that contains imports or includes of other WSDL and/or XML Schema files, you must package all the files in a zip file and publish the zip file as an `ExtrinsicObject` with the object type that of the WSDL concept and the mime type `"application/zip"`.

When you remove a service from the Registry, the service bindings are also removed. However, the extrinsic objects associated with the service are not removed. Similarly, if you remove an extrinsic object and its WSDL file from the Registry and repository, the service associated with them is not removed.

Creating a Service by Publishing a WSDL File: Example

For an example of creating a service by publishing a WSDL file, see `JAXRPublishService.java` in the directory *INSTALL*`/registry-samples/publish-service/src`. This example publishes a simple WSDL file named `MyCoffeeService.wsdl`.

▼ To Run the `JAXRPublishService` Example

1. **Go to the directory** *INSTALL*`/registry-samples/publish-service`**.**
2. **Type the following command:**

   Ant-base`/ant run`

Creating Organizations

An `Organization` object is probably the most complex registry object. This object normally includes the following attributes, in addition to those common to all objects:

- One or more `PostalAddress` objects.
- One or more `TelephoneNumber` objects.
- A `PrimaryContact` object, which is a `User` object. A `User` object normally includes a `PersonName` object and collections of `TelephoneNumber`, `EmailAddress`, and `PostalAddress` objects.
- One or more `Service` objects and their associated `ServiceBinding` objects.

An organization can also have one or more child organizations, which can in turn have children, to form a hierarchy of organizations.

The following code fragment creates an organization and specifies its name, description, postal address, and telephone number.

```
// Create organization name and description
Organization org =
     blcm.createOrganization("The ebXML Coffee Break");
InternationalString is =
    blcm.createInternationalString("Purveyor of " +
        "the finest coffees. Established 1905");
org.setDescription(is);
```

```
// create postal address for organization
String streetNumber = "99";
String street = "Imaginary Ave. Suite 33";
String city = "Imaginary City";
String state = "NY");
String country = "USA");
String postalCode = "00000";
String type = "Type US";
PostalAddress postAddr =
     blcm.createPostalAddress(streetNumber, street, city, state,
        country, postalCode, type);
org.setPostalAddress(postAddr);

// create telephone number for organization
TelephoneNumber tNum = blcm.createTelephoneNumber();
tNum.setCountryCode("1");
tNum.setAreaCode("100");
tNum.setNumber("100-1000");
tNum.setType(CanonicalConstants.CANONICAL_PHONE_TYPE_CODE_OfficePhone);
Collection tNums = new ArrayList();
tNums.add(tNum);
org.setTelephoneNumbers(tNums);
```

The telephone number type is the value of a concept in the `PhoneType` classification scheme: `"OfficePhone"`, `"MobilePhone"`, `"HomePhone"`, `"FAX"`, or `"Beeper"`. Use the `CanonicalConstants` code for the phone type.

To create a hierarchy of organizations, use the `Organization.addChildOrganization` method to add one organization to another, or use the `Organization.addChildOrganizations` method to add a `Collection` of organizations to another.

Adding Services to an Organization

Most organizations publish themselves to a registry to offer services, so JAXR has facilities to add services to an organization. Typically, you first create the service by publishing a WSDL file (see "Creating Services by Publishing WSDL Files" on page 73). Then you add the service to the organization.

Like an `Organization` object, a `Service` object has a name, a description, and a unique key that is generated by the Registry when the service is registered. A `Service` object can also have classifications.

In addition to the attributes common to all objects, a service also commonly has *service bindings*, which provide information about how to access the service. A `ServiceBinding` object normally has a description and an access URI,.

The following code fragment shows how to locate a previously published service and add it to the organization. This example uses the service published in "Creating a Service by Publishing a WSDL File: Example" on page 74.

```
String serviceId = "urn:Foo:service:MyCoffeeService";
Service service = (Service) bqm.getRegistryObject(serviceId);
System.out.println("Service URN is " + serviceId);

Collection services = new ArrayList();
services.add(service);
org.addServices(services);
```

Creating Users

If you create an organization without specifying a primary contact, the default primary contact is the `User` object that created the organization (that is, the user whose credentials you set when you created the connection to the Registry). However, you can specify a different user as the primary contact.

A `User` is also a complex type of registry object. It normally includes the following attributes, in addition to those common to all objects:

- A `PersonName` object
- One or more `PostalAddress` objects
- One or more `TelephoneNumber` objects
- One or more `EmailAddress` objects
- One or more `URL` objects that represent the user's home page

---

Note – Typically, users create themselves by registering using the Web Console. It is highly uncommon to use JAXR to create a user. The sample programs create users to illustrate the `User` object and to generate organizations with different primary contacts.

The following code fragment creates a `User` and then sets that `User` as the primary contact for the organization. This `User` has a telephone number and email address but no postal address.

```
// Create primary contact, set name
User primaryContact = blcm.createUser();
String userId = primaryContact.getKey().getId();
System.out.println("User URN is " + userId);
PersonName pName =
     blcm.createPersonName("Jane", "M.", "Doe");
primaryContact.setPersonName(pName);

// Set primary contact phone number
TelephoneNumber pctNum = blcm.createTelephoneNumber();
```

```
pctNum.setCountryCode("1");
pctNum.setAreaCode("100");
pctNum.setNumber("100-1001");
pctNum.setType(CanonicalConstants.CANONICAL_PHONE_TYPE_CODE_MobilePhone);
Collection phoneNums = new ArrayList();
phoneNums.add(pctNum);
primaryContact.setTelephoneNumbers(phoneNums);

// Set primary contact email address
EmailAddress emailAddress =
blcm.createEmailAddress("jane.doe@TheCoffeeBreak.com");
emailAddress.setType(CanonicalConstants.CANONICAL_EMAIL_TYPE_CODE_OfficeEmail));
Collection emailAddresses = new ArrayList();
emailAddresses.add(emailAddress);
primaryContact.setEmailAddresses(emailAddresses);

URL pcUrl = new URL((bundle.getString("person.url"));
primaryContact.setUrl(pcUrl);

// Set primary contact for organization
org.setPrimaryContact(primaryContact);
```

The telephone number type for the primary contact is the value of a concept in the `PhoneType` classification scheme: `"OfficePhone"`, `"MobilePhone"`, `"HomePhone"`, `"FAX"`, or `"Beeper"`. The email address type for the primary contact is the value of a concept in the `EmailType` classification scheme: either `"OfficeEmail"` or `"HomeEmail"`. Use the `CanonicalConstants` codes for these types.

Creating an Organization: Examples

For examples of creating an organization, see `JAXRPublishOrg.java` and `JAXRPublishOrgNoPC.java` in the directory *INSTALL*`/registry-samples/organizations/src`.

The `JAXRPublishOrg` example creates an organization and its primary contact. It adds a service, the one published in "Creating a Service by Publishing a WSDL File: Example" on page 74. The example displays the unique identifiers for the organization, user, and service so that you can use the identifiers later when you delete the objects. This example creates a fictitious `User` as the primary contact for the organization. The name of the organization is `The ebXML Coffee Break`.

The other example, `JAXRPublishOrgNoPC.java`, does not set a primary contact for the organization. In this case, the primary contact by default is the `User` who is authenticated when you run the program. The name of this organization is `DefaultPCOrg`.

▼ To Run the `JAXRPublishOrg` and `JAXRPublishOrgNoPC` Examples

1. **Go to the directory** *INSTALL*`/registry-samples/organizations`**.**
2. **Type the following commands:**

   Ant-base`/ant pub-org`
   Ant-base`/ant pub-org-nopc`

Creating and Retrieving an Organization Hierarchy: Examples

For examples of publishing and retrieving an organization hierarchy, see `JAXRPublishOrgFamily.java` and `JAXRSearchOrgFamily.java` in the directory *INSTALL*`/registry-samples/organizations/src`.

▼ To Run the `JAXRPublishOrgFamily` and `JAXRSearchOrgFamily` Examples

1. **Go to the directory** *INSTALL*`/registry-samples/organizations`**.**
2. **Type the following command to publish the organizations:**

   Ant-base`/ant pub-fam`
3. **Type the following command to retrieve the organizations that you published:**

   Ant-base`/ant search-fam`

Saving Objects in the Registry

After you have created an object and set its attributes, you publish it to the Registry by calling either the `LifeCycleManager.saveObjects` method or an object-specific save method like `BusinessLifeCycleManager.saveOrganizations` or `BusinessLifeCycleManager.saveServices`. You always publish a collection of objects, not a single object. The save methods return a `BulkResponse` object that contains the keys (that is, the unique identifiers) for the saved objects. The following code fragment saves an organization and retrieves its key:

```
// Add organization and submit to registry
// Retrieve key if successful
Collection orgs = new ArrayList();
orgs.add(org);
BulkResponse response = blcm.saveObjects(orgs);
Collection exceptions = response.getExceptions();
if (exceptions == null) {
```

```
    System.out.println("Organization saved");

    Collection keys = response.getCollection();
    Iterator keyIter = keys.iterator();
    if (keyIter.hasNext()) {
        javax.xml.registry.infomodel.Key orgKey =
             (javax.xml.registry.infomodel.Key) keyIter.next();
        String id = orgKey.getId();
        System.out.println("Organization key is " + id);
    }
}
```

If one of the objects exists but some of the data have changed, the save methods update and replace the data. This may result in the creation of a new version of the object (see "Changing the State of Objects in the Registry" on page 85).

◆ ◆ ◆ CHAPTER 5

Managing Objects in the Registry

After you publish objects to Service Registry, you can perform operations on the objects. This chapter describes these operations.

Creating Relationships Between Objects: Associations

You can create an `Association` object and use it to specify a relationship between any two objects. The ebXML specification specifies an `AssociationType` classification scheme that contains a number of canonical concepts you can use when you create an `Association`. You can also create your own concepts within the `AssociationType` classification scheme.

The canonical association types are as follows:

- `AccessControlPolicyFor`
- `AffiliatedWith`, which has the subconcepts `EmployeeOf` and `MemberOf`
- `Contains`
- `ContentManagementServiceFor`
- `EquivalentTo`
- `Extends`
- `ExternallyLinks`
- `HasFederationMember`
- `HasMember`
- `Implements`

- `InstanceOf`
- `InvocationControlFileFor`, which has the subconcepts `CatalogingControlFileFor` and `ValidationControlFileFor`
- `OffersService`
- `OwnerOf`
- `RelatedTo`
- `Replaces`
- `ResponsibleFor`
- `SubmitterOf`
- `Supersedes`
- `Uses`

The Registry uses some of these association types automatically. For example, when you add a `Service` to an `Organization`, the Registry creates an `OffersService` association with the `Organization` as the source and the `Service` as the target.

Associations are directional: each `Association` object has a source object and a target object. Establishing an association between two objects is a three-step process:

1. Find the `AssociationType` concept that you want to use, or create one.
2. Use the `LifeCycleManager.createAssociation` method to create the association. This method takes two arguments, the target object and the concept that identifies the relationship.
3. Use the `RegistryObject.addAssociation` method to add the association to the source object.

For example, suppose you have two objects, `obj1` and `obj2`, and you want to establish a `RelatedTo` relationship between them. (In this relationship, which object is the source and which is the target is arbitrary.) First, locate the `RelatedTo` concept:

```
// Find RelatedTo concept for Association
String concString =
    CanonicalConstants.CANONICAL_ASSOCIATION_TYPE_ID_RelatedTo;
Concept relConcept = (Concept) bqm.getRegistryObject(concString);
```

Create the association, specifying `obj2` as the target:

```
Association relAssoc =
     blcm.createAssociation(obj2, relConcept);
```

Add the association to the source object, `obj1`:

```
obj1.addAssociation(relAssoc);
```

Finally, save the association:

```
Collection associations = new ArrayList();
associations.add(relAssoc1);
BulkResponse response = blcm.saveObjects(associations);
```

Associations can be of two types, intramural and extramural. You create an *intramural association* when both the source and target object are owned by you. You create an *extramural association* when at least one of these objects is not owned by you. The owner of an object can use an access control policy to restrict the right to create an extramural association with that object as a source or target.

Creating Associations: Example

For an example of creating an association, see `JAXRPublishAssociation.java` in the directory *INSTALL*`/registry-samples/publish-association/src/`. This example creates a `RelatedTo` association between any two objects whose unique identifiers you specify. For example, you could specify the two child organizations created in "Creating and Retrieving an Organization Hierarchy: Examples" on page 78.

▼ To Run the `JAXRPublishAssociation` Example

1. **Go to the directory** *INSTALL*`/registry-samples/organizations`**.**
2. **Retrieve the organization hierarchy by running the following command:**

   Ant-base`/ant search-fam`

   Notice the key ID strings of the two child organizations.
3. **Go to the directory** *INSTALL*`/registry-samples/publish-association`**.**
4. **Type the following command:**

   Ant-base`/ant run -Did1=`*string1* `-Did2=`*string2*

   Replace *string1* and *string2* with the two child organization ID strings.

   Whether the association is intramural or extramural depends upon who owns the two objects. In this case, the association is intramural.

Organizing Objects Within Registry Packages

Registry packages allow you to group a number of logically related registry objects, even if the individual member objects belong to different owners. A `RegistryPackage` is analogous to a directory or folder in a file system, and the registry objects it contains are analogous to the files in the directories or folders.

To create a `RegistryPackage` object, call the `LifeCycleManager.createRegistryPackage` method, which takes a `String` or `InternationalString` argument. Then call the `RegistryPackage.addRegistryObject` or `RegistryPackage.addRegistryObjects` method to add objects to the package.

For example, you could create a `RegistryPackage` object that is named "SunPackage":

```
RegistryPackage pkg =
     blcm.createRegistryPackage("SunPackage");
```

Then, after finding all objects with the string `"Sun"` in their names, you could iterate through the results and add each object to the package:

```
pkg.addRegistryObject(object);
```

A common use of packages is to organize a set of extrinsic objects. A registry administrator can use the `cp` command of the Admin Tool to load a file system into the Registry, storing the directories as registry packages and the files as the package contents. See "cp" in *Service Registry 3.1 Administration Guide* for details.

Organizing Objects Within Registry Packages: Examples

For examples of using registry packages, see `JAXRPublishPackage.java` and `JAXRSearchPackage.java` in the directory *INSTALL*`/registry-samples/packages/src`. The first example publishes a `RegistryPackage` object that includes all objects in the Registry whose names contain the string `"free"`. The second example searches for this package and displays its contents.

▼ To Run the `JAXRPublishPackage` and `JAXRSearchPackage` Examples

1. **Go to the directory** *INSTALL*`/registry-samples/packages`**.**
2. **Type the following command:**

   Ant-base`/ant pub-pkg`

3 **Type the following command:**

Ant-base`/ant search-pkg`

Changing the State of Objects in the Registry

You add an `AuditableEvent` object to the audit trail of an object when you publish the object to the Registry or when you modify the object in certain ways. See "Retrieving the Audit Trail of an Object" on page 52 for details on these events and on how to obtain information about them.

Many events are created as a side effect of some other action:

- Saving an object to the Registry creates an `EVENT_TYPE_CREATED` event.
-

  The following actions create an `EVENT_TYPE_VERSIONED` event if versioning is enabled for the object type of the object being modified:

  - Changing an object's name or description
  - Adding, modifying, or removing a `Classification`, `ExternalIdentifier`, or `Slot`
  - For an `Organization` or `User`, adding, modifying, or removing a `PostalAddress` or `TelephoneNumber`

    You can retrieve version information for an object. See "Retrieving the Version of an Object" on page 54 for details.

Note – At this release, versioning of objects is disabled by default. To enable versioning of objects, an administrator must perform the task described in "Enabling Versioning of Registry Objects" in *Service Registry 3.1 Administration Guide*. The administrator commonly enables versioning for some object types but not for all.

You can also change the state of objects explicitly. This feature may be useful in a production environment where different versions of objects exist and where you wish to use some form of version control. For example, you can approve a version of an object for general use and deprecate an obsolete version before you remove it. If you change your mind after deprecating an object, you can undeprecate it. As a registered user, you can perform these actions only on objects you own.

- You can approve objects by using the `LifeCycleManagerImpl.approveObjects` method. This feature is implementation-specific.
- You can deprecate objects by using the `LifeCycleManager.deprecateObjects` method.
- You can undeprecate objects by using the `LifeCycleManager.unDeprecateObjects` method.

The `LifeCycleManagerImpl.approveObjects` method has the following signature:

```
public BulkResponse approveObjects(java.util.Collection keys)
    throws JAXRException
```

The code to deprecate an object typically looks like this:

```
String id = id-string;
Key key = blcm.createKey(id);
Collection keys = new ArrayList();
keys.add(key);

// deprecate the object
blcm.deprecateObjects(keys);
```

It is possible to restrict access to these actions to specific users, user roles, and user groups, such as registry administrators. See "Controlling Access to Objects" on page 87.

No `AuditableEvent` is created for actions that do not alter the state of a `RegistryObject`. For example, queries do not generate an `AuditableEvent`, and no `AuditableEvent` is generated for a `RegistryObject` when it is added to a `RegistryPackage` or when you create an `Association` with the object as the source or target.

Changing the State of Objects in the Registry: Examples

For examples of approving, deprecating, undeprecating objects, see the examples in *INSTALL*`/registry-samples/auditable-events/src`: `JAXRApproveObject.java`, `JAXRDeprecateObject.java`, and `JAXRUndeprecateObject.java`. Each example performs an action on an object whose unique identifier you specify, then displays the object's audit trail so that you can see the effect of the example.

For all examples, the object that you specify must be one that you created.

▼ To Run the `JAXRApproveObject`, `JAXRDeprecateObject`, and `JAXRUndeprecateObject` Examples

1. **Go to the directory** *INSTALL*`/registry-samples/auditable-events`**.**
2. **Type the following command:**

   Ant-base`/`**`ant approve-obj -Did=`***id-string***
3. **Type the following command:**

   Ant-base`/`**`ant deprecate-obj -Did=`***id-string***

4 **Type the following command:**

Ant-base`/ant undeprecate-obj -Did=`*id-string*

Controlling Access to Objects

Access to objects in the Registry is set by access control policies (ACPs). The default access control policy specifies the following:

- The predefined user `Registry Guest` can read any object. All users have this identity when they are not logged in to the Registry.
- All registered users can create objects and can perform actions on objects they own.
- Any user classified as a `RegistryAdministrator` can perform actions on all objects in the Registry. By default, only the predefined user `Registry Operator` is classified as an administrator. For instructions on how to become an administrator, see "Creating an Administrator" in *Service Registry 3.1 Administration Guide*.

Very fine-grained access control on individual objects is possible through custom ACPs. However, writing an ACP is currently a manual process that requires knowledge of OASIS eXtensible Access Control Markup Language (XACML). For details, refer to Chapter 9, "Access Control Information Model," of ebXML RIM 3.0, especially the examples in Sections 9.7.6 through 9.7.8.

Removing Objects From the Registry and Repository

A registry allows you to remove from it any objects that you have submitted to it. You use the object's ID as an argument to the `LifeCycleManager.deleteObjects` method.

The following code fragment deletes the object that corresponds to a specified key string and then displays the key again so that you can confirm that it has deleted the correct one.

```
String id = key.getId();
Collection keys = new ArrayList();
keys.add(key);
BulkResponse response = blcm.deleteObjects(keys);
Collection exceptions = response.getException();
if (exceptions == null) {
    System.out.println("Objects deleted");
    Collection retKeys = response.getCollection();
    Iterator keyIter = retKeys.iterator();
    javax.xml.registry.infomodel.Key orgKey = null;
    if (keyIter.hasNext()) {
        orgKey =
             (javax.xml.registry.infomodel.Key) keyIter.next();
```

```
        id = orgKey.getId();
        System.out.println("Object key was " + id);
    }
}
```

Deleting an `Organization` object does not delete the `Service` and `User` objects that belong to the `Organization`. If you want to delete those objects, you must delete them separately.

Deleting a `Service` object deletes the `ServiceBinding` objects that belong to it. Deleting the `Service` and `ServiceBinding` objects, however, does not delete the associated `ExtrinsicObject` instances and their associated repository items. You must delete the extrinsic objects separately.

When you delete an object that has `Classification` or `ExternalIdentifier` or `Slot` objects, those objects are also deleted. However, if the object has `ExternalLink` objects, those objects are not deleted.

`AuditableEvent` objects are not deleted when the objects associated with them are deleted. You might find that as you use the Registry, a large number of these objects accumulates.

Removing Objects from the Registry: Example

For an example of deleting an object from the Registry, see `JAXRDelete.java` in the directory *INSTALL*`/registry-samples/delete-object/src`. This example deletes the object whose unique identifier you specify.

▼ To Run the `JAXRDelete` Example

1. **Go to the directory** *INSTALL*`/registry-samples/delete-object`**.**
2. **Type the following command:**

   Ant-base`/ant run -Did=`*id-string*

◆ ◆ ◆ CHAPTER 6

Developing Client Programs for the UDDI Interface

This chapter explains how to create client programs for the Universal Description, Discovery and Integration (UDDI) interface to Service Registry.

Creating Client Programs

Client programs can access the UDDI interface to Service Registry by using the SOAP 1.1 protocol over HTTP. Client programs in any programming language can access the UDDI interface service endpoint of Service Registry by using UDDI 3.0.2 Inquiry protocols. The endpoint for the UDDI Inquiry interface is as follows:

`http://`*host*`:`*port*`/soar/uddi/inquire`

The UDDI interface to Service Registry conforms to the UDDI 3.0.2 Inquiry API WSDL as defined at the following URLs:

- UDDI API Binding: `http://uddi.org/wsdl/uddi_api_v3_binding.wsdl`
- UDDI API Port Type: `http://uddi.org/wsdl/uddi_api_v3_portType.wsdl`

You can develop a Java client program for the UDDI interface using JAX-RPC 1.1 by generating the client stubs from the previously listed UDDI 3.0.2 WSDL files.

Additions and changes to the UDDI 3.0.2 WSDL and schemas to enable a Java client to be generated according to the requirements of the JAX-RPC 1.1 Specification are described in the UDDI Spec TC Technical Note at the following URL: `http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-jax-rpc-20050126.htm`.

The Java client program can then invoke methods on the UDDI Inquiry interface by using the methods exposed by the client stub.

In the current release of Service Registry, the UDDI interface does not support the UDDI 3.0.2 Publication, Security, Custody Transfer, or Subscription protocols. The following UDDI 3.0.2 interfaces are implemented to return `E_unsupported` (10050) error codes for every method:

```
http://host:port/soar/uddi/custody
http://host:port/soar/uddi/publish
http://host:port/soar/uddi/security
http://host:port/soar/uddi/subscription
```

The Inquiry interface implementation does not support authorization using `-authInfo` arguments or requests for partial results using either `-listHead` or `-maxRows` arguments.

Client programs that publish to the registry must use the JAXR API as described in earlier sections.

◆ ◆ ◆ CHAPTER 7

Troubleshooting

This chapter describes solutions to some problems that you can encounter when using JAXR with Service Registry.

See "Known Issues and Bugs" in *Service Registry 3.1 Release Notes* for details about other problems you might encounter, along with workarounds.

"Message Send Failed" Error from Service Registry

If you run a program and get a stack trace with a `Message send failed` error, the probable reason is that the Registry is not running or that the specified URL is incorrect. The beginning of the stack trace looks like this:

```
[java] Jul 10, 2006 10:23:09 PM com.sun.xml.messaging.saaj.client.p2p.HttpSOAPConnection post
[java] SEVERE: SAAJ0009: Message send failed
[java] com.sun.xml.messaging.saaj.SOAPExceptionImpl: java.security.PrivilegedActionException:
com.sun.xml.messaging.saaj.SOAPExceptionImpl: Message send failed
```

Make sure that the host name and port in *INSTALL*`/registry-samples/common/build.properties` are correct for your installation.

To make sure that the Registry is running, use the command-line or web interface to the Application Server domain for the Registry. For details, see "Administering the Application Server Domain for Service Registry" in *Service Registry 3.1 Administration Guide*.

Unable to Create ExternalLink or ServiceBinding

You might get an error in one of the following situations, even if you have specified the system properties `http.proxyHost` and `http.proxyPort`:

- When you specify an External URI for an `ExternalLink` object
- When you specify an Access URI for a `ServiceBinding` object

The error message looks like this:

```
The URL: uri is not resolvable.
Use Absolute Path Format [scheme:][//authority][path][?query][#fragment]
```

This error means that the administrative task described in "Configuring the Java Virtual Machine (JVM) for the Registry Domain" in *Service Registry 3.1 Administration Guide* has not been performed. The Service Registry administrator for your site needs to perform this task and restart the Registry before you can create these objects.

FileNotFoundException for Keystore File

An error like the following indicates that you did not create a keystore file:

```
[java] Query URL is http://localhost:6480/soar/registry/soap
[java] Created connection to registry
[java] java.io.FileNotFoundException:
/home/myname/soar/3.0/jaxr-ebxml/security/keystore.jks (No such file or directory)
```

If you see this error, follow all the procedures described in the section "Getting Access to the Registry" on page 23.

◆ ◆ ◆ APPENDIX A

Canonical Constants

This appendix lists the canonical constants for unique identifiers that are defined by the ebXML Registry and Repository specification. The constants are defined in the interface `org.freebxml.omar.common.CanonicalConstants`, which extends `org.freebxml.omar.common.CanonicalSchemes`.

These constants define the unique identifier strings for known objects. Use the constants to look up these objects by identifier.

The canonical constants for concepts defined in `org.freebxml.omar.common.CanonicalConstants` also include constants for the logical identifier (`lid`) of each concept and for the concept's code, which is its name. For example, the `MemberOf` concept has the following three constants:

- `CANONICAL_ASSOCIATION_TYPE_ID_Uses`, defined as `"urn:oasis:names:tc:ebxml-regrep:AssociationType:Uses"`
- `CANONICAL_ASSOCIATION_TYPE_LID_Uses`, defined as `"urn:oasis:names:tc:ebxml-regrep:AssociationType:Uses"`
- `CANONICAL_ASSOCIATION_TYPE_CODE_Uses`, defined as `"Uses"`

Classification schemes have constants for the unique identifier and the logical identifier, but do not have a code constant.

This appendix lists only the unique identifier constants, but you can use the `lid` and code constants where appropriate.

Constants for Classification Schemes

The constants for the unique identifiers of canonical classification schemes are as follows:

- CANONICAL_CLASSIFICATION_SCHEME_ID_AssociationType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ContentManagementService
- CANONICAL_CLASSIFICATION_SCHEME_ID_DataType
- CANONICAL_CLASSIFICATION_SCHEME_ID_DeletionScopeType
- CANONICAL_CLASSIFICATION_SCHEME_ID_EmailType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ErrorHandlingModel
- CANONICAL_CLASSIFICATION_SCHEME_ID_ErrorSeverityType
- CANONICAL_CLASSIFICATION_SCHEME_ID_EventType
- CANONICAL_CLASSIFICATION_SCHEME_ID_InvocationModel
- CANONICAL_CLASSIFICATION_SCHEME_ID_NodeType
- CANONICAL_CLASSIFICATION_SCHEME_ID_NotificationOptionType
- CANONICAL_CLASSIFICATION_SCHEME_ID_ObjectType
- CANONICAL_CLASSIFICATION_SCHEME_ID_PhoneType
- CANONICAL_CLASSIFICATION_SCHEME_ID_QueryLanguage
- CANONICAL_CLASSIFICATION_SCHEME_ID_ResponseStatusType
- CANONICAL_CLASSIFICATION_SCHEME_ID_StabilityType
- CANONICAL_CLASSIFICATION_SCHEME_ID_StatusType
- CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectGroup
- CANONICAL_CLASSIFICATION_SCHEME_ID_SubjectRole

Constants for Association Type Concepts

The constants for unique identifiers for the concepts that identify `Association` objects are as follows:

- CANONICAL_ASSOCIATION_TYPE_ID_AccessControlPolicyFor
- CANONICAL_ASSOCIATION_TYPE_ID_AffiliatedWith
- CANONICAL_ASSOCIATION_TYPE_ID_CatalogingControlFileFor
- CANONICAL_ASSOCIATION_TYPE_ID_Contains
- CANONICAL_ASSOCIATION_TYPE_ID_ContentManagementServiceFor
- CANONICAL_ASSOCIATION_TYPE_ID_EmployeeOf
- CANONICAL_ASSOCIATION_TYPE_ID_EquivalentTo
- CANONICAL_ASSOCIATION_TYPE_ID_Extends
- CANONICAL_ASSOCIATION_TYPE_ID_ExternallyLinks
- CANONICAL_ASSOCIATION_TYPE_ID_HasFederationMember
- CANONICAL_ASSOCIATION_TYPE_ID_HasMember
- CANONICAL_ASSOCIATION_TYPE_ID_Implements
- CANONICAL_ASSOCIATION_TYPE_ID_InstanceOf
- CANONICAL_ASSOCIATION_TYPE_ID_InvocationControlFileFor
- CANONICAL_ASSOCIATION_TYPE_ID_MemberOf

- CANONICAL_ASSOCIATION_TYPE_ID_OffersService
- CANONICAL_ASSOCIATION_TYPE_ID_OwnerOf
- CANONICAL_ASSOCIATION_TYPE_ID_RelatedTo
- CANONICAL_ASSOCIATION_TYPE_ID_Replaces
- CANONICAL_ASSOCIATION_TYPE_ID_ResponsibleFor
- CANONICAL_ASSOCIATION_TYPE_ID_SubmitterOf
- CANONICAL_ASSOCIATION_TYPE_ID_Supersedes
- CANONICAL_ASSOCIATION_TYPE_ID_Uses
- CANONICAL_ASSOCIATION_TYPE_ID_ValidationControlFileFor

Constants for Content Management Service Concepts

The constants for unique identifiers for the concepts that identify content management services are as follows:

- CANONICAL_CONTENT_MANAGEMENT_SERVICE_ID_ContentCatalogingService
- CANONICAL_CONTENT_MANAGEMENT_SERVICE_ID_ContentValidationService

Constants for Data Type Concepts

The constants for unique identifiers for the concepts that identify data types are as follows:

- CANONICAL_DATA_TYPE_ID_Boolean
- CANONICAL_DATA_TYPE_ID_Date
- CANONICAL_DATA_TYPE_ID_DateTime
- CANONICAL_DATA_TYPE_ID_Double
- CANONICAL_DATA_TYPE_ID_Duration
- CANONICAL_DATA_TYPE_ID_Float
- CANONICAL_DATA_TYPE_ID_Integer
- CANONICAL_DATA_TYPE_ID_ObjectRef
- CANONICAL_DATA_TYPE_ID_String
- CANONICAL_DATA_TYPE_ID_Time
- CANONICAL_DATA_TYPE_ID_URI

Constants for Deletion Scope Type Concepts

The constants for unique identifiers for the concepts that identify deletion scope types are as follows:

- CANONICAL_DELETION_SCOPE_TYPE_ID_DeleteAll
- CANONICAL_DELETION_SCOPE_TYPE_ID_DeleteRepositoryItemOnly

Constants for Email Type Concepts

The constants for unique identifiers for the concepts that identify email types are as follows:

- `CANONICAL_EMAIL_TYPE_ID_HomeEmail`
- `CANONICAL_EMAIL_TYPE_ID_OfficeEmail`

Constants for Error Handling Model Concepts

The constants for unique identifiers for the concepts that identify error handling models are as follows:

- `CANONICAL_ERROR_HANDLING_MODEL_ID_FailOnError`
- `CANONICAL_ERROR_HANDLING_MODEL_ID_LogErrorAndContinue`

Constants for Error Severity Type Concepts

The constants for unique identifiers for the concepts that identify error severity types are as follows:

- `CANONICAL_ERROR_SEVERITY_TYPE_ID_Error`
- `CANONICAL_ERROR_SEVERITY_TYPE_ID_Warning`

Constants for Event Type Concepts

The constants for unique identifiers for the concepts that identify event types are as follows:

- `CANONICAL_EVENT_TYPE_ID_Approved`
- `CANONICAL_EVENT_TYPE_ID_Created`
- `CANONICAL_EVENT_TYPE_ID_Deleted`
- `CANONICAL_EVENT_TYPE_ID_Deprecated`
- `CANONICAL_EVENT_TYPE_ID_Downloaded`
- `CANONICAL_EVENT_TYPE_ID_Relocated`
- `CANONICAL_EVENT_TYPE_ID_Undeprecated`
- `CANONICAL_EVENT_TYPE_ID_Updated`
- `CANONICAL_EVENT_TYPE_ID_Versioned`

Constants for Invocation Model Concepts

The constants for unique identifiers for the concepts that identify invocation models are as follows:

- `CANONICAL_INVOCATION_MODEL_ID_Decoupled`
- `CANONICAL_INVOCATION_MODEL_ID_Inline`

Constants for Node Type Concepts

The constants for unique identifiers for the concepts that identify node types are as follows:

- `CANONICAL_NODE_TYPE_ID_EmbeddedPath`
- `CANONICAL_NODE_TYPE_ID_NonUniqueCode`
- `CANONICAL_NODE_TYPE_ID_UniqueCode`

Constants for Notification Option Type Concepts

The constants for unique identifiers for the concepts that identify notification option types are as follows:

- `CANONICAL_NOTIFICATION_OPTION_TYPE_ID_ObjectRefs`
- `CANONICAL_NOTIFICATION_OPTION_TYPE_ID_Objects`

Constants for Object Type Concepts

The constants for unique identifiers for the concepts that identify object types are as follows:

- `CANONICAL_OBJECT_TYPE_ID_AdhocQuery`
- `CANONICAL_OBJECT_TYPE_ID_Association`
- `CANONICAL_OBJECT_TYPE_ID_AuditableEvent`
- `CANONICAL_OBJECT_TYPE_ID_Classification`
- `CANONICAL_OBJECT_TYPE_ID_ClassificationNode`
- `CANONICAL_OBJECT_TYPE_ID_ClassificationScheme`
- `CANONICAL_OBJECT_TYPE_ID_ExternalIdentifier`
- `CANONICAL_OBJECT_TYPE_ID_ExternalLink`
- `CANONICAL_OBJECT_TYPE_ID_ExtrinsicObject`
- `CANONICAL_OBJECT_TYPE_ID_Federation`
- `CANONICAL_OBJECT_TYPE_ID_Notification`
- `CANONICAL_OBJECT_TYPE_ID_Organization`
- `CANONICAL_OBJECT_TYPE_ID_Person`
- `CANONICAL_OBJECT_TYPE_ID_Registry`

- CANONICAL_OBJECT_TYPE_ID_RegistryObject
- CANONICAL_OBJECT_TYPE_ID_RegistryPackage
- CANONICAL_OBJECT_TYPE_ID_Service
- CANONICAL_OBJECT_TYPE_ID_ServiceBinding
- CANONICAL_OBJECT_TYPE_ID_SpecificationLink
- CANONICAL_OBJECT_TYPE_ID_Subscription
- CANONICAL_OBJECT_TYPE_ID_User

Constants for Extrinsic Object Types

The constants for unique identifiers for the concepts that specify extrinsic object types are as follows.

- CANONICAL_OBJECT_TYPE_ID_Policy
- CANONICAL_OBJECT_TYPE_ID_PolicySet
- CANONICAL_OBJECT_TYPE_ID_XACML
- CANONICAL_OBJECT_TYPE_ID_XForm
- CANONICAL_OBJECT_TYPE_ID_XHTML
- CANONICAL_OBJECT_TYPE_ID_XML
- CANONICAL_OBJECT_TYPE_ID_XMLSchema
- CANONICAL_OBJECT_TYPE_ID_XSLT

The extrinsic object constants listed above are defined in `org.freebxml.omar.common.CanonicalConstants`. The following constants for WSDL object type concepts, however, are defined in `org.freebxml.omar.common.profile.ws.wsdl.CanonicalConstants`:

- CANONICAL_OBJECT_TYPE_ID_WSDL
- CANONICAL_OBJECT_TYPE_ID_WSDL_BINDING
- CANONICAL_OBJECT_TYPE_ID_WSDL_PORT
- CANONICAL_OBJECT_TYPE_ID_WSDL_PORT_TYPE
- CANONICAL_OBJECT_TYPE_ID_WSDL_SERVICE

Constants for Phone Type Concepts

The constants for unique identifiers for the concepts that identify phone types are as follows:

- CANONICAL_PHONE_TYPE_ID_Beeper
- CANONICAL_PHONE_TYPE_ID_FAX
- CANONICAL_PHONE_TYPE_ID_HomePhone
- CANONICAL_PHONE_TYPE_ID_MobilePhone
- CANONICAL_PHONE_TYPE_ID_OfficePhone

Constants for Query Language Concepts

The constants for unique identifiers for the concepts that identify query languages are as follows:

- `CANONICAL_QUERY_LANGUAGE_ID_ebRSFilterQuery`
- `CANONICAL_QUERY_LANGUAGE_ID_SQL_92`
- `CANONICAL_QUERY_LANGUAGE_ID_XPath`
- `CANONICAL_QUERY_LANGUAGE_ID_XQuery`

Constants for Response Status Type Concepts

The constants for unique identifiers for the concepts that identify response status types are as follows:

- `CANONICAL_RESPONSE_STATUS_TYPE_ID_Failure`
- `CANONICAL_RESPONSE_STATUS_TYPE_ID_Success`
- `CANONICAL_RESPONSE_STATUS_TYPE_ID_Unavailable`

Constants for Stability Type Concepts

The constants for unique identifiers for the concepts that identify stability types are as follows:

- `CANONICAL_STABILITY_TYPE_ID_Dynamic`
- `CANONICAL_STABILITY_TYPE_ID_DynamicCompatible`
- `CANONICAL_STABILITY_TYPE_ID_Static`

Constants for Status Type Concepts

The constants for unique identifiers for the concepts that identify status types are as follows:

- `CANONICAL_STATUS_TYPE_ID_Approved`
- `CANONICAL_STATUS_TYPE_ID_Deprecated`
- `CANONICAL_STATUS_TYPE_ID_Submitted`
- `CANONICAL_STATUS_TYPE_ID_Withdrawn`

Constants for Subject Role Concepts

The constants for unique identifiers for the concepts that identify subject roles are as follows:

- `CANONICAL_SUBJECT_ROLE_ID_ContentOwner`
- `CANONICAL_SUBJECT_ROLE_ID_Intermediary`
- `CANONICAL_SUBJECT_ROLE_ID_RegistryAdministrator`
- `CANONICAL_SUBJECT_ROLE_ID_RegistryGuest`

Constants for Stored Queries

The constants defined for predefined queries are as follows:

- `CANONICAL_QUERY_BasicQuery`
- `CANONICAL_QUERY_BasicQueryCaseSensitive`
- `CANONICAL_QUERY_FindAllMyObjects`
- `CANONICAL_QUERY_FindObjectByIdAndType`
- `CANONICAL_QUERY_GetAuditTrailForRegistryObject`
- `CANONICAL_QUERY_GetCallersUser`
- `CANONICAL_QUERY_GetClassificationSchemesById`
- `CANONICAL_QUERY_GetRegistryPackagesByMemberId`

Additional constants for WSDL queries are defined in the interface `omar.common.profile.ws.wsdl.CanonicalConstants:`

- `CANONICAL_QUERY_WSDL_DISCOVERY`
- `CANONICAL_QUERY_SERVICE_DISCOVERY`
- `CANONICAL_QUERY_PORT_DISCOVERY`
- `CANONICAL_QUERY_BINDING_DISCOVERY`
- `CANONICAL_QUERY_PORTTYPE_DISCOVERY`

Index

D

E

F

G

O

P

Q

R

S

T

U

V

W