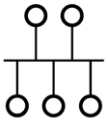


Orbix®

Web Services Development Tools

Version 6.1, December 2003

Making Software Work Together™

Updated: 15-Dec-2003

M 3 1 4 0

Contents

List of Tables

Preface

Audience

This guide is aimed at developers who wish to construct Web services. Java or other programming experience is assumed. Little prior knowledge of Web services and related technologies is required.

Updated documentation

The latest documentation updates can be found at http://www.iona.com/docs/.

Additional resources

The IONA knowledge base contains helpful articles, written by IONA experts, about IONA products. You can access the knowledge base at the following location:

http://www.iona.com/support/kb/

The IONA update center contains the latest releases and patches for IONA products:

http://www.iona.com/support/update/

Typographical conventions

This guide uses the following typographical conventions:

`Constant width`

Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class.

Constant width paragraphs represent code examples or information a system displays on the screen. For example:

```
#include <stdio.h>
```

Italic	Italic words in normal text represent *emphasis* and *new terms*. Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*`your_name`* **Note:** Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying conventions

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, Windows NT, Windows 95, or Windows 98 command prompt.
. . . . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

CHAPTER 1

Building Web Services and Clients

With the Web Service Builder GUI tool, you can build and deploy Web services and Web service clients. This chapter gives an overview of how to use Web Service Builder.

In this chapter

This chapter discusses the following topics:

Web Service Development

Overview

You can use Web Service Builder to generate Web services from many types of existing applications. For example, you can automatically generate Web services from Java classes and CORBA resources. Table 1 summarizes the various options that are available for generating a Web service.

Table 1: *Web Service Development Scenarios*

Basis for Web Service	Description
Java Class	Transform a Java implementation of an existing application into a Web service. The Java class can be on the classpath or stored in an archive file, of types XAR, ZIP, or JAR.
CORBA	Transform a CORBA object into a Web service.
Operation Flow	Graphically create operation flows that combine input and output from various methods and data to create the desired Web service functionality. Web Service Builder can then automatically generate a Web service from the operation flow.
Schema	Create a Web service from an existing schema (`.xsd`). This service enables client and server applications to exchange documents that use the same XML schema. The service uses a DOM handler, which processes the input document.
Schema Map	Graphically create associations between the elements of different XML schemas to produce a mapping. This schema map can then be use to produce a Web service, whose schema mapping enables clients and servers that use different schemas to exchange documents.

Table 1: *Web Service Development Scenarios*

Basis for Web Service	Description
WSDL	Create a Web service from any existing Web service's WSDL. This lets you refactor an existing Web service into your own implementation of a new Web service.

Input and output for Web Service Builder

To create a Web service, one of the input file types is always required, and a XAR file is always produced. Web Service Builder can produce the other kinds of output listed after the Web service is created, if desired.

Starting the Web Service Builder

Overview

Start Web Service Builder in one of the following ways:

- Launch the Web Service Builder from the IONA central toolbar, or
- From the command line, run the `itws_builder[.bat]` script.

Note: Some Web Service Builder features are enabled only if the Web services container is running.

Project hierarchy

Web service application information is organized into a hierarchical tree of projects. Double-click a project to display its applications. Double-click an application to display the services within each application. Continue double-clicking lower-level tree items to display more information about each Web service. The work area displays information about the item selected in the projects area.

Data Entries

Only ASCII text should be used in text fields of Web Service Builder because the WSDL generated and the resulting URLs produced should remain human-readable.

Java Classes

When a Web service is generated from a Java class, the generated Web service interface is based on the public methods defined on the target Java class.

Inherited Methods

When building a Web service from Java classes or CORBA objects, inherited methods can also be included in the Web service.

Interoperability

When you build a Web service using Web Service Builder, the result is standards-compliant and interoperable. Interoperability is verified against Microsoft's .NET toolkit and MS SOAP. Thus, a SOAP client built using Microsoft tools can access XMLBus Web services just like any other Web service.

Client Development

Generate clients and servers

The Web Service Builder can also generate client code for accessing a Web service and Java skeleton code that you can use to write your own Web service implementation.

CHAPTER 2

Listing Web Services

The Web Services Manager tool lists deployed Web services, obtains Web service URLs, and displays the WSDL of Web services.

In this chapter

This chapter discusses the following topics:

Listing in other formats

The Web services container can dynamically generate lists of deployed Web services and the locations of the WSDL documents that describe them, using either the standard Web Service Inspection Language (WSIL) or the older Microsoft-specific DISCO language. Integrated development

environments (IDEs) that are WSIL and DISCO-aware (such as Visual Studio) can use these dynamically-generated lists to make it easier to build Web service clients.

Starting Web Services Manager

Overview

Web Services Manager is a graphical tool that shows a server's deployed Web services. You can use Web Services Manager to list and undeploy Web services. You can also use Web Services Manager to manage the availability of any deployed Web service by activating or deactivating the Web service's endpoints.

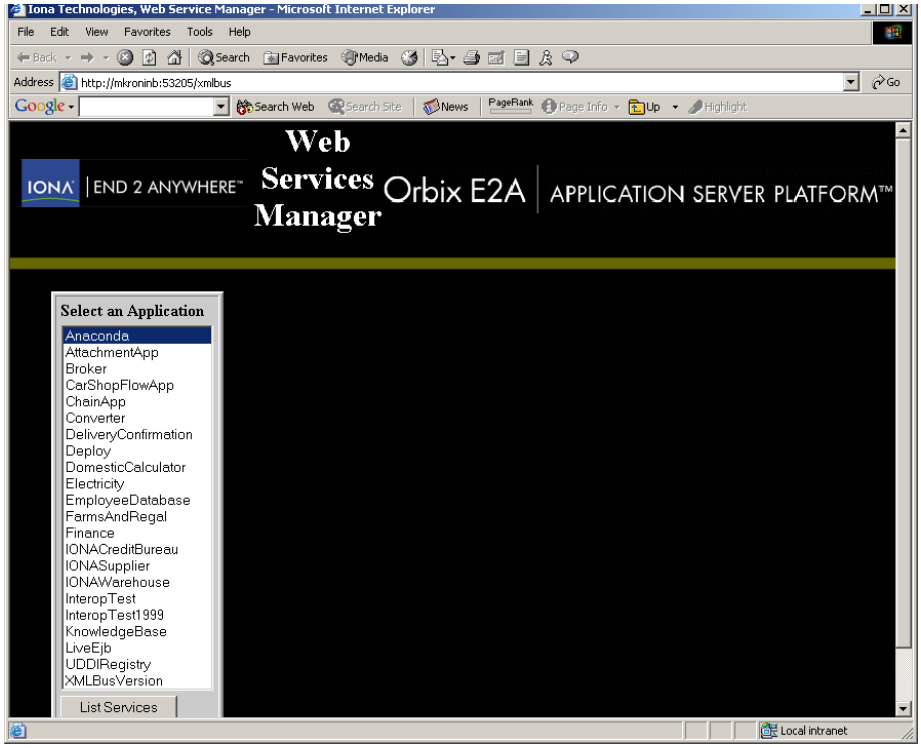
Note: In order to run Web Services Manager, the Web services container must also be running.

Start Web Services Manager

Start Web Services Manager in one of the following ways:

- Launch the Web Services Manager from the IONA Central Toolbar, or
- Enter the following URL into a Web browser:
  `http://localhost:53205/xmlbus/container?admin=true`
- In a secure domain, enter the following URL into a Web browser:
  `https://`*HostName*`:53206/xmlbus/container?admin=true`

When Web Services Manager starts, the following window displays:



Web Services Manager shows the list of deployed Web service applications.

Listing Deployed Services

Follow these steps:

1. Start Web Services Manager.

   All applications that are deployed into the Web services container display under **Select an Application**.

   Note: Refresh your browser to update the list if applications have been recently deployed or undeployed.

2. Select an application from the list.
3. Click **List Services**. The Web service or services that the application supports are displayed.

Listing Web Service Endpoints

Overview

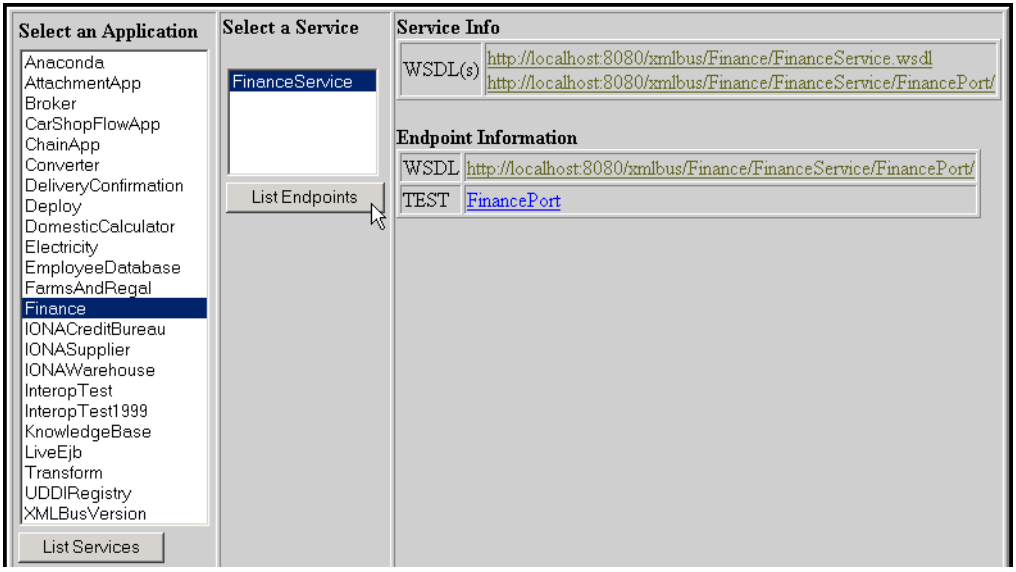
A port name identifies the interface to your Web service and is an address to a particular Web service implementation where SOAP messages or XML documents are sent. In WSDL, this is also known as an endpoint, which is a binding to a specific network transport protocol and represents a set of operations that use a particular message format. A Web service can have more than one endpoint representing different services within the Web service, or different running invocations of the same Web service.

List endpoints

Follow these steps to list Web service endpoints.

1. Start Web Services Manager.
2. Select an application.
3. Click **List Services** to display the Web services the application supports.
4. From **Select a Service**, choose a Web service.

5. Click **List Endpo**ints. The endpoint data is displayed as follows:



Endpoint data

The data under **Endpoint Information** offers the following data and links:

- **WSDL URL**: Each endpoint has its WSDL's URL listed under **Endpoint Information**.You can view the endpoint's WSDL by clicking on the URL. You can also copy the URL for other tool input, scripts, and so on.
- Test endpoint methods: You can try out the methods of a Web service by clicking on the endpoint name next to **Test**. This starts Web Services Test Client.

Listing Web Services in WSIL

Overview

The Web Service Inspection Language (WSIL) is a language for identifying a set of Web Service endpoints and where to find the Web Service Description Language (WSDL) documents that describe each endpoint. WSIL is useful because it lets you find Web services without having to know the exact URLs for the WSDL definitions. A WSIL document is generated for each Web services container.

Generating WSIL

To generate WSIL, enter the following URL in your browser (assuming that the Web services container is running on port 53205 on your local host):

```
http://localhost:53205/xmlbus/inspection.wsil
```

After entering the appropriate URL, your browser displays XML that lists the location of all deployed Web services.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
    xmlns:wsilwsdl="http://schemas.xmlsoap.org/ws/2001/10/inspection/wsdl/">
  - <service>
    - <description location="http://localhost:8080/xmlbus/InteropTest1999/InteropTest1999Service.wsdl"
        referencedNamespace="http://schemas.xmlsoap.org/wsdl/">
      - <wsilwsdl:reference endpointPresent="true">
          <wsilwsdl:referencedService
            xmlns:ns1="http://soapinterop.org/">ns1:InteropTest1999PortBinding</wsilwsdl:referencedService>
        </wsilwsdl:reference>
      </description>
    </service>
  - <service>
    - <description location="http://localhost:8080/xmlbus/InteropTest1999/InteropTest1999Service/InteropTest1999Port/
        referencedNamespace="http://schemas.xmlsoap.org/wsdl/">
      - <wsilwsdl:reference endpointPresent="true">
          <wsilwsdl:referencedService
            xmlns:ns1="http://soapinterop.org/">ns1:InteropTest1999PortBinding</wsilwsdl:referencedService>
        </wsilwsdl:reference>
      </description>
    </service>
```

URL for WSIL-aware tools

Web Services Manager's default web page also contains a meta tag that can automatically redirect WSIL-aware client tools to the full WSIL URL.

Listing Web Services in DISCO

Overview

XMLBus supports the generation of DISCO documents on a per-container basis to make integration with .NET clients easier. DISCO is a proprietary language used only by Microsoft .NET tools, which supports Web service inspection. While WSIL will eventually supersede DISCO as the industry standard, Microsoft .NET tools currently produce and consume DISCO documents.

This section discusses the following topics:

Generating DISCO

To generate DISCO documents, enter the following URL in your browser (assuming that the Web services container is running on port 53205 on your local host):

```
http://localhost:53205/xmlbus/default.disco
```

After entering the appropriate URL, your browser displays DISCO-formatted XML that lists the location of all deployed Web services.

URL for DISCO-aware tools

The DISCO URL is also available via an implicit link in the Web Services Manager's default web page. The Web Services Manager's default web page can automatically redirect DISCO-aware client tools to the full DISCO URL.

Integration with Visual Studio.NET

Overview

If you use Microsoft's Visual Studio.NET development environment to build a client for an XMLBus Web service, you must add a reference to the service's WSDL file. This is done through the **Add Web Reference** dialog box, which is accessible from the Visual Studio.NET **Project** menu.

Add Web Reference

The **Add Web Reference** dialog is a simple Web browser that recognizes WSDL and DISCO files. If you type in the URL for the Web services container, the dialog box downloads the implicitly referenced DISCO file and presents a list of all the Web services currently deployed in the Web services container. When you select a service from the list, Visual Studio.NET asks XMLBus for the service's WSDL document, and uses it to generate client proxy code.

See also "Listing Web Services in DISCO" on page 15.

CHAPTER 3

Testing Web Service Methods

Web Services Test Client is a browser-based graphical tool that you can use to dynamically test active Web services.

You provide Web Services Test Client with the URL location of the WSDL file that describes the Web service. When the Web service's methods are displayed, you can select a method, enter input values, and invoke the method.

Limitations

The following restrictions apply:

- Processing the document style of interaction is not supported.
- Processing the literal style of encoding is not supported.

In this chapter

This chapter discusses the following topics:

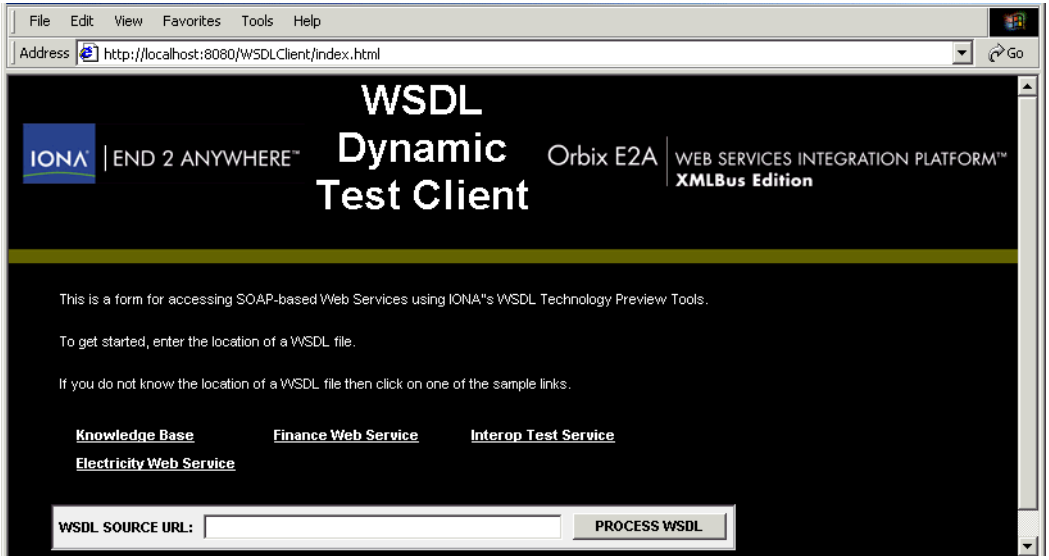
Starting Web Services Test Client

Overview

Start Web Services Test Client in one of the following ways:

- Launch the Web Services Test Client from the IONA Central Toolbar, or
- Enter the following URL into a Web browser:
  `http://localhost:53205/WSDLClient`
- In a secure domain, enter the following URL into a Web browser:
  `https://`*HostName*`:53206/WSDLClient`

The following window displays:



Note: The Web services container must be running to use Web Services Test Client.

Display

Web Services Test Client shows the following information:

- A brief explanation of the tool.
- A list of links to some sample Web services that come with XMLBus.

- A form in which to input a URL that represents the location of any Web service's WSDL.

Selecting a Web service

Select one of the sample Web services displayed, or enter into the form a URL such as the following:

```
http://localhost:port/xmlbus/Finance/FinanceService.wsdl
```

Testing Method Calls

Overview

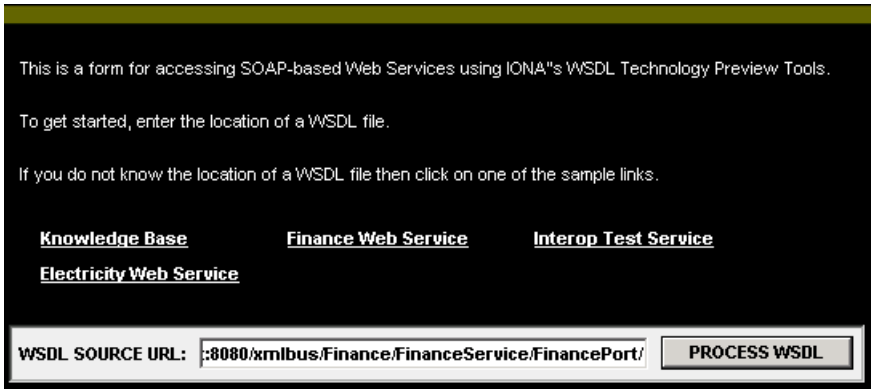
Web Services Test Client shows the Web service's methods in a Web page where you can select a method, enter appropriate input values for the method, then invoke the method. Web Services Test Client shows the results of the executed method call, along with any relevant SOAP messages.

The SOAP messages are useful as an aid in debugging. For example, the messages can indicate an invalid input or the possibility that the Web service is no longer active to receive messages.

Steps

Follow these steps to test a Web service:

1. Start Web Services Test Client.
2. Enter a WSDL URL into the **WSDL SOURCE URL** form. For example:

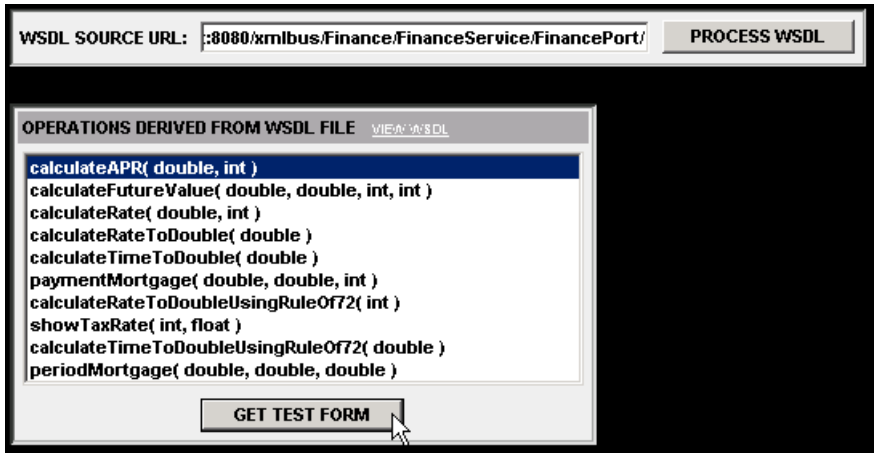


There are several ways to obtain a WSDL URL:

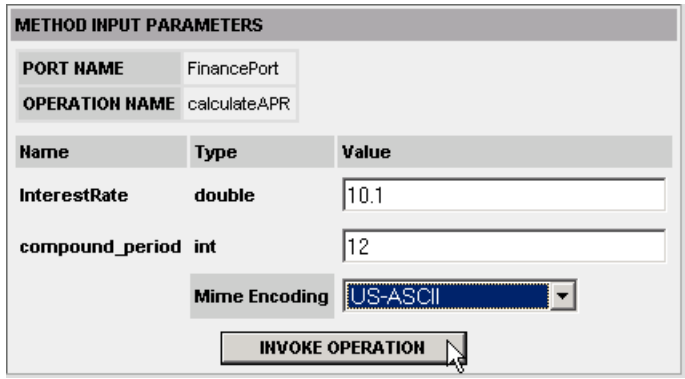
- Select one of the sample Web services displayed with Web Services Test Client.
- Run Web Services Manager and copy the WSDL file's URL (see page 13).

- Enter a URL of any known WSDL.

3. Click **PROCESS WSDL** to produce the list of methods available for the Web service:

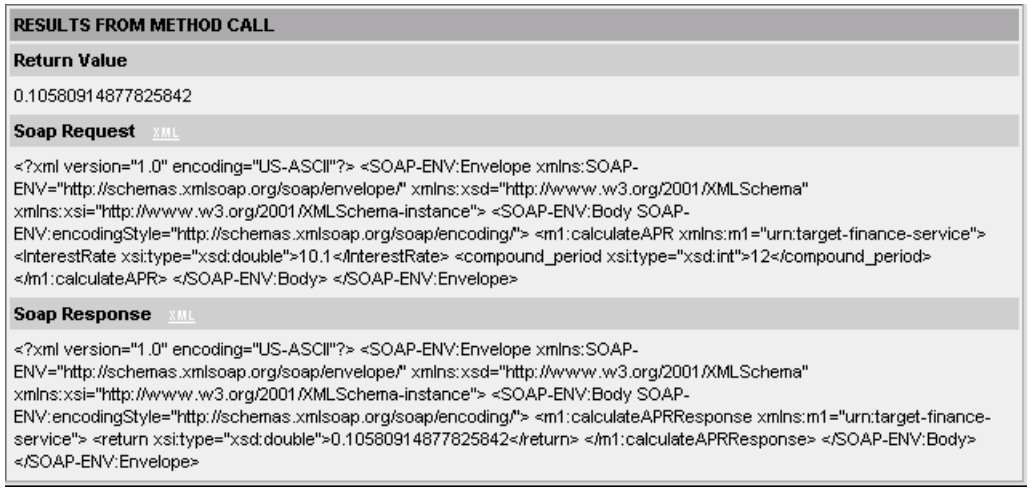


4. Select one of the methods and click **Get Test Form**. The **METHOD INPUT PARAMETERS** dialog is displayed:



5. Enter a value for each method parameter. In this example, a `double` value of 10.1 and an `int` value of 12 are entered.

6. The default MIME encoding is UTF-8. If your internationalization needs require it, select a different MIME encoding type from the drop-down list.
7. Click **INVOKE OPERATION** to invoke the method with the specified parameter values. The **RESULTS FROM METHOD CALL** dialog displays the results:



RESULTS FROM METHOD CALL

Return Value

0.10580914877825842

Soap Request XML

<?xml version="1.0" encoding="US-ASCII"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> <m1:calculateAPR xmlns:m1="urn:target-finance-service"> <InterestRate xsi:type="xsd:double">10.1</InterestRate> <compound_period xsi:type="xsd:int">12</compound_period> </m1:calculateAPR> </SOAP-ENV:Body> </SOAP-ENV:Envelope>

Soap Response XML

<?xml version="1.0" encoding="US-ASCII"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> <m1:calculateAPRResponse xmlns:m1="urn:target-finance-service"> <return xsi:type="xsd:double">0.10580914877825842</return> </m1:calculateAPRResponse> </SOAP-ENV:Body> </SOAP-ENV:Envelope>

CHAPTER 4

Monitoring and Testing SOAP Messages

Most Web service clients and servers communicate using the SOAP and HTTP protocols. It is sometimes useful for debugging Web services to know exactly what was sent to and received from the Web service. XMLBus provides the SOAP Message Test Client graphical tool for monitoring SOAP messages to and from servers at specified endpoints. With this tool, you can enter specific SOAP messages, send them to servers, and view the responses. XMLBus also provides a SOAP message logging facility on the server.

In this chapter

This chapter discusses the following topics:

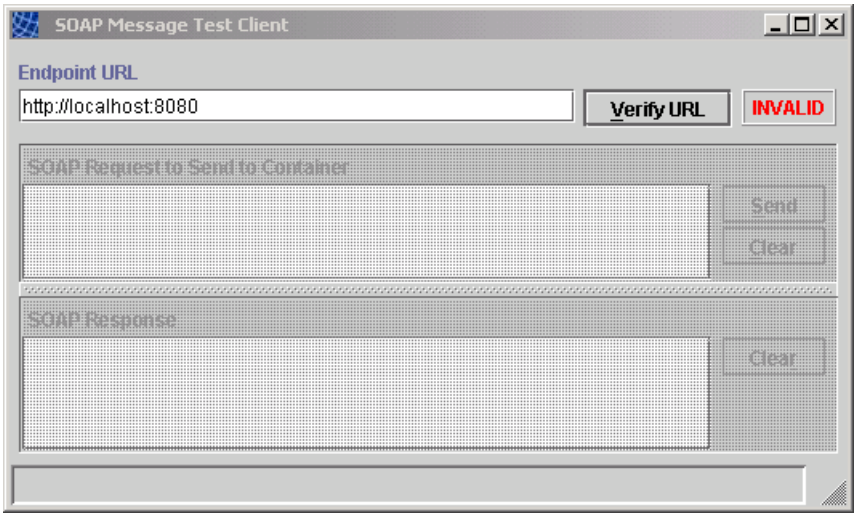
Starting the SOAP Message Test Client

Note: The Web services container must be running to use the SOAP Message Test Client.

Start the SOAP Message Test Client in one of the following ways:

- Launch the Test Client from the IONA central toolbar, or
- From the `bin` directory, run the script `itws_msgtestclient[.bat]`.

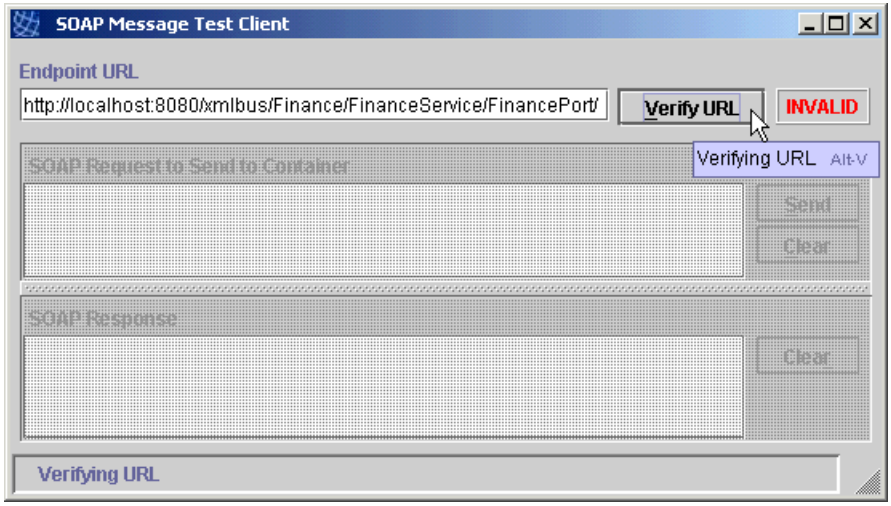
The following dialog displays:



Obtaining and Verifying an Endpoint URL

Follow these steps to obtain and verify a URL.

1. Obtain the endpoint URL for the Web service from an appropriate source, such as from Web Services Manager.
2. Paste the URL into the **Endpoint URL** of the SOAP Message Test Client:



3. Click **Verify URL**. If the URL is for a valid, running Web service, **VALID** is displayed.

Sending Test SOAP Messages

Overview

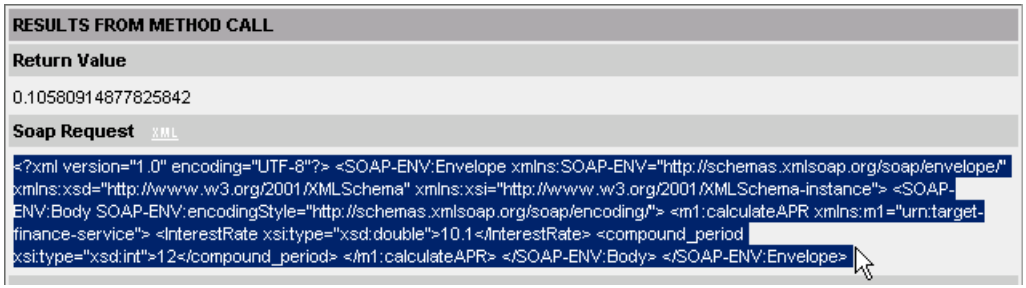
You can enter any SOAP messages into the SOAP Message Test Client and observe the responses. If you are very familiar with SOAP request message formats, you can type a message into the SOAP Message Test Client directly, or you can copy a SOAP request message from some other source, such as Web Services Test Client. See "Testing Method Calls" on page 20 for how to use Web Services Test Client.

Steps

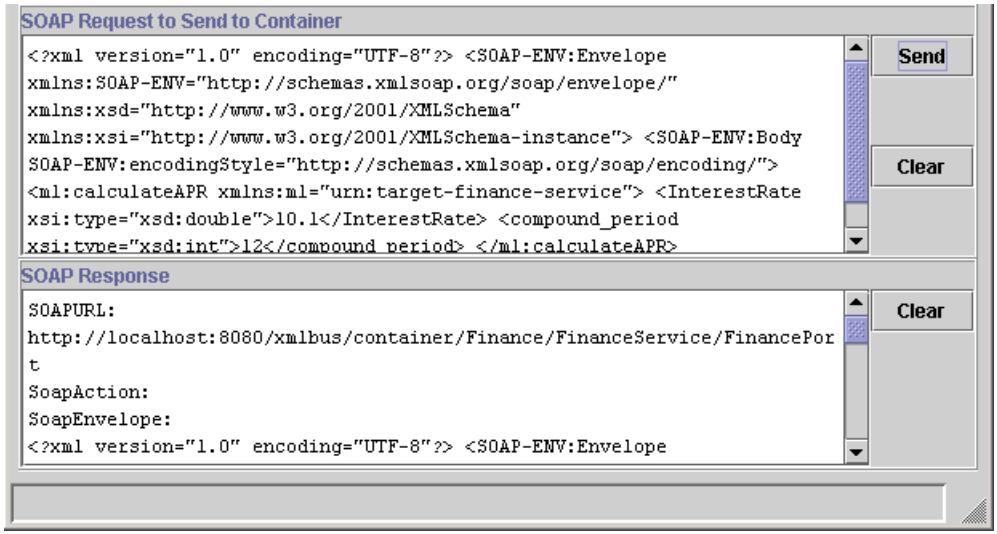
Follow these steps to test a SOAP message.

1. Obtain a SOAP message to test.

   For this example, copy a SOAP request message from the **RESULTS FROM METHOD CALL** panel of Web Services Test Client:



2. Enter a SOAP message into the **SOAP Request To Send to Container** panel. In the following example, the SOAP message is copied from Web Services Test Client and pasted into the SOAP request message:



3. Click **Send**.

Observe the complete request-response streams in the **SOAP Response** panel. For this example, the complete information is as follows:

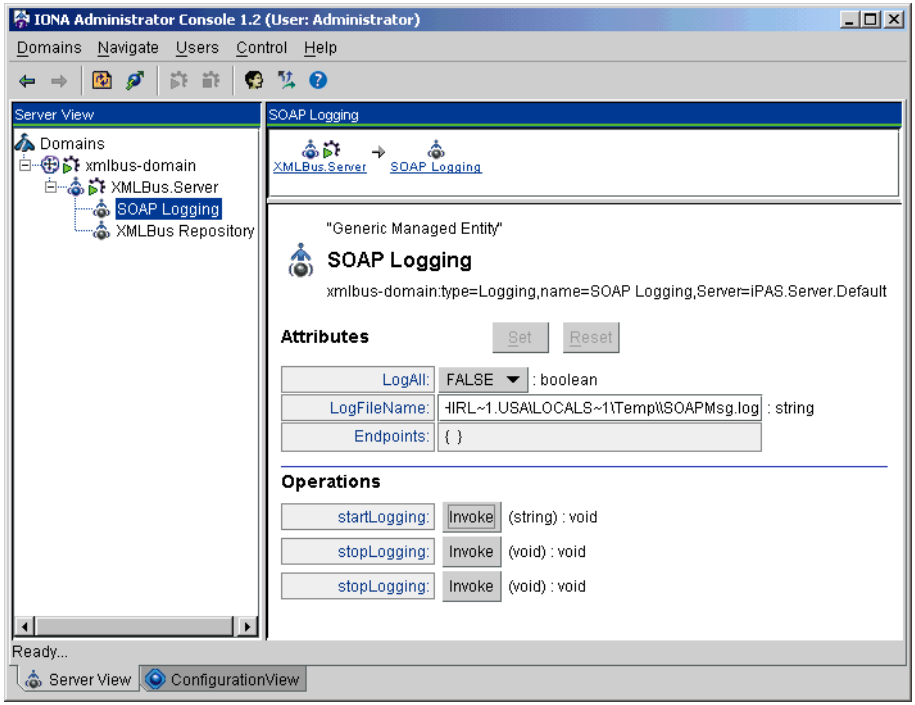
```
SOAPURL:
   http://localhost:8080/xmlbus/container/Finance/FinanceService
   /FinancePort
SoapAction:
SoapEnvelope:
<?xml version="1.0" encoding="UTF-8"?> <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod
   ing/"> <m1:calculateAPR
   xmlns:m1="urn:target-finance-service"> <InterestRate
   xsi:type="xsd:double">10.1</InterestRate> <compound_period
   xsi:type="xsd:int">12</compound_period> </m1:calculateAPR>
   </SOAP-ENV:Body> </SOAP-ENV:Envelope>
Response:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-E
   NV:Body
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod
   ing/"><m1:calculateAPRResponse
   xmlns:m1="urn:target-finance-service"><return
   xsi:type="xsd:double">0.10580914877825842</return></m1:calcul
   ateAPRResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

4. You can easily edit the SOAP request message, click **Send** again, and observe the new results.
5. Click **Clear** to clear the associated SOAP message panel in preparation for another message.

SOAP Message Logging

Overview

Server-side SOAP Message logging is a mechanism that logs the SOAP requests that come to the Web services container and the SOAP responses sent by the container. The SOAP messages are redirected to a specified file. You can control the logging by using the IONA Administrator tool:



The IONA Administrator shows the `SOAPLogging` bean which has the following attributes:

`LogAll (on/off)` Enable or disable logging for all endpoints.

`LogFileName(String filename)` Set the name of log file.

The `SOAPLogging` bean has the following operations:

`startLogging(String endpointName)`	Start logging for specified endpoint name.
`stopLogging(String endpointName)`	Turn off logging for specified endpoint.
`getEndpoints()`	List endpoints for which logging is currently on.
`stopLogging()`	Stop logging for all endpoints.

CHAPTER 5

Using the Registry Manager

IONA's Web Services Registry Manager lets you build and browse Web service UDDI version 2 registries.

The Registry Manager implements Sun Microsystem's Java API for XML Registries (JAXR) specification, which provides a superset of interfaces to UDDI and ebXML. This implementation is aimed at users who are already familiar with UDDI and JAXR.

What is a Web services registry?

Like SOAP and WSDL, JAXR/UDDI provide specifications for a core Web service technology. A Web services registry contains categorized information about businesses and the services that they offer, and associates those services with the Web service's WSDL description. Users can query the registry to find desired services and their WSDL descriptions.

The Registry Manager provides a browser that lets you query your own or third-party registries. You can also use the manager to add, update, and delete registry entries.

Each installation provides a private registry, which you can populate with your own service entries, for internal and external use.

Starting

Start the Registry Manager from the command line as follows:

```
itws_registrymanager[.bat]
```

In this chapter

This chapter is divided into the following sections:

Connecting to a Registry

Overview

You can connect to the installed private registry; you can also connect to a public registry. For example, the following companies currently maintain public registries:

- IBM
- Microsoft
- Systinet

Connection requirements

In order to connect to a registry, the following fields require valid entries:

Registry User/Password: Required in order to publish to a registry. User names and passwords are set in the XML file `etc/domains/`*Domain*`/securityInfo.xml`. In order to access public registries, you must obtain the required credentials. For more information about securing access to your own registries, see the Securing Web Services section in the *Security Guide*.

Query URL: The URL for browser connections.

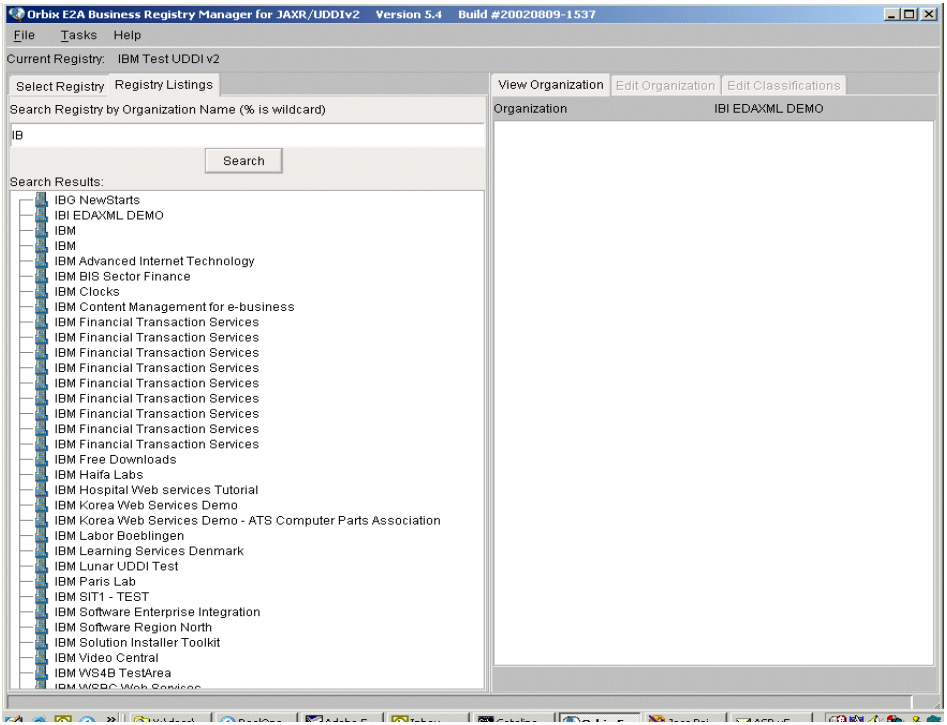
Publish URL: The URL for connections to registries where you can add new entries and edit existing data. In the current release, you can only edit instances of IONA registries.

After you supply the required data, click **Connect**.

Browsing a Registry

Overview

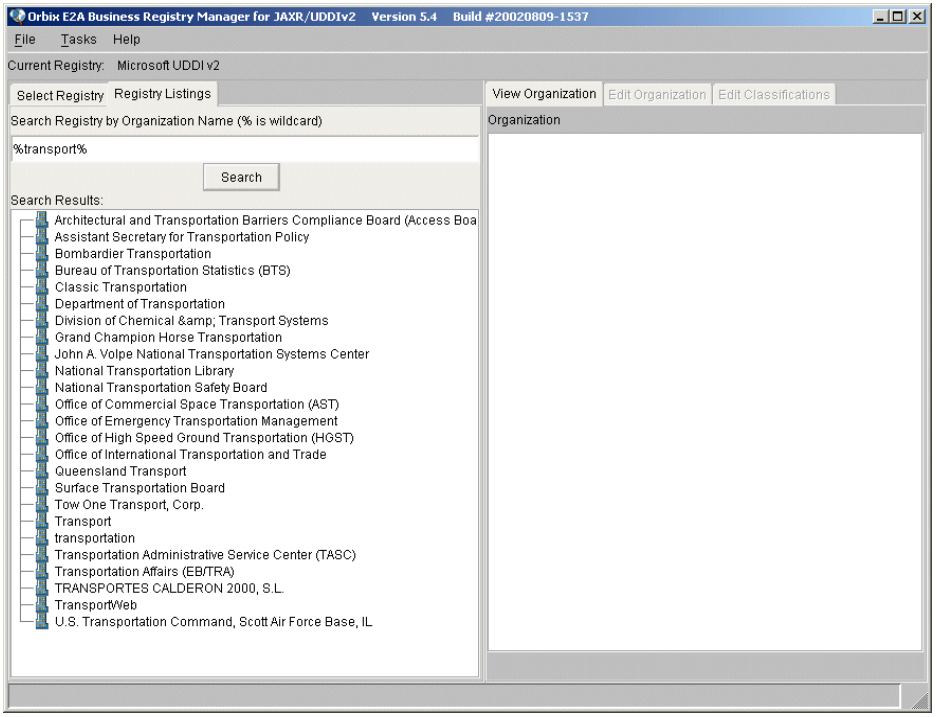
After you connect to a registry, you can browse its contents. To do so, create a selection set of registered business entities. Specify the set by entering a query in the Search field, and pressing **Search**. The following figure shows the results of a query for all business entities that begin with `IB`:



Note: Queries are case-insensitive.

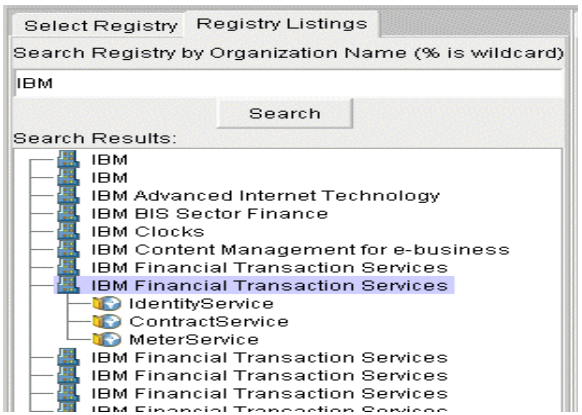
Wildcards

Some registries also support wildcard queries. For example, the Microsoft UDDI registry allows the wildcard `%`. Thus, querying this registry with `%transport%` finds all business entities whose names include the string `transport`:

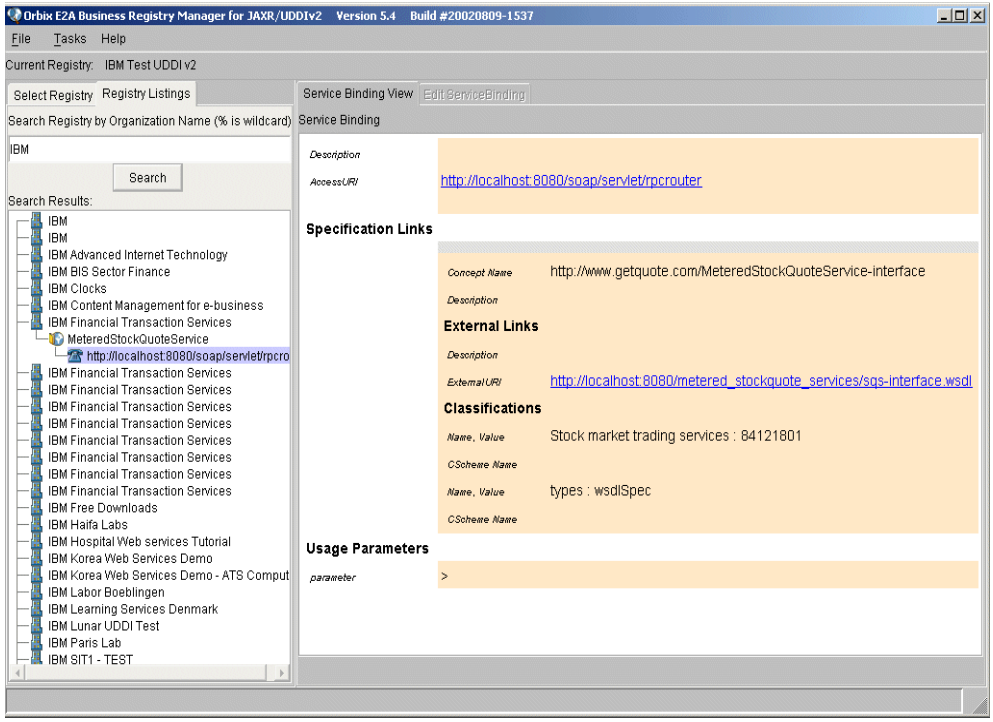


Listing Object Details

Each registered business typically has one or more services associated with it. Services are listed as subentries under the business, and are accessible by double-clicking on the business entry. Similarly, service bindings are listed under the service, and are accessible by double-clicking on the service entry. To collapse an entry's subentries, double-click on it again:



When you select any entry, its details display in the right-hand View Organization panel. For example, when you select a service binding, the following details display, including the URL of the service's WSDL:



After listing a Web service's details, you can perform several tasks, including:

- Select a service URL to view the WSDL code. The WSDL displays in a browser window.
- Copy the Web service's URL for use in other tools such as the Web Services Test Client (see page 17).

Updating a Registry

Overview

You can update the data in any Web services registry that is accessible to you. The local IONA registry is always accessible. Third-party registries are accessible only to users with login privileges.

You can perform these tasks:

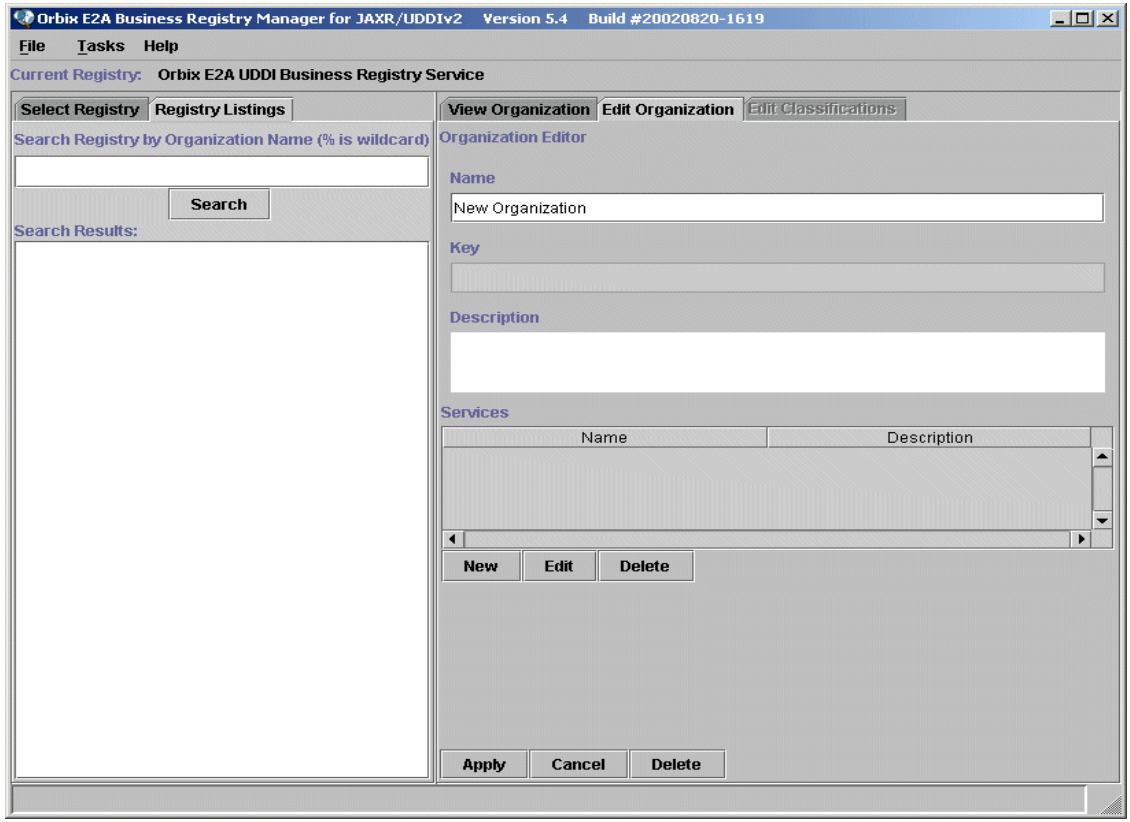
- Add new entries.
- Edit registry data.
- Delete entities.

Add new entries

If a registry is available for publishing and updating services, you can add a new business service data as follows:

1. Select **Tasks | New Organization**.

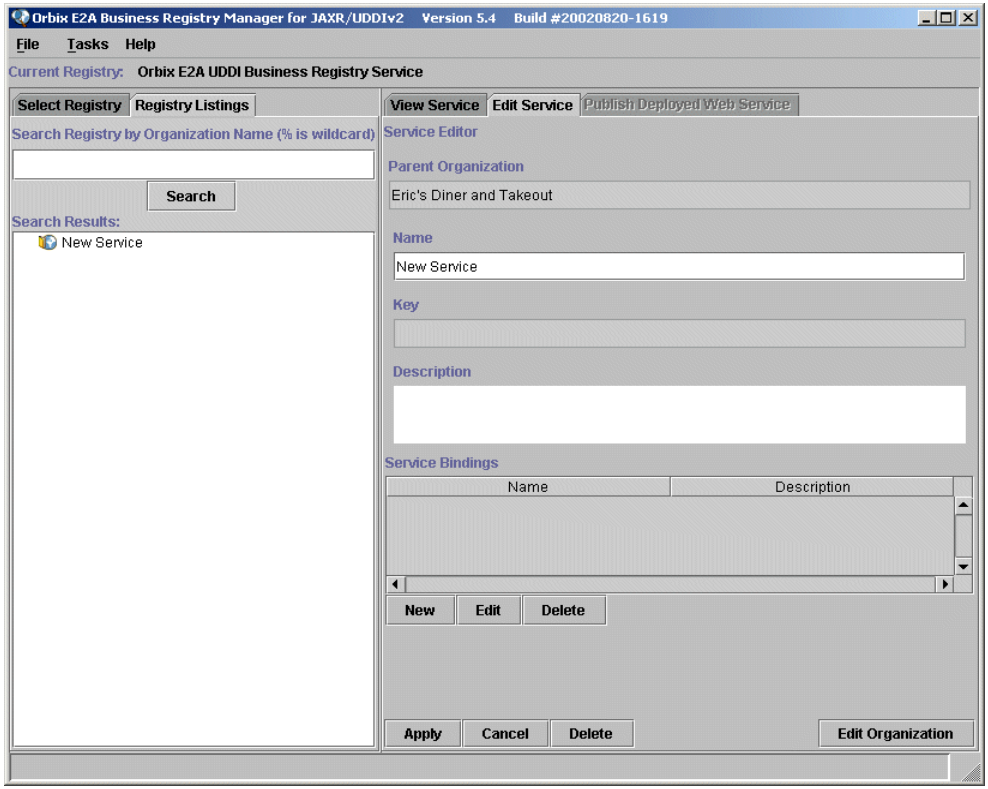
The Edit Organization dialog is displayed:



2. Enter the business data.
3. Define a service for this business entity by choosing **New**.

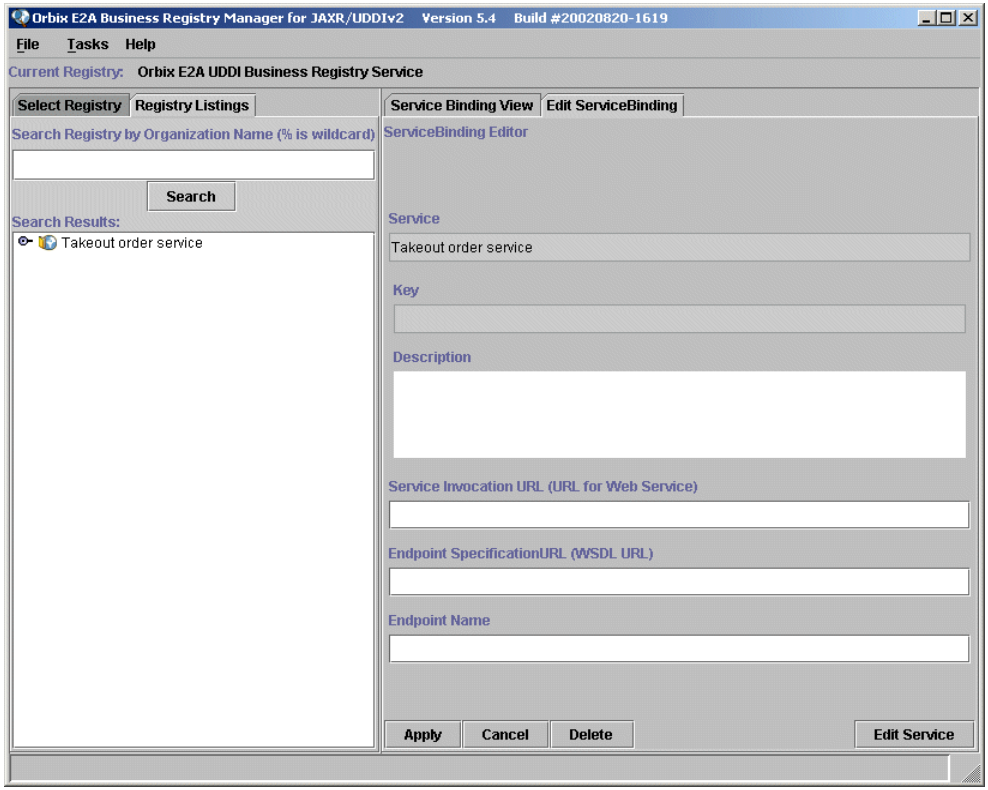
> **Note:** Choosing New implicitly accepts the current data. Apply saves the current data; Cancel removes all changes and restores the data last applied to this entity.

The Edit Service dialog is displayed:



4. Enter the service data.
5. Define a service binding for this service entity by choosing **New**.

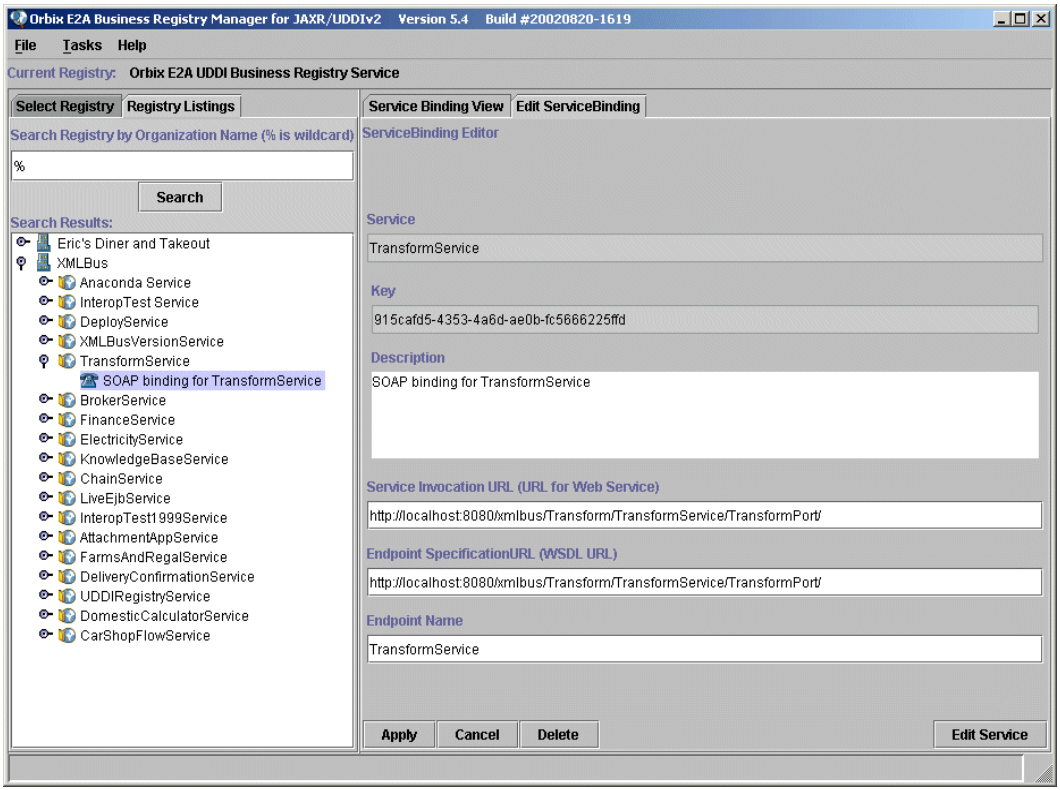
The Edit ServiceBinding dialog is displayed:



6. Enter the service binding data.

Edit registry data

You can edit the data of any registry entity by selecting that entity and choosing its Edit tab. For example, the following Edit ServiceBinding dialog lets you access the editable data for TransformService's service binding:



Delete entities

To delete an entity:

1. Select the entity.
2. Choose the entity's Edit tab.
3. From the entity's Edit dialog, click **Delete**.

Implementing Registry Clients

Overview

If you implement your own registry client, you can enable it to query and update an Orbix registry in the following ways:

- Use Orbix Web service APIs.
- Use supported third-party APIs. The Orbix registry currently supports IBM's UDDI for Java (UDDI4J), and Sun Microsystem's Java API for XML Registries Reference Implementation (JAXR RI).

In this section

This section contains information about using three sets of APIs that enable access to an Orbix registry:

Using Orbix Web Service APIs

Overview

A set of client proxy classes in `com.iona.uddi.v2.informodel` is generated from WSDL. You can use these proxy classes to interact with any UDDI registry.

Two classes are especially important:

- `com.iona.uddi.v2.informodel.InquiryInterface` is a proxy to query UDDI registry services. Its methods map to UDDI inquiry API specification.
- `com.iona.uddi.v2.informodel.PublishingInterface` is a proxy to update a UDDI registry. Its methods map to UDDI publishing API specification.

For more information about these classes, refer to the distribution's JavaDoc descriptions.

Software requirements

You must have JDK 1.3.1_02 or higher. Also, the class path must include the following jars, located off the installation directory:

```
asp/Version/lib/webservices/jaxm-api.jar
asp/Version/lib/webservices/soap_client.jar
asp/Version/lib/webservices/workbench.jar
asp/Version/lib/webservices/xerces.jar
lib/apache/jakarta-log4j/1.2.6/log4j.jar
lib/common/ifc/1.1/ifc.jar
lib/sun/mail/1.2/mail.jar
lib/xmlbus/workbench/5.4.1/ionaworkbench.jar
lib/xmlbus/jaxm/5.4.1/it_jaxm.jar
lib/xmlbus/registry_tool/5.4.1/uddistub.jar
```

Querying an IONA Web services registry

In order to query an IONA Web services registry, follow these steps:

1. Call `getProxy()` on a WebServiceProxy object to bind the Inquiry interface. For example:

```
import com.iona.uddi.v2.infomodel.*;
//...
public InquiryInterface getInquiryProxy(
   String wsdlPath, String url, boolean debug)
   throws Exception {
      try {
         Object proxy =
         WebServiceProxy.getProxy(
            "UDDIRegistryService",
            "InquiryPort",
            InquiryInterface.class,
            wsdlPath,
            debug,
            url,
            null);

            MessageSettings msettings =
               WebServiceProxy.getMessageSettings(proxy);
            msettings.setAddXSIType(false);
            msettings.setUseDefaultNamespaces(true);
            msettings.setSoapEnvelopePrefix("soap");
            return (InquiryInterface) proxy;
      } catch (Exception ex) {
         throw ex;
         }
      }
//...
```

2. Call `InquiryInterface` operations to query the UDDI registry. For example:

```
String names[] = null;
find_business fb = new find_business();
name uddiname = new name();
uddiname._simpleTypeValue = "X%";
fb.setname(new name[]{uddiname});
fb.generic = "2.0";
fb.maxRows = new Integer(50);
businessList bl = null;
String inquiryURL="http://localhost:8080/xmlbus/
   UDDIRegistry/UDDIRegistryService/InquiryPort/";
boolean debug=true;
```

```
try {
   InquiryInterface inquiryProxy =
      getInquiryProxy(inquiryURL,null,debug);
   bl = inquiryProxy.find_business(fb);
   client.printBusinessList(bl);
} catch (Exception e) {
   e.printStackTrace();
}
```

Updating an IONA registry

In order to update an IONA registry, follow these steps:

1. Call `getProxy()` on the WebServiceProxy object in order to bind the `Publish` interface. For example:

```
public PublishingInterface getPublishProxy(String wsdlPath,
   String url, boolean debug) throws Exception {
       try {
          Object proxy = WebServiceProxy.getProxy(
             "UDDIRegistryService",
             "PublishingPort",
             PublishingInterface.class,
             wsdlPath,
             debug,
             url,
             null);

            MessageSettings msettings =
               WebServiceProxy.getMessageSettings(proxy);
            msettings.setAddXSIType(false);
            msettings.setUseDefaultNamespaces(true);
            msettings.setSoapEnvelopePrefix("soap");

            return (PublishingInterface) proxy;
        } catch (Exception ex) {
            throw ex;
        }
    }
```

2. Obtain authorization from the registry. For example:

```
get_authToken getAuthToken = new get_authToken();
getAuthToken.userID = "admin";
getAuthToken.cred = "admin";

authToken token =
   publishingInterface.get_authToken(getAuthToken);

String authInfo = token.getauthInfo();
```

3. Call operations on the `PublishingInterface` object in order to update the registry. For example:

```
PublishingInterface proxy =
getPublishProxy(

   "http://localhost:8080/xmlbus/UDDIRegistry/UDDIRegistryS
   ervice/InquiryPort/",
   null, true);

//...
businessEntity be = new businessEntity();
be.businessKey = "";
/
name uddiname = new name();
uddiname._simpleTypeValue = "business entity name";
be.setname(new name[]{uddiname});

description desc = new description();
desc._simpleTypeValue = " Business Entity description";
be.setdescription(new description[]{desc});

businessEntity[]   businessEntities = new businessEntity[1];
businessEntities [0] = be;

// ...
save_business sb = new save_business();
sb.generic = "2.0";
sb.setauthInfo(token);
if (businessEntities != null && businessEntities.length > 0)
   {
      sb.setbusinessEntity(businessEntities);
   }
proxy.save_business(sb);
```

Using UDDI4J APIs

Overview

UDDI4J is an open-source Java implementation for interacting with a UDDI registry, originally developed by IBM. The latest code is now at the developerWorks open source site http://www-136.ibm.com/developerworks/opensource.

Software requirements

The following requirements apply:

- UDDI4J version 2. The uddi4j.jar is available at http://www-124.ibm.com/developerworks/oss/uddi4j/.
- Apache SOAP jar, available at http://xml.apache.org/soap/
- JDK 1.3.1_02+ with JSSE

Steps

The following describes the basic steps required in order to enable a UDDI4j client to query and update an Orbix registry:

1. Configure the `org.uddi4j.client.UDDIProxy` class to connect to an Orbix Web services registry.

```
String ionaInquiryURL =
   "http://localhost:8080/xmlbus/UDDIRegistry/UDDIRegistryS
   ervice/InquiryPort/";
String ionaPublishURL =
   "http://localhost:8080/xmlbus/UDDIRegistry/UDDIRegistryS
   ervice/PublishingPort/";

UDDIProxy proxy = new UDDIProxy();
proxy.setInquiryURL(ionaInquiryURL);
proxy.setPublishURL(ionaPublishURL);
```

2. Configure the UDDI4j SOAP transport:

```
System.setProperty(TransportFactory.PROPERTY_NAME,
   "org.uddi4j.transport.ApacheSOAPTransport")

System.setProperty("java.protocol.handler.pkgs",
   "com.sun.net.ssl.internal.www.protocol");
java.security.Security.addProvider((java.security.Provider)
   Class.forName("com.sun.net.ssl.internal.ssl.Provider").n
   ewInstance());
```

3. Obtain authorization from the registry. For example:

```
String uid = "admin";
String pwd = "admin");
AuthToken token = uddiProxy.get_authToken(uid, pwd);
```

Using JAXR RI

Software requirements

The following requirements apply:

- Java SDK: Use version 1.3.1 or 1.4.0_01 of the J2SE SDK instead of version 1.4. If you install the Java XML Pack on version 1.3.1 or 1.4.0_01, the Java XML Pack includes a fixed version of the JSSE library.
- Java API for XML Registries (JAXR) 1.0_01: JAXR reference is included in Java Web Service Developer Pack 1.0, at http://java.sun.com/webservices/downloads/webservicespack.html.

Steps

The following describes the basic steps that a JAXR client must follow in order to query and update an Orbix registry:

1. Create a connection to the registry service: To create a connection, a client first creates a set of properties that specify the URL of each registry to access, as follows:

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
   http://localhost:8080/xmlbus/UDDIRegistry/UDDIRegistrySe
   rvice/InquiryPort/");
props.setProperty("javax.xml.registry.lifeCycleManagerURL",
   "http://localhost:8080/xmlbus/UDDIRegistry/UDDIRegistryS
   ervice/PublishingPort/");
props.setProperty("javax.xml.registry.uddi.maxRows",
   "100");
props.setProperty("javax.xml.registry.ConnectionFactoryClas
   s", "com.sun.xml.registry.uddi.ConnectionFactoryImpl");
ConnectionFactory connectionFactory =
   ConnectionFactory.newInstance();
connectionFactory.setProperties(props);
connection = connectionFactory.createConnection();
```

2. Specify the HTTP proxy host and port. If the registry to access is inside a firewall, the client must also specify proxy host and port information for the network on which it is running:

```
props.setProperty("com.sun.xml.registry.http.proxyHost",
   "host.domain"");
props.setProperty("com.sun.xml.registry.http.proxyHost",
   "hostPort");

props.setProperty("com.sun.xml.registry.https.proxyHost",
   "host.domain");
props.setProperty("com.sun.xml.registry.https.proxyPort",
   "hostPort");
```

3. Specify to use Apache SOAP:

```
props.put("com.sun.xml.registry.useSOAP", "true");
```

4. Obtain authorization from the registry. For example:

```
String username="admin";
String password="admin";
PasswordAuthentication passwdAuth = new
   PasswordAuthentication(username,
   password.toCharArray());
Set creds = new HashSet();
creds.add(passwdAuth);
connection.setCredentials(creds);
```

Known issues

The name element in business, service and binding is null. The current release of the JAXR reference implementation requires `xml:lang` attribute in the name element. JAXR RI converts the element to `null` if the language attribute is not included.

Support for UDDI APIs

Overview

IONA's Web services registry supports all programming APIs in UDDI version 2 except assertion and advanced search. The following tables show which operations are implemented (Y), partially implemented (P), and not implemented (N):

Table 2: *Inquiry operations*

Operation	Status
`find_binding()`	**Y**
`find_business()`	Y
`find_relatedBusinesses()`	N
`find_service()`	Y
`findTModel()`	Y
`get_bindingDtail()`	Y
`get_businessDetail()`	Y
`get_businessDetailExt()`	N
`get_serviceDetail()`	Y
`get_tModelDetail()`	Y

Table 3: *Publishing operations*

Operation	Status
`add_publisherAssertions()`	**N**
`delete_binding()`	Y
`delete_business()`	Y
`delete_publisherAssertions()`	N
`delete_service()`	Y

Table 3: *Publishing operations*

Operation	Status
`delete_tModel()`	Y
`discard_authToken()`	Y
`get_assertionStatusReport()`	N
`get_authToken()`	Y
`get_publisherAssertions()`	N
`get_registeredInfo()`	N
`save_binding()`	Y
`save_business()`	Y
`save_service()`	Y
`save_tModel()`	Y
`set_assertions()`	N

Current Limitations

The following limitations currently apply:

- Orbix supports version 2 UDDI registries but not version 2 errata.
- Orbix registry does not support the `assertion` interface.

CHAPTER 6

Command-line Tools

Command-line tools let you perform many of the operations available in Web Service Builder. You can perform the following tasks:

- Create and modify XAR files
- Deploy Web services
- Generate code from WSDL

CLASSPATH requirements

To use command-line tools, your CLASSPATH must include the following files:

- `asp/`*`Version`*`/lib/webservices/workbench.jar`
- `asp/`*`Version`*`/lib/webservices/soap_client.jar`

User tools API

A user-tools API is also available for writing your own applications to perform most of these same tasks. See the `com.iona.webservices.usertools` package in the *IONA Web Services API Reference*.

Creating and Modifying XARs

The following command-line tools are available for creating and modifying XARs:

- `xmlbus.AddResourcesToXAR`
- `xmlbus.CORBAToXAR`
- `xmlbus.JavaToXAR`
- `xmlbus.SchemaToXAR`
- `xmlbus.TransformToXAR`
- `xmlbus.XMLToXAR`

xmlbus.AddResourcesToXAR

```
java xmlbus.AddResourcesToXAR -x XAR-name
    [-r resource_by_reference]...[-i include_resource]...
    [-s Web-service-name] [-v] [-license] [-?]
```

Description

Adds resources to an existing XAR file. Types of resources can be any resources needed by the XAR including:

- class implementation files
- interface files
- other resources

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-i include_resource`	Embed the specified resource into the XAR. You can use this option multiple times in a command.
`-license`	Display the current license.
`-r resource_by_reference`	Include a reference to the resource into the XAR. You can use this option multiple times in a command.
`-s Web-service-name`	The name of the Web service in the XAR to which the resources are added.
`-v`	Display the version of the command.
`-x XAR-name`	The XAR file to which the resources are added. **Note:** This parameter is required.

See also

`xmlbus.JavaToXAR`
`xmlbus.CORBAToXAR`
`xmlbus.XMLToXAR`

xmlbus.CORBAToXAR

```
java xmlbus.CORBAToXAR -x XAR-name
    { -ior object-reference | -targetName target-name  | -bind...}
    -repID repository-ID
    [-oc ORB-class-name] [-os ORB-singleton-class]
    [-ORB argument-name argument-value]...
    [-w WSDL-file-name] [-u base-URL] [-a SOAP-action]
    [-n namespace] [-d data-namespace] [-l application-name]
    [-s Web-service-name] [-e endpoint-name]
    [-doc] [-literal] [-override] [-1999]
    [-v] [-license] [-?]
```

Description

Creates a XAR file from a CORBA object.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-1999`	Use the 1999 XML Schema specification. The default is to use most recent schema specification supported.
`-a SOAP-action`	The SOAP action, if any, for the Web service implementation.
`-bind ...`	The bind semantics of Orbix servers used to build the object reference of the CORBA object. **Note:** This parameter is required unless the `-targetName` or `-ior` parameter is used. For more details, see page 59.
`-d data-namespace`	The namespace to use for the Web service's data.
`-doc`	Set the XARs methods to use the XML document style of interaction. The default interaction style is to use the RPC style of interaction.
`-e endpoint-name`	The endpoint name to use for the Web service in the generated XAR WSDL.
`-ior object-reference`	The object reference for the CORBA object. **Note:** This parameter is required unless the `-targetName` or `-bind` parameter is used.
`-l application-name`	The name to use for the Web service application being generated.

`-license`	Display the current license.
`-literal`	Set the XARs methods to use literal encoding. The default is to use SOAP encoding.
`-n` *`namespace`*	The namespace to use in the generated XAR WSDL that uniquely identifies the Web service's properties.
`-oc` *`ORB-class-name`*	The object request broker (ORB) class name of the CORBA resource that represents the implemented Web service. This option is required.
`-ORB` *`argument-name argument-value...`*	The name-value pair arguments that are to be passed to the initializing ORB. These are the `ORBinit()` parameters.
`-os` *`ORB-singleton-class`*	The object request broker (ORB) singleton class for the CORBA resource. This option is required.
`-override`	Override the existing XAR file information in *`XAR-name`*. The default is to add the information to the existing XAR.
`-repID` *`repository-ID`*	The identifier of the interface repository. **Note:** This parameter is required.
`-s` *`Web-service-name`*	The name of the Web service for the XAR being created.
`-targetName` *`target-name`*	The target name of the CORBA object. **Note:** This parameter is required unless the `-ior` or `-bind` parameter is used.
`-u` *`base-URL`*	The URL of the Web Service Container.
`-v`	Display the version of the command.
`-w` *`WSDL-file-name`*	The name for the Web service's WSDL file generated.
`-x` *`XAR-name`*	The XAR file that specifies the Web service. **Note:** This parameter is required.

-bind option details

The `-bind` option specifies the bind semantics of Orbix servers, which is used to build the object reference of the CORBA object. This option is required unless the `-targetName` or `-ior` parameter is used.

The semantics are as follows:

```
java xmlbus.CORBAToXAR ...
-bind servername:server-name interface:fully-scoped-interface
    [host:hostname] [port:server-listening-port]
    [marker:object-marker] [iiopVersion:IIOP-version]
...
```

servername:*server-name* The name of the server on which the CORBA object resides.

interface:*fully-scoped-interface* The CORBA object's interface name. The following formats are valid:

- Orbix-style format: *`PragmaPrefix-Module-Interface`*
- C++ style scoping: *`PragmaPrefix`*`::`*`Module`*`::`*`Interface`*
- TYPE ID notation: *`PragmaPrefix/Module/Interface`*

host:*hostname* The name of the host on which the server is running. The default value is `localhost`.

port:*server-listening-port* The port on which the server is listening, or the Orbix daemon port. If not specified, the default is `1570`, the well-known Orbix daemon port.

marker:*object-marker* The CORBA object's marker. The default value is an empty marker ("").

iiopVersion:IIOP-version The IIOP version supported by the server. Valid values are either `1.0` (default) or `1.1`.

See also

`xmlbus.AddResourcesToXAR`
`xmlbus.JavaToXAR`
`xmlbus.XMLToXAR`

xmlbus.JavaToXAR

```
java xmlbus.JavaToXAR -x XAR-name
    [-j Java-name]...
    [-w WSDL-file-name] [-u base-URL] [-a SOAP-action]
    [-n namespace] [-d data-namespace] [-l application-name]
    [-s Web-service-name] [-e endpoint-name]
    [-doc] [-literal] [-override] [-1999]
    [-v] [-license] [-?]
```

Description

Creates a XAR file from a Java class.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-1999`	Use the 1999 XML Schema specification. The default is to use most recent schema specification supported.
`-a SOAP-action`	The SOAP action, if any, for the Web service implementation.
`-c class-name`	The name of the Java target class in the JAR that represents the implemented Web service. **Note:** This parameter is required.
`-d data-namespace`	The namespace to use for the Web service's data.
`-doc`	Set the XARs methods to use the XML document style of interaction. The default interaction style is to use the RPC style of interaction.
`-e endpoint-name`	The endpoint name to use for the Web service in the generated XAR WSDL.
`-j Java-name...`	Java classes referenced by the main implementation class. More than one of this option is allowed.
`-l application-name`	The name to use for the Web service application being generated.
`-license`	Display the current license.
`-literal`	Set the XARs methods to use literal encoding. The default is to use SOAP encoding.

`-n` *`namespace`*	The namespace to use in the generated XAR WSDL that uniquely identifies the Web service's properties.
`-override`	Override the existing XAR file information in *`XAR-name`*. The default is to add the information to the existing XAR.
`-s` *`Web-service-name`*	The name of the Web service for the XAR being created.
`-u` *`base-URL`*	The URL of the Web Service Container.
`-v`	Display the version of the command.
`-w` *`WSDL-file-name`*	The name for the Web service's WSDL file generated.
`-x` *`XAR-name`*	The XAR file that specifies the Web service. **Note:** This parameter is required.

See also

`xmlbus.AddResourcesToXAR`
`xmlbus.CORBAToXAR`
`xmlbus.XMLToXAR`

xmlbus.SchemaToXAR

```
java xmlbus.xmlbus.SchemaToXAR -x XAR-name
    -sc schema-name -dh dom-handler
    {-pt part-name::uri::element-name::{in|out|inout} }...
    [-op operation-name]
    [-w WSDL-file-name] [-u base-URL] [-a SOAP-action]
    [-n namespace] [-d data-namespace] [-l application-name]
    [-s Web-service-name] [-e endpoint-name]
    [-override]
    [-v] [-license] [-?]
```

Description

Creates a Web service from a schema and DOM handler. The specified schema determines the structure of the XML document to be processed, while the DOM handler class provides the mechanism that processes the document data.

Generates a schema-based Web service.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-a SOAP-action`	The SOAP action, if any, for the Web service implementation.
`-d data-namespace`	The namespace to use for the Web service's data.
`-dh dom-handler`	The DOM handler class.
`-e endpoint-name`	The endpoint name to use for the Web service in the generated XAR WSDL.
`-l application-name`	The name to use for the Web service application being generated.
`-license`	Display the current license.
`-n namespace`	The namespace to use in the generated XAR WSDL that uniquely identifies the Web service's properties.
`-op operation-name`	The name of the operation to call when invoking on this service
`-override`	Override the existing XAR file information in *XAR-name*. The default is to add the information to the existing XAR.

`-pt` *`part-name`*`::`*`uri`*`::`*`element-name`*`::{in|out|inout}`	Specifies a root element in the service's input or output XML documents, where *`uri`* is a schema namespace for *`element-name`*, and *`element-name`* is a schema root element.
`-s` *`Web-service-name`*	The name of the Web service for the XAR being created.
`-sc` *`schema-name`*	The schema for documents received by this service.
`-u` *`base-URL`*	The URL of the Web service container.
`-v`	Display the version of the command.
`-w` *`file-name`*	Generate this Web service's WSDL and output it to *`file-name`*.
`-x` *`XAR-name`*	The XAR file that specifies the Web service.

After creating the Web service from the mapping specification, use `xmlbus.AddResourcesToXAR` to add all dependent resources to the XAR including:

- The DOM handler
- Schema files
- Java resources

xmlbus.TransformToXAR

```
java xmlbus.TransformToXAR -m mapping-specification -x XAR-name
    [-w WSDL-file-name] [-u base-URL] [-a SOAP-action]
    [-n namespace] [-d data-namespace] [-l application-name]
    [-s Web-service-name] [-e endpoint-name]
    [-override] [-v] [-license] [-?]
```

Description

Creates a Web service from a schema map. A schema map is a specification of how to transform information from one or more source XML documents to a target XML document.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-a SOAP-action`	The SOAP action, if any, for the Web service implementation.
`-d data-namespace`	The namespace to use for the Web service's data.
`-e endpoint-name`	The endpoint name to use for the Web service in the generated XAR WSDL.
`-l application-name`	The name to use for the Web service application being generated.
`-license`	Display the current license.
`-m mapping-specification`	The schema mapping specification to use. **Note:** This parameter is required.
`-n namespace`	The namespace to use in the generated XAR WSDL that uniquely identifies the Web service's properties.
`-override`	Override the existing XAR file information in *XAR-name*. The default is to add the information to the existing XAR.
`-s Web-service-name`	The name of the Web service for the XAR being created.
`-u base-URL`	The URL of the Web Service Container.
`-v`	Display the version of the command.
`-w WSDL-file-name`	The name for the Web service's WSDL file generated.

`-x` *`XAR-name`*	The XAR file that specifies the Web service. **Note:** This parameter is required.

After creating the Web service from the mapping specification, use `xmlbus.AddResourcesToXAR` to add all dependent resources to the XAR including:

- The mapping specification file used with this command
- Schema files
- Java resources

See also `xmlbus.AddResourcesToXAR`.

xmlbus.XMLToXAR

```
java xmlbus.XMLToXAR -x XAR-name  -f project-file-name
    [-r] [-v] [-license] [-?]
    script.xml
```

Description

Builds or modifies a XAR based on a "script" of properties in an XML file. For more on properties, see the Web Services Programmer's Reference.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-f project-file-name`	Web Service Builder project file name. **Note:** This parameter is required.
`-license`	Display the current license.
`-r`	Force a complete rebuild of the XAR.
`-v`	Display the version of the command.
`-x XAR-name`	The XAR file that is modified. **Note:** This parameter is required.

See also

`xmlbus.AddResourcesToXAR`
`xmlbus.JavaToXAR`
`xmlbus.CORBAToXAR`

Deploying Web Services

The following command-line tools are available for deploying and undeploying Web services:

- `xmlbus.Deploy`
- `xmlbus.Undeploy`

xmlbus.Deploy

```
java xmlbus.Deploy -x XAR-name -u deployment-service-URL
    [-debug] [-username username  -password password]
    [-v] [-license] [-?]
```

Description

Deploys a Web service application to the Web services container.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-debug`	Produce debugging information. The default is no debugging.
`-license`	Display the current license.
`-password` *`password`*	A password to use if the Web service has been made secure. If this is needed, the `-username` option is also required.
`-u` *`deployment-service-URL`*	The URL representing the deployment service application, the XMLBus service that performs the actual deployment. **Note:** This parameter is required. For example: `http://localhost:9000/xmlbus/Deploy/DeployService/DeployPort` Be sure to set the host and port number to match your installation.
`-username` *`username`*	A user name to use if the Web service has been made secure. If this is needed, the `-password` option is also required.
`-v`	Display the version of the command.
`-x` *`XAR-name`*	The XAR file that specifies the Web service to deploy to the Web services container. **Note:** This parameter is required.

See also

`xmlbus.Undeploy`

xmlbus.Undeploy

```
java xmlbus.Undeploy { -app application-name | -x XAR-name }
    -u deployment-service-URL
    [-debug] [-username username  -password password]
    [-v] [-license] [-?]
```

Description

Undeploys a Web service application from the Web services container.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-app` *`application-name`*	The name of the Web service application within the XAR file. **Note:** This option or the `-x` option is required.
`-debug`	Produce debugging information. The default is no debugging.
`-license`	Display the current license.
`-password` *`password`*	A password to use if the Web service has been made secure. If this is required, the `-username` option is also required.
`-u` *`deployment-service-URL`*	The URL representing the deployment service application, the XMLBus service that performs the actual undeployment. **Note:** This parameter is required.
`-username` *`username`*	A user name to use if the Web service has been made secure. If this is required, the `-password` option is also required.
`-v`	Display the version of the command.
`-x` *`XAR-name`*	The XAR file that specifies the Web services to undeploy from the Web services container. **Note:** This or the `-app` option is required.

Generating Code from WSDL

The following command-line tools are available for generating code from WSDL:

- `xmlbus.WSDLToInterface`
- `xmlbus.WSDLToJ2MEClient`
- `xmlbus.WSDLToJ2SEDemo`
- `xmlbus.WSDLToSkeleton`

xmlbus.WSDLToInterface

```
java xmlbus.WSDLToInterface { -w WSDL-URL | -x XAR-file }
    [-e Web-service-name] [-t port] [-b binding-name]
    [-d output-directory] [-p Java-package]
    [-v] [-license] [-?]
```

Description

Generates a Java interface class and proxy code from the Web service WSDL input. The interface can then be called from custom client code to access the Web service via the generated proxy.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-b` *`binding-name`*	Specify the binding to use if the service and port are not specified and there is more than one binding.
`-d` *`output-directory`*	The directory in which the code is placed. The default is to use the current directory, if this option is not specified.
`-e` *`Web-service-name`*	The name of the Web service in the WSDL file being processed. The default is to use the first Web service in the WSDL file, if this option is not specified.
`-license`	Display the current license.
`-p` *`Java-package`*	The Java package name to use for the generated code.
`-t` *`port`*	The WSDL port name or endpoint name. This port, combined with the Web service name identifies the Web service. The default is to use the port of the first Web service, if this option is not specified.
`-v`	Display the version of the command.
`-w` *`WSDL-URL`*	A URL of the WSDL file to use. **Note:** This or the `-x` option is required.
`-x` *`XAR-name`*	A XAR file that contains the Web service's WSDL specification. **Note:** This or the `-w` option is required.

See also

`xmlbus.WSDLToJ2MEClient`
`xmlbus.WSDLToJ2SEDemo`
`xmlbus.WSDLToSkeleton`

xmlbus.WSDLToJ2MEClient

```
java xmlbus.WSDLToJ2MEClient { -w WSDL-URL | -x XAR-file }
    [-e Web-service-name] [-t port] [-b binding-name]
    [-d output-directory] [-p Java-package]
    [-v] [-license] [-?]
```

Description

Generates J2ME client code from the Web service WSDL input.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-b` *`binding-name`*	Specify the binding to use if the service and port are not specified and there is more than one binding.
`-d` *`output-directory`*	The directory in which the code is placed. The default is to use the current directory, if this option is not specified.
`-e` *`Web-service-name`*	The name of the Web service in the WSDL file being processed. The default is to use the first Web service in the WSDL file, if this option is not specified.
`-license`	Display the current license.
`-p` *`Java-package`*	The Java package name to use for the generated code.
`-t` *`port`*	The WSDL port name or endpoint name. This port, combined with the Web service name identifies the Web service. The default is to use the port of the first Web service, if this option is not specified.
`-v`	Display the version of the command.
`-w` *`WSDL-URL`*	A URL of the WSDL file to use. **Note:** This or the `-x` option is required.
`-x` *`XAR-name`*	A XAR file that contains the Web service's WSDL specification. **Note:** This or the `-w` option is required.

See also

`xmlbus.WSDLToJ2SEDemo`
`xmlbus.WSDLToSkeleton`

xmlbus.WSDLToJ2SEDemo

```
java xmlbus.WSDLToJ2SEDemo { -w WSDL-URL | -x XAR-file }
    [-e Web-service-name] [-t port] [-b binding-name]
    [-d output-directory] [-p Java-package]
    [-v] [-license] [-?]
```

Description

Generates a Java client application for the Web service described by the WSDL.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-b` *`binding-name`*	Specify the binding to use if the service and port are not specified and there is more than one binding.
`-d` *`output-directory`*	The directory in which the code is placed. The default is to use the current directory, if this option is not specified.
`-e` *`Web-service-name`*	The name of the Web service in the WSDL file being processed. The default is to use the first Web service in the WSDL file, if this option is not specified.
`-license`	Display the current license.
`-p` *`Java-package`*	The Java package name to use for the generated code.
`-t` *`port`*	The WSDL port name or endpoint name. This port, combined with the Web service name identifies the Web service. The default is to use the port of the first Web service, if this option is not specified.
`-v`	Display the version of the command.
`-w` *`WSDL-URL`*	A URL of the WSDL file to use. **Note:** This or the `-x` option is required.
`-x` *`XAR-name`*	A XAR file that contains the Web service's WSDL specification. **Note:** This or the `-w` option is required.

See also

`xmlbus.WSDLToJ2MEClient`
`xmlbus.WSDLToSkeleton`

xmlbus.WSDLToSkeleton

```
java xmlbus.WSDLToSkeleton { -w WSDL-URL | -x XAR-file }
    [-e Web-service-name] [-t port] [-b binding-name]
    [-d output-directory] [-delagate] [-p Java-package]
    [-v] [-license] [-?]
```

Description

Generates Java skeleton code from the Web service WSDL input, to be used as a basis to write a Web service Implementation.

Options

The following options are available:

`-?`	Display the tool's usage and options.
`-binding` *`binding-name`*	Specify the binding to use if the service and port are not specified and there is more than one binding.
`-d` *`output-directory`*	The directory in which the code is placed. The default is to use the current directory, if this option is not specified.
`-delagate`	Delegates all calls in the skeleton to an object that implements the Web service's interface.
`-e` *`Web-service-name`*	The name of the Web service in the WSDL file being processed. The default is to use the first Web service in the WSDL file, if this option is not specified.
`-license`	Display the current license.
`-p` *`Java-package`*	The Java package name to use for the generated code.
`-t` *`port`*	The WSDL port name or endpoint name. This port, combined with the Web service name identifies the Web service. The default is to use the port of the first Web service, if this option is not specified.
`-v`	Display the version of the command.
`-w` *`WSDL-URL`*	A URL of the WSDL file to use. **Note:** This or the `-x` option is required.
`-x` *`XAR-name`*	A XAR file that contains the Web service's WSDL specification. **Note:** This or the `-w` option is required.

See also

`xmlbus.WSDLToJ2MEClient`
`xmlbus.WSDLToJ2SEDemo`

Index