



Building Web Services

Sun™ ONE Studio 4 Programming Series

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-7862-10
September 2002, Revision A

Send comments about this document to: docfeedback@sun.com

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

This product includes code licensed from RSA Data Security.

Sun, Sun Microsystems, the Sun logo, Forte, Java, NetBeans, iPlanet, docs.sun.com, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Netscape and Netscape Navigator are trademarks or registered trademarks of Netscape Communications Corporation in the United States and other countries.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuelle relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Ce produit comprend le logiciel licencié par RSA Data Security.

Sun, Sun Microsystems, le logo Sun, Forte, Java, NetBeans, iPlanet, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Netscape et Netscape Navigator sont des marques de Netscape Communications Corporation aux Etats-Unis et dans d'autres pays.

LA DOCUMENTATION EST FOURNIE “EN L'ÉTAT” ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Adobe PostScript

Contents

Before You Begin xiii

1. Web Services: A Conceptual Overview 1

What Are Web Services? 1

Industry Problems 1

Applications Within an Enterprise 2

Applications Shared Across Enterprises 2

High-Performance Utility Programs 3

A Standardized Solution 3

Web Services Standards 4

SOAP 4

WSDL 5

UDDI 6

Publishing and Using a Web Service 7

Sun ONE Studio 4 Java Web Services 9

Multitier Architecture 10

Web-centric Architecture 11

Software Versions 12

XML Operations 13

What Is an XML Operation? 14

| | |
|---|-----------|
| Request-Response Mechanism | 15 |
| 2. Building a Web Service | 17 |
| Web Service Development Tasks | 17 |
| Creating a JAX-RPC Web Service from Java Methods | 18 |
| Developing XML Operations | 20 |
| Adding References to Methods, XML Operations, and JAR Files | 20 |
| Deleting References From a Web Service | 22 |
| Creating a JAX-RPC Web Service from WSDL | 22 |
| Resolving References to Runtime Objects | 26 |
| Adding Environment Entries | 26 |
| Generating Runtime Classes | 28 |
| Assembling and Deploying a Web Service | 29 |
| A Web-Centric Application | 29 |
| Assembling the J2EE Application | 30 |
| Configuring the RI Server | 32 |
| Deploying the J2EE Application | 34 |
| Creating a Test Client | 34 |
| Web Service Client Features | 34 |
| Setting a Default Test Client for a Web Service | 34 |
| Testing a Web Service | 35 |
| Working With UDDI Registries | 36 |
| Generating WSDL | 37 |
| Managing UDDI Registry Options | 38 |
| Setting Default Publish Categories and Identifiers | 39 |
| Editing Registries Information in the IDE | 41 |
| Gaining Access to an External UDDI Registry | 43 |

| | |
|---|-----------|
| Publishing Your Web Service to a UDDI Registry | 43 |
| Publication Tasks and Terminology | 43 |
| Publication Procedure | 44 |
| The Internal UDDI Registry | 49 |
| Starting and Stopping the Internal UDDI Registry Server | 49 |
| Using the Sample Registry Browser | 50 |
| Instantiating Objects and Resolving References | 54 |
| Deployment Descriptors | 54 |
| Support for Arrays and Collections | 55 |
| Arrays | 55 |
| Collections | 55 |
| 3. Creating a Web Service Client | 57 |
| Creating a Client From a Local Web Service | 57 |
| Creating the Client | 57 |
| Setting the Client's SOAP Runtime Property | 59 |
| Generating the Client | 60 |
| The Client HTML Pages and JSP Pages | 61 |
| The Client SOAP Proxy | 63 |
| Using Your Own Front End Client Components | 64 |
| Assembling the Client Application | 65 |
| Deploying and Executing the Client | 65 |
| Creating a Client From WSDL | 68 |
| Creating a Client From a UDDI Registry | 69 |
| Creating a Client: Planning and Implementation | 69 |
| Creating a Client: Procedure | 69 |
| The Service Endpoint URL | 78 |

4. Developing XML Operations 81

Overview of Tools 81

The Data Source Pane 83

Input Document Elements Node 83

Methods Node 84

Executing Methods and Returning Data 84

Providing Parameter Values 85

Retrieving More or Less Data 85

Trimming the Data Returned to the Client 86

Development Work Flow 86

Creating XML Operations 87

Creating an XML Operation 87

Generating XML Operations From an Enterprise Bean 89

Editing an XML Operation 90

Adding a Method to an XML Operation 91

Adding an Input Document Element 92

Renaming an Input Document Element 94

Renaming an Output Document Element 94

Giving an Input Document Element a Default Value 95

Making an Input Document Element Permanent 96

Reordering a Method or Input Document Element 97

Deleting a Method or Input Document Element 97

Mapping a Method Parameter to a Source 98

Casting a Method Return Value 100

Displaying and Selecting Inherited Methods 100

Excluding an Element From the XML Output 100

Including an Element in the XML Output 101

| | |
|--|------------|
| Expanding a Class | 102 |
| Collapsing a Class | 102 |
| System Shared Objects | 102 |
| Static Utility Methods | 103 |
| Organizing Static Utility Methods | 103 |
| Using Static Utility Methods | 103 |
| A. Integration of C++ Shared Libraries | 107 |
| Tasks of the C++ Developer | 107 |
| Tasks of the Java Web Service Developer | 110 |
| Software Requirements | 111 |
| Customizing Your J2EE RI Installation | 111 |
| Installing the Shared Libraries in Your Application Server | 112 |
| B. Instantiating Objects and Resolving References | 113 |
| Specifying the Target of an Object Reference | 113 |
| Defining a New Target Object | 116 |
| Editing a Target Object Definition | 116 |
| C. Deployment Descriptors | 121 |
| Fields Propagated to Deployment Descriptors | 121 |
| Fields Propagated to the EJB Module Deployment Descriptor | 122 |
| Fields Propagated to the Web Module Deployment Descriptor | 123 |
| Viewing a Deployment Descriptor | 123 |
| Editing a Deployment Descriptor | 123 |
| Index | 125 |

Figures

| | | |
|-------------|---|----|
| FIGURE 1-1 | Publishing and Using a Web Service | 8 |
| FIGURE 1-2 | A Sun ONE Studio 4 Java Web Service and Client (Multitier Model) | 10 |
| FIGURE 1-3 | A Sun ONE Studio 4 Java Web Service and Client (Web-centric Model) | 11 |
| FIGURE 1-4 | XML Operation Calling Multiple Methods to Fulfill a Client Request | 16 |
| FIGURE 2-1 | New Web Service Wizard | 18 |
| FIGURE 2-2 | Web Service Properties With Context Root Set to <code>StockApp</code> | 20 |
| FIGURE 2-3 | Logical EJB Nodes in the Explorer | 21 |
| FIGURE 2-4 | Web Service References | 21 |
| FIGURE 2-5 | New Web Service Wizard | 23 |
| FIGURE 2-6 | New Web Service Wizard: Select WSDL File | 24 |
| FIGURE 2-7 | New Web Service Wizard: Specify New EJB | 25 |
| FIGURE 2-8 | Web Service and EJB Nodes Generated from WSDL: Explorer View | 26 |
| FIGURE 2-9 | Environment Entries Dialog Box | 27 |
| FIGURE 2-10 | Add Environment Entry Dialog Box | 27 |
| FIGURE 2-11 | Web Service Hierarchy of Nodes | 28 |
| FIGURE 2-12 | J2EE Application for a Web Service | 31 |
| FIGURE 2-13 | Web Service WAR File Properties With Web Context Set to <code>StockApp</code> | 31 |
| FIGURE 2-14 | Application Server Instance for Deployment | 32 |
| FIGURE 2-15 | Creating an Application Server Instance | 33 |

| | | |
|-------------|---|----|
| FIGURE 2-16 | UDDI Registries Options Dialog Box | 38 |
| FIGURE 2-17 | UDDI Publish Categories Dialog Box | 39 |
| FIGURE 2-18 | UDDI Categories (Taxonomies) | 40 |
| FIGURE 2-19 | UDDI Publish Identifiers Dialog Box | 40 |
| FIGURE 2-20 | UDDI Add Identifier Dialog Box | 41 |
| FIGURE 2-21 | UDDI Registries Property Editor | 42 |
| FIGURE 2-22 | UDDI Publish New Web Service Dialog Box | 45 |
| FIGURE 2-23 | UDDI Login and Business Information Dialog Box | 46 |
| FIGURE 2-24 | UDDI tModel Selection Dialog Box | 47 |
| FIGURE 2-25 | UDDI Set Categories and Identifiers | 48 |
| FIGURE 2-26 | Start Internal UDDI Registry Server | 49 |
| FIGURE 2-27 | JWSDP Sample Registry Browser | 51 |
| FIGURE 2-28 | JWSDP Sample Registry Browser URL Selection | 51 |
| FIGURE 2-29 | JWSDP Sample Registry Browser With Internal Registry URL | 52 |
| FIGURE 2-30 | JWSDP Sample Registry Browser Displaying Selected Business | 53 |
| FIGURE 2-31 | JWSDP Sample Registry Browser Displaying Submissions Tabbed Pane | 53 |
| FIGURE 2-32 | Serialization Classes Property Editor | 56 |
| FIGURE 3-1 | New Client From Web Service Dialog Box | 58 |
| FIGURE 3-2 | New Web Service Client Dialog Box | 59 |
| FIGURE 3-3 | Client SOAP Runtime Property | 60 |
| FIGURE 3-4 | Client Documents and GenClient Nodes in Explorer Hierarchical Display | 61 |
| FIGURE 3-5 | Client Sample HTML Welcome Page | 62 |
| FIGURE 3-6 | Client Sample JSP Page | 62 |
| FIGURE 3-7 | Client SOAP Proxy GenClient Node | 63 |
| FIGURE 3-8 | Client Sample WAR File Display | 65 |
| FIGURE 3-9 | Client Welcome Page | 66 |
| FIGURE 3-10 | Client Output Display of Collection of Customer Reviews | 67 |
| FIGURE 3-11 | New Web Service Client Wizard | 70 |

| | | |
|-------------|--|-----|
| FIGURE 3-12 | UDDI Registry Selection Dialog Box | 71 |
| FIGURE 3-13 | UDDI Registry Search Dialog Box | 72 |
| FIGURE 3-14 | UDDI Registry Search Types | 73 |
| FIGURE 3-15 | UDDI Registry Search Dialog Box With Matching Businesses | 75 |
| FIGURE 3-16 | UDDI Registry Filter Business Progress Monitor | 76 |
| FIGURE 3-17 | UDDI Registry Select Service | 77 |
| FIGURE 3-18 | UDDI Registry Display Service Details and tModel | 78 |
| FIGURE 4-1 | XML Operation Source Editor, Showing Complex Input | 82 |
| FIGURE 4-2 | XML Operation Source Editor, Showing Complex Output | 82 |
| FIGURE 4-3 | The Input Document Elements Node | 83 |
| FIGURE 4-4 | New XML Operation Dialog Box | 88 |
| FIGURE 4-5 | Select Method Dialog Box | 88 |
| FIGURE 4-6 | Collection of What? Dialog Box | 90 |
| FIGURE 4-7 | Add Method to XML Operation Dialog Box | 91 |
| FIGURE 4-8 | Select Method Dialog Box | 91 |
| FIGURE 4-9 | Add Input Document Element Dialog Box | 93 |
| FIGURE 4-10 | Input Document Element Properties Dialog Box | 94 |
| FIGURE 4-11 | Output Document Element Properties Dialog Box | 95 |
| FIGURE 4-12 | Input Document Element Properties (Default Value) | 96 |
| FIGURE 4-13 | Method Parameter Source Dialog Box | 98 |
| FIGURE 4-14 | Source Editor: (Excluding an Output Element) | 101 |
| FIGURE 4-15 | Method Parameter Source Dialog Box | 104 |
| FIGURE A-1 | Native Connector Tool Work Flow | 109 |
| FIGURE A-2 | Native Connector Node | 110 |
| FIGURE B-1 | Resolve Object References Dialog Box | 114 |
| FIGURE B-2 | Resolve Object References Dialog Box With XML Operations | 115 |
| FIGURE B-3 | Map Parameters Dialog Box | 119 |
| FIGURE C-1 | Deployment Descriptor Final Edit Dialog Box | 124 |

Before You Begin

This book explains how to build and deploy web services and web service clients using the Sun™ Open Net Environment (Sun ONE) Studio 4 integrated development environment (IDE).

The book is intended primarily for web service developers. A conceptual overview is provided that can benefit anyone seeking a general understanding of web services.

See the release notes for a list of environments in which you can create the examples in this book. The release notes are available on this web page:

<http://forte.sun.com/ffj/documentation/index.html>

Screen shots vary slightly from one platform to another. You should have no trouble translating the slight differences to your platform. Although almost all procedures use the interface of the Sun™ ONE Studio 4 software, occasionally you might be instructed to enter a command at the command line. Here too, there are slight differences from one platform to another. For example, a Microsoft Windows command might look like this:

```
c:>cd MyWorkDir\MyPackage
```

To translate for UNIX® or Linux environments, simply change the prompt and use forward slashes:

```
% cd MyWorkDir/MyPackage
```

Before You Read This Book

Before starting, you should be familiar with the following subjects:

- Java programming language
- Enterprise JavaBeans™ (EJB™) component model
- JavaServer Pages™ (JSP™) software
- HTML syntax
- J2EE application assembly and deployment concepts

This book requires a knowledge of J2EE concepts, as described in the following resources:

- Java 2 Platform, Enterprise Edition Blueprints, Version 1.3
<http://java.sun.com/blueprints>
- *Java 2 Platform, Enterprise Edition Specification*, v. 1.4
<http://java.sun.com/j2ee/download.html#platformspec>
- *The J2EE Tutorial* (SDK v. 1.3)
<http://java.sun.com/j2ee/tutorial>
- *Java Servlet Specification Version 2.3*
<http://java.sun.com/products/servlet/download.html#specs>
- *JavaServer Pages Specification Version 1.2*
<http://java.sun.com/products/jsp/download.html#specs>

Familiarity with the Java API for XML-Based RPC (JAX-RPC) is helpful. For more information, see this web page:

<http://java.sun.com/xml/jaxrpc>

The following resources provide useful background knowledge of web services standards:

- *SOAP 1.1 Specification*—
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508>
- *Apache SOAP 2.2 Implementation of the SOAP 1.1 Specification*—
<http://xml.apache.org/soap/>
- *kSOAP 1.0 (a SOAP API suitable for the Java 2 Microedition)*
<http://www.ksoap.org/>
- *WSDL 1.1 Specification*—
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- *UDDI 2.0 Specification*—
<http://www.uddi.org/specification.html>
- JAXR 1.0_01 API specification
<http://jcp.org/jsr/detail/93.jsp>

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document and does not endorse and is not responsible or liable for any content, advertising, products, or other materials on or available from such sites or resources. Sun will not be responsible or liable for any 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 available on or through any such sites or resources.

How This Book Is Organized

Chapter 1 provides an overview of web services standards and the web services features of the Sun ONE Studio 4 IDE.

Chapter 2 outlines the work flow for developing and testing a web service and explains how to use the web service development tools. It also explains how you can use a UDDI registry to make your web service available to others.

Chapter 3 explains how you create clients that can use your web service. It also explains how to search a UDDI registry for web services and create clients that can use those web services.

Chapter 4 explains how you create and edit XML operations, which are optional building blocks of a web service. It also provides a description of the tools you use for this job.

Appendix A describes how to expose native C++ library methods in your web service, using the Sun ONE Studio 4 Native Connector Tool.

Appendix B describes how to manually specify the target of an object reference in a web service.

Appendix C describes how to view and edit a deployment descriptor. It also lists the IDE fields that are propagated to the deployment descriptor of an EJB module or web service module.

Typographic Conventions

| Typeface | Meaning | Examples |
|------------------|--|---|
| AaBbCc123 | The names of commands, files, and directories; on-screen computer output | Edit your .cvspass file. Use DIR to list all files. Search is complete. |
| AaBbCc123 | What you type, when contrasted with on-screen computer output | > login Password: |
| <i>AaBbCc123</i> | Book titles, new words or terms, words to be emphasized | Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> save your changes. |
| <i>AaBbCc123</i> | Command-line variable; replace with a real name or value | To delete a file, type DEL <i>filename</i> . |

Related Documentation

Sun ONE Studio 4 documentation includes books delivered in Acrobat Reader (PDF) format, release notes, online help, readme files for example applications, and Javadoc™ documentation.

Documentation Available Online

The documents described in this section are available from the docs.sun.comSM web site and from the documentation page of the Sun ONE Studio Developer Resources portal (<http://forte.sun.com/ffj/documentation>).

The docs.sun.com web site (<http://docs.sun.com>) enables you to read, print, and buy Sun Microsystems manuals through the Internet. If you cannot find a manual, see the documentation index installed with the product on your local system or network.

- Release notes (HTML format)

Available for each Sun ONE Studio 4 edition. Describe last-minute release changes and technical notes.

- Getting Started guides (PDF format)

Describe how to install the Sun ONE Studio 4 integrated development environment (IDE) on each supported platform and include other pertinent information, such as system requirements, upgrade instructions, application server configuration instructions, command-line switches, installed subdirectories, database integration, and information on how to use the Update Center.

- *Sun ONE Studio 4, Community Edition Getting Started Guide* - part no. 816-7871-10
- *Sun ONE Studio 4, Enterprise Edition for Java Getting Started Guide* - part no. 816-7859-10
- *Sun ONE Studio 4, Mobile Edition Getting Started Guide* - part no. 816-7872-10

- Sun ONE Studio 4 Programming series (PDF format)

- This series provides in-depth information on how to use various Sun ONE Studio 4 features to develop well-formed J2EE applications. *Building Web Components* - part no. 816-7869-10

Describes how to build a web application as a J2EE web module using JSP pages, servlets, tag libraries, and supporting classes and files.

- *Building J2EE Applications* - part no. 816-7863-10

Describes how to assemble EJB modules and web modules into a J2EE application, and how to deploy and run a J2EE application.

- *Building Enterprise JavaBeans Components* - part no. 816-7864-10

Describes how to build EJB components (session beans, message-driven beans, and entity beans with container-managed or bean-managed persistence) using the Sun ONE Studio 4 EJB Builder wizard and other components of the IDE.

- *Building Web Services* - part no. 816-7862-10

Describes how to use the Sun ONE Studio 4 IDE to build web services, to make web services available to others through a UDDI registry, and to generate web service clients from a local web service or a UDDI registry.

- *Using Java DataBase Connectivity* - part no. 816-7870-10

Describes how to use the JDBC productivity enhancement tools of the Sun ONE Studio 4 IDE, including how to use them to create a JDBC application.

- Sun ONE Studio 4 tutorials (PDF format)

These tutorials demonstrate how to use the major features of each Sun ONE Studio 4 edition.

- *Sun ONE Studio 4, Community Edition Tutorial* - part no. 816-7868-10

Provides step-by-step instructions for building a simple J2EE web application.

- *Sun ONE Studio 4, Enterprise Edition for Java Tutorial* - part no. 816-7860-10

Provides step-by-step instructions for building an application using EJB components and Web Services technology.

- *Sun ONE Studio 4, Mobile Edition Tutorial* - part no. 816-7873-10

Provides step-by-step instructions for building a simple application for a wireless device, such as a cellular phone or personal digital assistant (PDA). The application will be compliant with the Java 2 Platform, Micro Edition (J2ME™ platform) and conform to the Mobile Information Device Profile (MIDP) and Connected, Limited Device Configuration (CLDC).

You can also find the completed tutorial applications at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

Online Help

Online help is available inside the Sun ONE Studio 4 IDE. You can open help by pressing the help key (F1 in Microsoft Windows and Linux environments, Help key in the Solaris environment), or by choosing Help → Contents. Either action displays a list of help topics and a search facility.

Examples

You can download examples that illustrate a particular Sun ONE Studio 4 feature, as well as completed tutorial applications, from the Sun ONE Studio Developer Resources portal at:

<http://forte.sun.com/ffj/documentation/tutorialsandexamples.html>

The site includes the applications used in this document.

Javadoc Documentation

Javadoc documentation is available within the IDE for many Sun ONE Studio 4 modules. Refer to the release notes for instructions on installing this documentation. When you start the IDE, you can access this Javadoc documentation within the Javadoc pane of the Explorer.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Email your comments to Sun at this address:

`docfeedback@sun.com`

Please include the part number (816-7862-10) of your document in the subject line of your email.

Web Services: A Conceptual Overview

This book explains how you can build simple and complex web services using the Sun™ Open Net Environment (Sun ONE) Studio 4, Enterprise Edition for Java integrated development environment (IDE). The explanations assume that you have general knowledge of how to use the IDE.

Industry standards for web services are still evolving. The software implementing the standards are evolving. Even the meaning of web services can be confusing to developers who simply want to get started using the new technologies. This chapter provides an overview of web services, and lays the groundwork for subsequent chapters that describe how to use the web services features of the IDE.

What Are Web Services?

In general terms, *web services* are distributed application components that conform to standards that make them externally available and solve certain kinds of widespread industry problems. This section describes some of the problems addressed by web services, and provides an overview of web services standards.

Industry Problems

One of the basic problems of a modern private enterprise or public agency is how to integrate diverse computer applications that have been developed independently and run on a variety of software and hardware platforms. The proliferation of distributed environments has created a need for an enterprise to be able to expose part of the functionality of an application to other applications over a network. Those functions might be used within the enterprise, by related enterprises such as business partners, and even by external enterprises that have no organizational or contractual relationship with the enterprise providing the service.

Decoupling client and service development enables a given service to be accessed by a variety of client types (such as browser-based clients and wireless clients) that adhere to the standardized web services technologies. In this context, a “client” means any user of a service, including another service.

Applications Within an Enterprise

Traditional enterprise systems often require interfaces between related suites of applications. One example is software that manages accounts payable, accounts receivable, general ledger, and departmental budgets. Another example is software for personnel data, payroll, and benefits. A payroll system typically is responsible for tax withholding, direct deposit to banks, and other functions that provide information to systems outside the enterprise.

The traditional use of programmed interfaces and shared data for integration requires coordination of the logic and data structures of the applications involved, as well as the ability to handle differences between hardware and software platforms. This approach creates maintenance problems even within an enterprise. A desired change in one application might require a series of changes in other applications. Changes in tax regulations and government-mandated reporting requirements can prove difficult and costly for enterprises and industries that rely on heterogeneous computer systems.

Applications Shared Across Enterprises

Recent years have seen enormous growth of Internet commerce, electronic transfer of funds, and other networked activities that are necessary to economic and social institutions. Production, supply, and trade is often widely distributed, with complex logistics and tracking requirements. Enterprises need to accommodate elaborate processes, regulations, and relationships with external enterprises and government entities. These developments have created a greater need for computer applications that are interoperable across platforms, capable of being shared across networks, and adaptable to change.

A simple example of a widely shared type of Internet application is an online catalog. Many web sites make browser-based interfaces available to customers for this purpose. A more complex example is a business application that orders parts of many different types from a variety of vendors. A given transaction might involve several suppliers, the decisions can depend on dynamically changing prices and dates of availability, and the items that make up a transaction must have compatible technical specifications. The desire of enterprises to be able to do business in this way is a major motivation behind the efforts of computer vendors and standards groups to develop a web services infrastructure.

High-Performance Utility Programs

Another industry requirement is for modern applications to access standardized, computationally intensive algorithms written in low-level programming languages. Examples include actuarial calculations, stock analysis formulas, weather forecasting, optimal route selections, and general-purpose statistical routines. Some of these programs are very complex, highly optimized, and widely established, so that it is not feasible to consider rewriting them. Web services make it possible to expose the functionality of these programs to new applications. (Appendix A explains how you can access C and C++ shared libraries in a web service.)

A Standardized Solution

Web services are distributed, reusable application components that selectively expose the functionality of business services such as EJB components and make that functionality available to applications through standardized Internet protocols.

The web services architecture satisfies these requirements:

- **Interoperability.** An application that uses a web service component need not be aware of the hardware and software platform on which the service runs. A web service can be accessed by different kinds of clients (such as web applications, wireless applications, and other services), as long as they use the standard web services technologies.
- **Encapsulation.** An application that uses a web service component need not be concerned with details of the component's internal programming.
- **Availability.** The developer of a web service must have a way to publish it with enough information so that other developers can find the web service and create a client application component capable of using it.

The Sun ONE Studio 4, Enterprise Edition for Java IDE meets these requirements and, additionally, gives you the ability to combine the low-level methods of one or more business components into higher-level functionality suitable for a web service.

The following section describes the standards, technologies, and architecture that define web services.

Web Services Standards

The web services architecture is based on three related standards: SOAP, WSDL, and UDDI. The standards documents are lengthy, but in general terms the three-pronged architecture can be described as follows:

- **SOAP** (Simple Object Access Protocol) defines the mechanism by which a web service is called and how data is returned. SOAP clients can call methods on SOAP services, passing objects in XML format.
- **WSDL** (Web Service Definition Language) describes the external interface of a web service so that developers can create clients capable of using the service.
- **UDDI** (Universal Discovery, Description, and Integration) registries contain information about web services, including the location of WSDL files and the location of the running services, so that a wide community of developers can create clients to access the services and incorporate their functionality into applications.

Note – Implementations of web services generally involve passing XML documents over HTTP. This book assumes that the reader is familiar with XML and HTTP.

Note – Web services terminology is evolving along with the technology. For example, the term WSDL is an acronym, but it is also used as a noun, meaning the description of a web service. The acronym for SOAP is still in use but the SOAP protocol is no longer “simple” and recent draft standards state that the name should not be considered an acronym. This book follows common industry usage and explains the meaning of terms in any places where they might be ambiguous.

SOAP

SOAP (Simple Object Access Protocol) is a W3C (World Wide Web Consortium) standard defining protocols for passing objects using XML. A SOAP runtime system (an implementation of the SOAP standard) enables a client to call methods on a SOAP-enabled service, passing objects in XML format.

The following summary is from the abstract of the W3C SOAP 1.1 specification. See <http://www.w3.org/TR/2000/NOTE-SOAP-20000508> for more information.

“SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to

process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.”

The payload of a SOAP RPC message contains application data (objects and primitive types) serialized and represented as XML.

Note – The web services generated by the Sun ONE Studio 4, Enterprise Edition for Java IDE are based on Java™ API for XML-based Remote Procedure Calls (JAX-RPC), which is an implementation of the SOAP specification.

WSDL

WSDL (Web Service Description Language) is a W3C standard, XML-based language used to describe a web service’s external interface.

You can make a web service available to other users by giving them a link to its WSDL file. UDDI registries, described later in this chapter, provide a place where you can publish information about a web service. Developers who search the UDDI registry and decide to use your service can find the location of the service and its WSDL file, and use this information to create a SOAP client capable of issuing remote requests to your service.

The following summary is from the abstract of the W3C WSDL 1.1 specification. See <http://www.w3.org/TR/2001/NOTE-wsdl-20010315> for more information.

“WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to enable description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.”

UDDI

UDDI (Universal Description, Discovery, and Integration) is a protocol for web-based registries that contain information about web services.

A UDDI registry can be *public*, *private*, or *hybrid*. These descriptive terms are not technical distinctions, but differences in the scope of a registry and how it is used.

- **A public registry is widely available over the Internet.** The registry owner enables other developers (including competitors) to publish their services in the registry and enables the general public to search the registry and download registry entries.
- **A private registry is restricted to a single enterprise.** In this case the registry facilitates the sharing of business components throughout the enterprise.
- **A hybrid registry is available beyond a single enterprise, but with restrictions.** For example, a registry might be available to business partners or to recognized development groups in an industry. The registry host determines who can publish services in the registry and who can search the registry.

A registry entry for a web service has enough information so that a developer can create a client application capable of binding to the service and calling its methods. The term “client” is relative, referring to any component that issues calls to the web service. A registry entry must contain the location of the WSDL file describing a service (needed to create the client) and the location of the runtime service (needed to run the client).

A registry entry can contain additional information about the service and its provider. For example, potential users might want to know how committed the provider is to supporting the service and whether the provider has “credentials” or a reputation as a trustworthy source for the service. This issue does not originate with web services. The question of trustworthy sources arises with SSL-based certificates, where certain companies are widely recognized as certificate-granting authorities. The same question arises with older technology, as in the credit card industry or, to take a governmental example, the National Bureau of Standards.

A public registry might grow to thousands or even millions of services. Therefore, the UDDI standard supports categorization of services. A variety of taxonomies are already available based on categories such as industry, geographical region, product, and service. UDDI registry implementations provide facilities for publishing services and for searching a registry.

See <http://www.uddi.org> for UDDI specification documents, discussions, articles, and a list of participating enterprises. Sun Microsystems, Inc., is one of the companies participating in the UDDI project.

Publishing and Using a Web Service

FIGURE 1-1 shows how a web service is made available through a UDDI registry. In the example, Company P is the web service *provider* and Company R is the web service *requester*. You can also think of Company P and Company R as divisions within a single company, using a private registry.

1. Company P creates the web service and the WSDL file that describes its external interface. Company P publishes the web service to a UDDI registry, providing information about the web service, including the location of the WSDL file and the location of the runtime web service.

Company P is responsible for running the web service and making it available over a network.

2. Company R searches the UDDI registry for a web service that provides certain functionality, and selects the web service created by Company P.

Many search criteria are possible. For example, Company R might search for a particular version of a web service. The search might be specific, as in the case where the two companies are already working together and Company P gives Company R the UDDI registry identifier for a particular web service.

3. Company R uses the WSDL description and other information about the web service to create a client component capable of making requests to the web service. Company R incorporates the client into its application.
4. Company R runs the application. At runtime, the client component binds to the web service and executes SOAP requests, passing data as XML over HTTP.

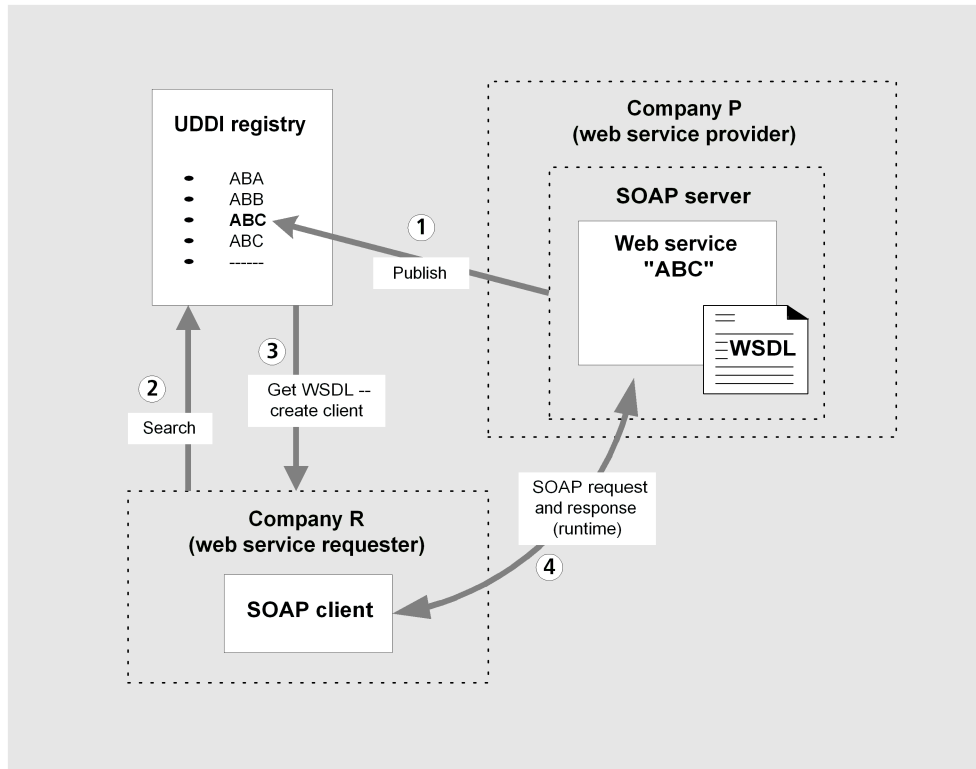


FIGURE 1-1 Publishing and Using a Web Service

The process includes the following tasks:

1. Company P gets permission from the UDDI registry host to write to the registry. Company P establishes itself in the registry through business entries that include name, locator information, and business categorization information.
2. Company P creates an entry in the registry with a URL pointing to the WSDL for its web service. This kind of registry entry is called a technical model (*tModel*).
3. Company P creates one or more service entries in the registry associated with the *tModel*. Each service entry has a URL pointing to a runtime instance of the web service. For example, there might be four different runtime instances: one for use by Company P, one for use by business partners, one for use by the general public, and one for test purposes. All of the instances have the same functionality and the same inputs and outputs, but they can have different performance and availability characteristics.

4. Company R creates a client using information from both a tModel and a service entry in the registry. The tModel, with its pointer to the WSDL, has structural information about the external interfaces of the web service. The service entry points to a runtime instance of the web service and might have additional information about the support, performance, and availability that is offered by Company P for the particular runtime instance.
5. Company R can use this information to create and customize a variety of clients for a given web service.

Sun ONE Studio 4 Java Web Services

Sun ONE Studio 4, Enterprise Edition for Java web services are implemented as J2EE components. You package a web service along with your business components into a single application.

The Sun ONE Studio 4, Enterprise Edition for Java IDE supports the following major functions without the need for coding (assuming that you have a business component available). These functions and others are described in subsequent chapters of this book.

- Creating a JAX-RPC web service
- Generating WSDL describing your service
- Generating a SOAP client for your service
- Testing your service in the IDE with a generated client and JSP component
- Assembling and deploying your service to supported application servers
- Publishing your service to a UDDI registry
- Searching a UDDI registry for a web service
- Generating a SOAP client from an external web service description (a WSDL file or UDDI registry entry)

You can customize the automatically generated entities in the IDE.

A Sun ONE Studio 4 web service can make direct calls to methods of business components. You can also make calls to *XML operations*. An XML operation encapsulates multiple business method calls into a single, higher-level method. This feature is covered in architectural terms later in this chapter, and its use is described in Chapter 4.

You can create a web service based on the methods of an existing business component (a *bottom-up model*), or based on a WSDL file (a *top-down model*). If you create a web service from a WSDL file, the IDE also generates the stub of an enterprise bean. The stub has all the methods designated in the WSDL, but you have to fill in the desired method code. For further information on both models, see Chapter 2.

Multitier Architecture

FIGURE 1-2 illustrates the main components of a web service application developed with the Sun ONE Studio 4, Enterprise Edition for Java IDE. The web service is a logical entity that at runtime spans the web and EJB containers of your J2EE application server and therefore has a *multitier* architecture.

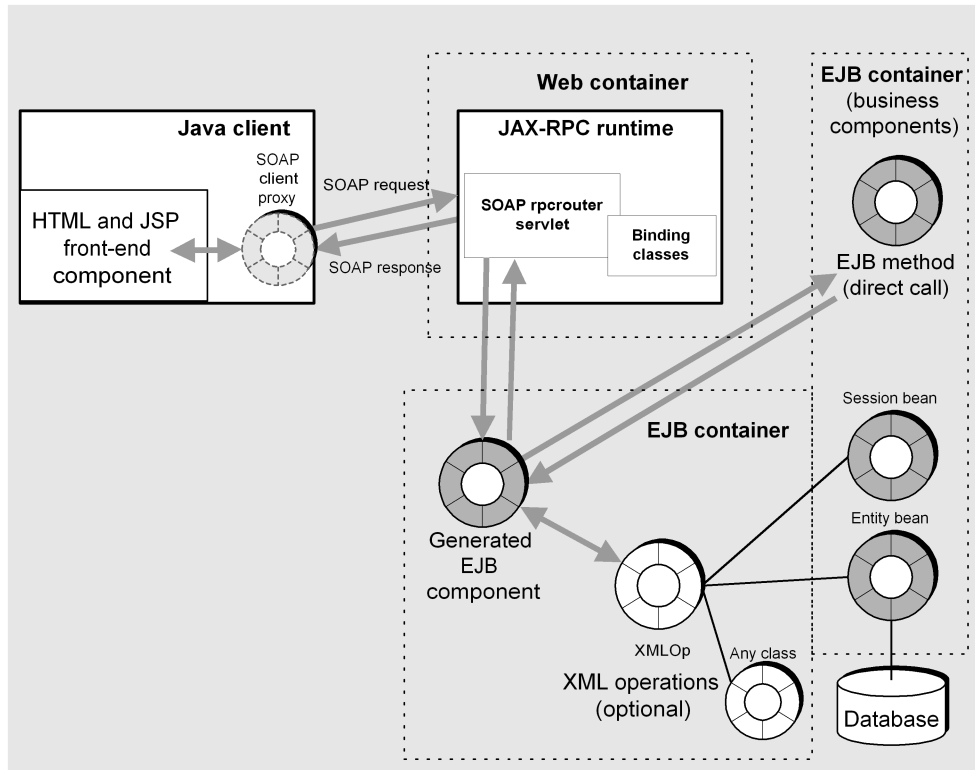


FIGURE 1-2 A Sun ONE Studio 4 Java Web Service and Client (Multitier Model)

On the client side, there is a *SOAP client proxy*. This is a Java class that uses a SOAP runtime system on the client to communicate with the SOAP runtime system on the server, sending SOAP requests and receiving SOAP responses. A SOAP request is an XML wrapper that contains a method call on the web service and input data in serialized form. A SOAP response is an XML wrapper that contains the return value of the method in serialized form. Also on the client side is a *front-end* client component that makes calls to the SOAP client proxy and processes the return value. This component might consist of HTML and JSP pages, or it could be a Java Swing component, another kind of client, or another service.

On the server side, the *JAX-RPC runtime* (an implementation of the SOAP 1.1 standard) is in the web container, handling SOAP messages that pass between the client and the web service.

The SOAP requests are transformed into method calls on a generated EJB component (a session bean, created automatically by the IDE as infrastructure for your web service). The generated session bean in turn makes method calls on business services in an EJB container or invokes an XML operation.

Web-centric Architecture

When you create a new web service, the IDE gives you a choice of architectures. You can choose a multitier architecture (the default), illustrated in FIGURE 1-2, or a *web-centric* architecture, illustrated in FIGURE 1-3. See “Creating a JAX-RPC Web Service from Java Methods” on page 18 for the procedure.

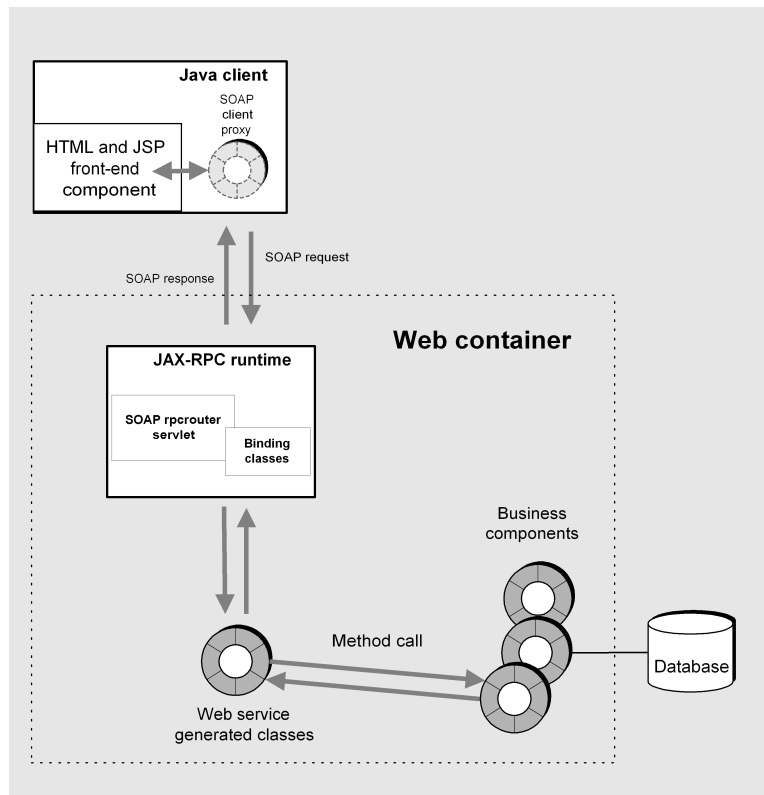


FIGURE 1-3 A Sun ONE Studio 4 Java Web Service and Client (Web-centric Model)

A web service created with the multitier architecture can call business methods only on components in an application server. A web service created with the web-centric architecture can call methods on web tier business components and EJB business components. If a web-centric web service uses only web tier business components, the service can be deployed to a web server, and in this case you do not need an application server.

If you choose the web-centric architecture, the IDE generates a support structure based on Java classes instead of an EJB session bean. This enables the support structure to run in a web container.

Note – The web-centric architecture is more flexible, since it can support both EJB business components and web tier business components. If your application uses only EJB business components, choose the multitier architecture. It's more efficient to have the generated support structure local to the same container as the business EJB components.

In terms of application assembly and deployment, there are three alternatives:

- In the multitier architecture, the IDE creates an EAR file for deployment to an application server.
- In the web-centric architecture with only web business components, the IDE creates a WAR file for deployment to a web server.
- In the web-centric architecture with EJB components, the IDE creates an EAR file for deployment to an application server.

For further information, see “Assembling and Deploying a Web Service” on page 29.

Software Versions

Web standards and technologies are still evolving. The Sun ONE Studio 4 IDE uses the following versions of each standard and software implementation.

- SOAP 1.1 specification
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508>
- JAX-RPC 1.0_01 implementation of the SOAP 1.1 specification
<http://java.sun.com/xml/jaxrpc/>
- Apache SOAP 2.2 implementation of the SOAP 1.1 specification
<http://xml.apache.org/soap/>
- kSOAP 1.0 (a SOAP API suitable for the Java 2 Microedition)
<http://www.ksoap.org/>
- WSDL 1.1 specification
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

- UDDI 2.0 specification
<http://www.uddi.org/specification.html>
- JAXR 1.0_01 API specification
<http://jcp.org/jsr/detail/93.jsp>
- Sun™ ONE Application Server 7
http://sun.com/software/products/appsrvr/home_appsrvr.html
- Java™ 2 Software Development Kit, Standard Edition, 1.4 or 1.4.1 (J2SE™ SDK)
- Java™ 2 Platform, Enterprise Edition (J2EE™ SDK) 1.3.1 (installed with the IDE)

Note – The J2EE SDK is the software development kit for the Java 2 Enterprise Edition Platform and includes the J2EE RI. The J2EE RI is a reference implementation of the J2EE platform intended as a proof of concept and example for implementing a J2EE application server.

Note – The IDE generates JAX-RPC web services. The IDE enables you to choose any of three client-types: JAX-RPC, kSOAP, or Apache SOAP. The Apache SOAP client type is provided for compatibility with the previous release of the IDE.

XML Operations

The IDE makes it easy for you to create web services that expose Java methods of existing application components. However, some EJB components are not designed to be exposed externally through web services. You might have to create a web service interface layer, either by programming additional Java components or by creating XML operations.

Note – You can bypass the XML operation machinery if your web service simply exposes individual methods of a business component. The procedure is explained in Chapter 2.

The IDE user interface enables you to create complex applications based on existing J2EE components without additional coding. You can:

- Create a web service that combines the functionality of multiple business components of different types
- Create a web service that selectively calls multiple methods on a business component in a desired order, passing return values from one method to another

EJB components are often designed to be used by smart clients that understand and can manage the internal logic of the components. The methods might be too low-level to be directly exposed in a web service, or several method calls on different components might be necessary to return a desired object in the desired form.

A new business component that is designed with a web service model in mind might have methods that provide just the right high-level features. However, an existing business component, or a component not specifically designed as the basis for a web service, might not have the necessary higher-level methods.

The IDE solves this problem by providing the *XML operation*, which plays an intermediary role. An XML operation can chain together multiple EJB methods into a single higher-level business function suitable for a web service. Chapter 4 explains how to create XML operations.

What Is an XML Operation?

An XML operation can encapsulate a number of business methods. To an external client, the operation looks like a single RPC call into the web service.

You define an XML operation using a codeless editor. Chapter 4 explains the capabilities of an XML operation and how to use the editor.

The XML operation is a logical entity that specifies how a particular web service request is processed. You create all the XML operations and add them to the web service along with business methods of other components. Then you generate the web service's runtime classes in the IDE. This creates an EJB session bean and one class for each XML operation. When a client sends a request to the web service, the request is transformed into a method call on the session bean. The request is processed either as a single direct call to a business method or as a more complex set of method calls defined by an XML operation.

Note – All the methods in a single XML operation have the same state and can share data for the duration of the operation.

In summary:

- An XML operation in a Sun ONE Studio 4, Enterprise Edition for Java web service plays the role of a high-level business method.
- An RPC call is implemented as a method in a generated EJB component that becomes part of the web service.
- The generated EJB component acts as a client to more low-level component methods of the business components.

- Each XML operation request executes within a single transaction, enabling the grouping of multiple EJB method calls within a single unit.

The XML operations feature has these major benefits, compared with trying to accomplish the same purpose by manual coding:

- You can use the codeless editor to construct an XML operation without extensive Java language expertise. All you need is an understanding of the application components and knowledge of how Java methods work.
- You avoid the substantial effort involved in manually designing, coding, and testing the stateless session bean and retrofitting service interfaces to existing business components.
- You can change the service interface by editing the XML operation in the codeless editor and regenerating the runtime classes for the service components.

Request-Response Mechanism

A Sun ONE Studio 4 web service provides two kinds of external functionality, which you can think of as simple and complex: direct method calls to business components and XML operations. From the standpoint of a client using the web service, they look alike, since an XML operation encapsulates a number of method calls. Each XML operation defines a response to a particular client request message. The web service developer defines and generates the XML operations from existing components.

When a web service receives a client request, it forwards the request in the form of an XML document (the *XML input document*) to the appropriate XML operation. The XML operation calls one or more methods on business components, transforms the return values of these method calls into an XML document (the *XML output document*), and returns the document to the client.

When an XML operation is executed, the web service:

1. Parses the XML input document, mapping the document's elements to the parameters of the methods that the XML operation is defined to call.
2. Calls the methods defined in the XML operation, in their specified order.
Return values from a method can be passed as input to another method, enabling you to construct very rich operations.
3. Formats return values of the methods into an XML output document according to the definition of the XML operation.
4. Returns the XML output document.

For example, FIGURE 1-4 shows an XML operation named `ProductName` that calls methods on three different objects.

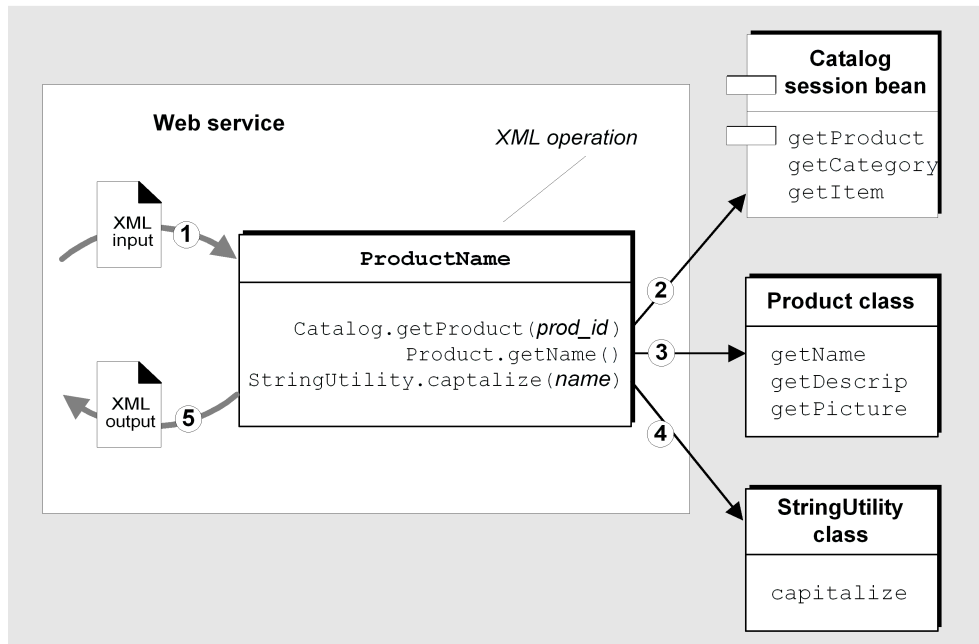


FIGURE 1-4 XML Operation Calling Multiple Methods to Fulfill a Client Request

The `ProductName` XML operation takes the product ID as a request parameter and returns the capitalized name of the corresponding product. When executed, the operation:

1. Parses the XML input document, using the value of the document's `prod_id` element as the input parameter to the `Catalog.getProduct` method.
2. Calls the `Catalog.getProduct` method, which returns an instance of the `Product` class.
3. Calls the `getName` method on the `Product` object, which returns a `String` object that contains the product name.
4. Calls the static method `StringUtility.capitalize`, passing the `String` object that contains the product name as a parameter. This method returns a `String` object containing the product name formatted with initial capital letters.
5. Formats the `String` object containing the capitalized product name as an XML document and returns it.

Building a Web Service

This chapter describes the tools and procedures that you can use to develop a web service. The procedures assume that you have available a business component, such as an enterprise bean, whose methods you want to expose in the web service.

Web Service Development Tasks

The following tasks are interdependent but not strictly sequential. For example, you can add a method reference while creating a web service or afterward. Web service development is an iterative process.

There are two approaches to developing a web service. In the *bottom-up* approach, you use the IDE to build the web service by adding method references from an existing business component such as an enterprise bean. In the *top-down* approach, you use the IDE to generate both the web service and an EJB session bean from WSDL. You can then customize the generated objects.

This chapter describes the following tasks:

- Creating a JAX-RPC web service from Java Methods
- Developing XML operations (if needed)
- Adding method references to a web service
- Creating a JAX-RPC web service from WSDL
- Resolving references to runtime objects
- Adding environment entries
- Generating runtime classes
- Assembling and deploying a web service as a J2EE application
- Creating a test client
- Testing a web service
- Working with UDDI registries

Creating a JAX-RPC Web Service from Java Methods

To create a new web service from Java methods:

1. Open the New Web Service wizard.

In the Explorer, right-click the Java package in which you want to create the web service, and choose New → Web Services → Web Service.

You can also navigate to the same wizard by choosing File → New Template → Web Services → Web Service from the IDE's main window.

The New Web Service wizard is illustrated in FIGURE 2-1.

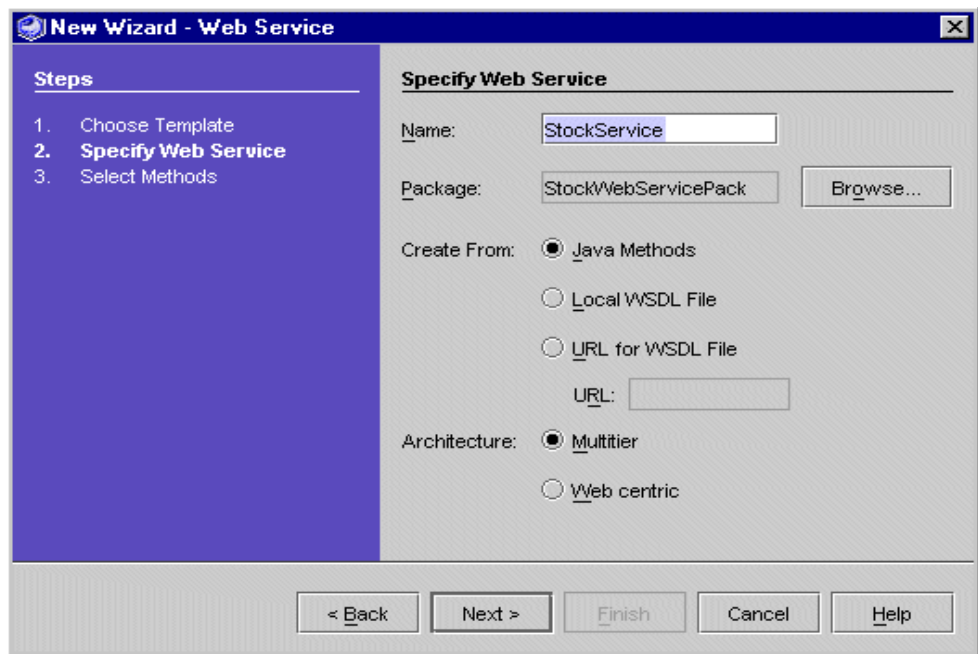


FIGURE 2-1 New Web Service Wizard

Note – The wizard has radio buttons that enable you to choose the web service architecture: Multitier (the default) or Web centric. The distinction between these architectures is explained in “Sun ONE Studio 4 Java Web Services” on page 9. If your application uses only EJB business components, both architectures work. Multitier is preferred because it puts the generated support structure in the same EJB container as the business components.

2. Ensure that the Create From Java Methods radio button is selected.

The alternatives, Create From Local WSDL File and Create From URL for WSDL File, are explained in “Creating a JAX-RPC Web Service from WSDL” on page 22.


3. Ensure that the Package field specifies the correct location in which to create the web service.

If a Java package is currently selected in the Explorer when you open the New Web Service wizard, the name of the package appears in the Package field of the wizard. You can click Browse in the wizard to find and select a package of your choice.

4. In the Name field, type a name for the web service.

If you enter the name of a web service that already exists in the package designated in the wizard, the IDE displays a highlighted message: *A web service with this name already exists in package.*

5. Click Finish to create the web service, or click Next to add method references.

If you click Finish, your new web service is displayed in the Explorer. It appears under the designated package as a node with a blue sphere icon (.

If you click Next, the IDE displays a Methods browser. Select one or more methods and click Finish in the browser to add method references to your web service. Then click Finish to create the web service.

You can add method references after creating a web service (see “Adding References to Methods, XML Operations, and JAR Files” on page 20).

6. In the Explorer, right-click the web service and choose Properties.

The SOAP RPC URL property has the following default form:

`http://hostname:portnum/webserviceName/webserviceName`

Set the values of `hostname` and `portnum` to match your server installation. The first instance of `webserviceName` in the URL is called the *context root* or *web context*. You can change it to any value of your choice, but it must match the web context property of the J2EE application WAR node that you create in a later step (see “Assembling the J2EE Application” on page 30).

Note – If you have more than one web service, make sure that each of the SOAP RPC URLs is unique.

FIGURE 2-2 shows the properties of a web service in which the context root of SOAP RPC URL has been changed from the default value to `StockApp`.

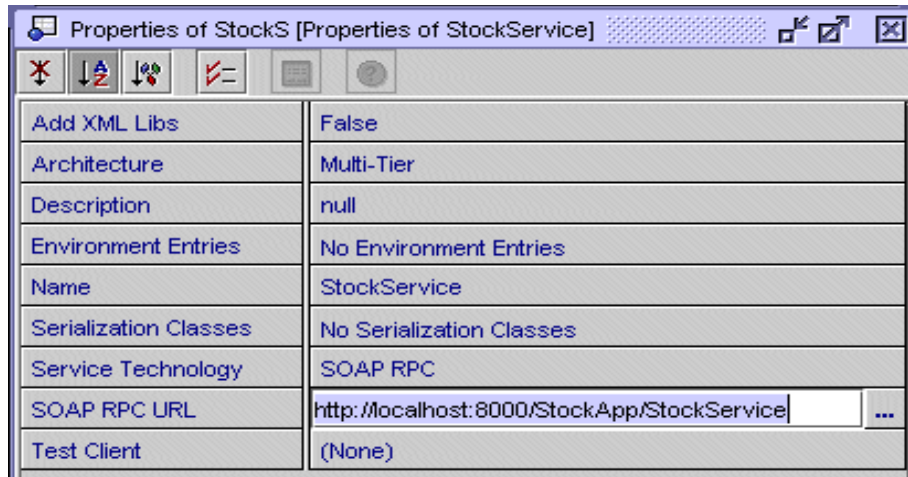


FIGURE 2-2 Web Service Properties With Context Root Set to StockApp

Developing XML Operations

The XML operation is a means of combining multiple business method calls into a higher-level method call for the web service. You do not need XML operations if your web service simply exposes individual methods of a business component.

For information about the role of XML operations, see “XML Operations” on page 13. For information on how to develop XML operations, see Chapter 4.

Adding References to Methods, XML Operations, and JAR Files

Note – You must have available in the IDE an enterprise bean or other business component whose methods you want to expose in the web service.

To add a reference to a web service:

1. **In the Explorer, right-click the web service and choose Add Reference.**

A file browser is displayed.

2. Select the desired business methods, XML operations, and JAR files.


You can select more than one item by holding down the Control key while clicking. Be sure to select EJB methods from the logical EJB node (the node with the bean icon ) , not the nodes that represent the EJB bean class or the home or remote interfaces. By adding a method from the logical EJB node, you provide the runtime information needed to call the method. FIGURE 2-3 shows examples of logical EJB nodes.



FIGURE 2-3 Logical EJB Nodes in the Explorer

3. Click OK.

References added to a web service are displayed in subnodes of the web service, as illustrated in FIGURE 2-4.

References are organized as follows:

- Methods are under the **Methods** node.
- XML operations are under the **XML Operations** node.
- JAR archive files are under the **Libraries** node.



FIGURE 2-4 Web Service References

The IDE prefixes references with the names of the packages containing the referenced objects. The IDE prepends the arrow symbol (->) to XML operation references and JAR references.

Note – You can also add references to a web service using the copy and paste commands. For example, you can select an EJB method, choose Edit → Copy, select your web service node, and choose Edit → Paste.

Deleting References From a Web Service

To delete a reference:

1. **In the Explorer, select the reference you want to delete.**
2. **Right-click the reference and choose Delete.**

The reference is deleted.

Creating a JAX-RPC Web Service from WSDL

To create a new web service from WSDL:

1. **Open the New Web Service wizard.**

In the Explorer, right-click the Java package in which you want to create the web service, and choose New → Web Services → Web Service.

You can also navigate to the same wizard by choosing File → New Template → Web Services → Web Service from the IDE's main window.

The New Web Service wizard is illustrated in FIGURE 2-5.

Note – Alternatively, you can right-click the desired WSDL node in the Explorer, and choose Create New Web Service. In this case the procedure is simplified. The IDE opens a window in which you can enter a target directory and package. The generated EJB session bean is put in the same package as the web service.

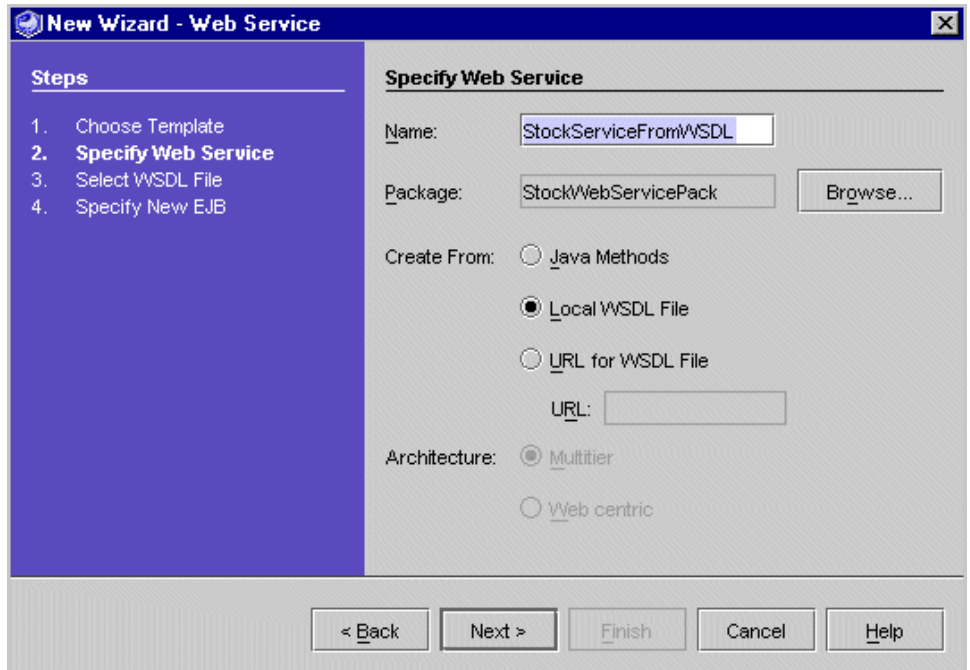


FIGURE 2-5 New Web Service Wizard

2. Check the radio button for Create From Local WSDL File, click Next, and select the desired WSDL file from the browser, as illustrated in FIGURE 2-6.

Alternatively, check the radio button for Create From URL for WSDL File, enter a URL, and click Next.

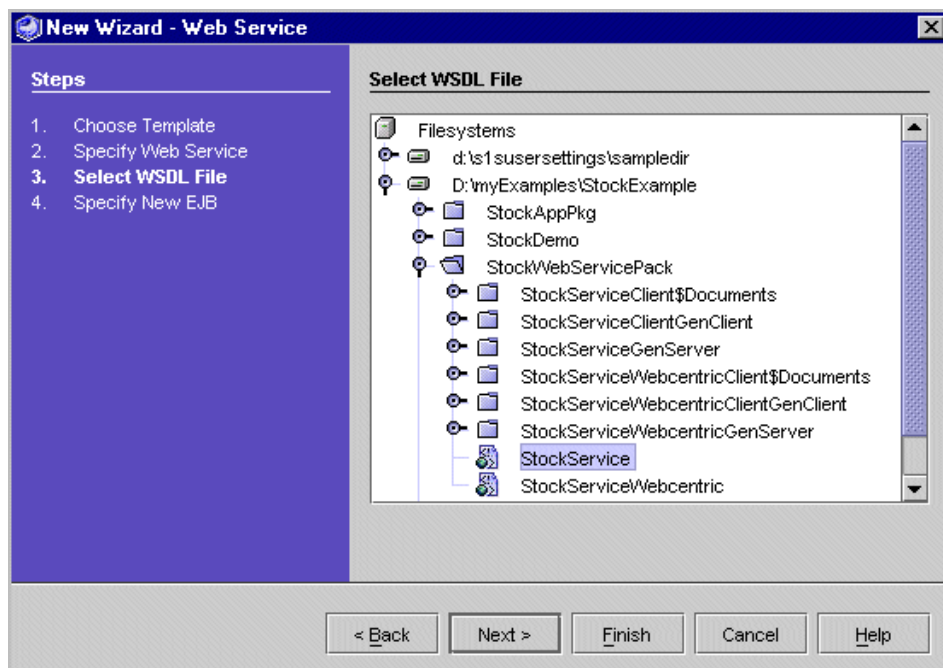


FIGURE 2-6 New Web Service Wizard: Select WSDL File

The IDE displays the Specify New EJB window, as illustrated in FIGURE 2-7.

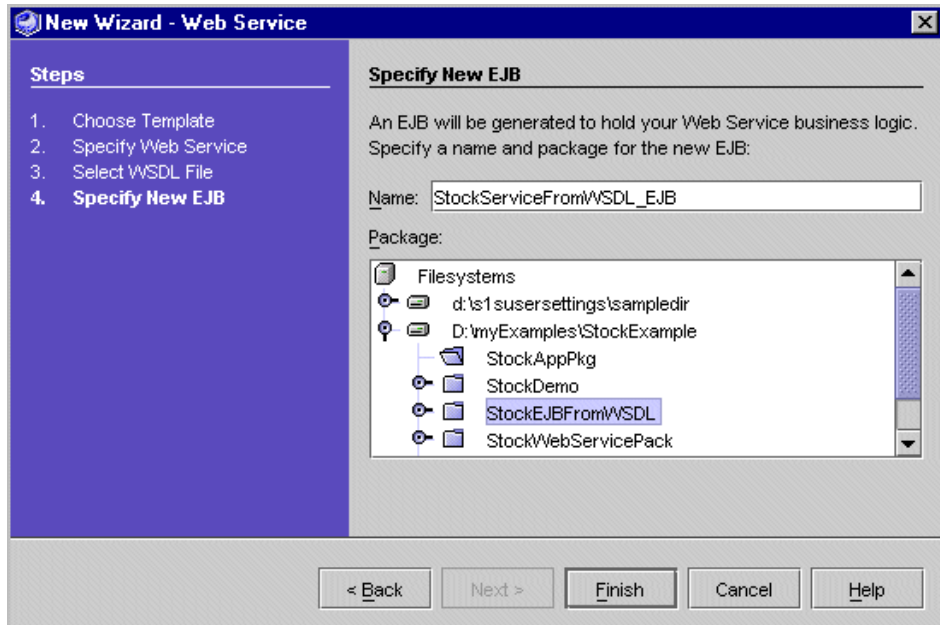


FIGURE 2-7 New Web Service Wizard: Specify New EJB

3. Specify the desired EJB name, select the target package, and click Finish.

The IDE creates the new web service and the new EJB, placing each in the package that you designated in the wizard, as illustrated in FIGURE 2-8.

The new web service and EJB session bean have the methods designated in the WSDL. The web service references those methods in the generated session bean.

Note – The WSDL does not contain the web service business logic. Therefore, the generated EJB session bean is not complete. You have to code the business methods and then compile or validate the EJB classes.

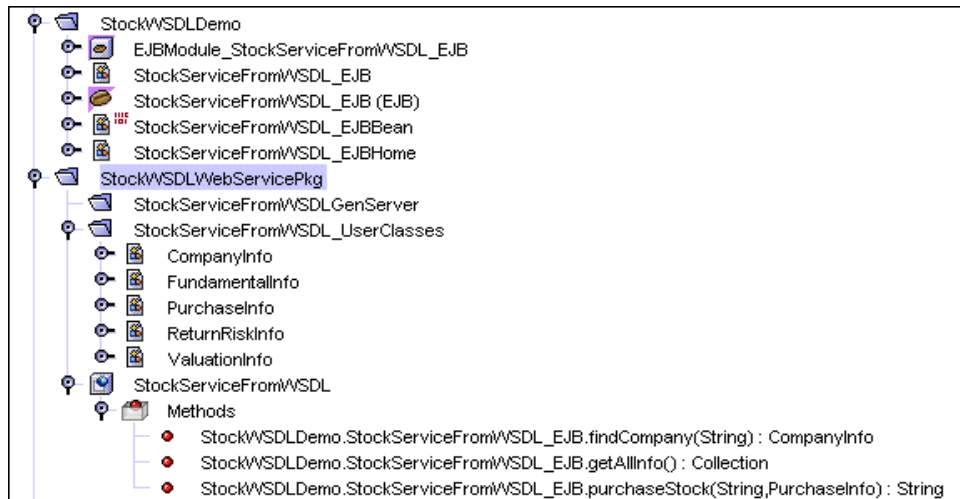


FIGURE 2-8 Web Service and EJB Nodes Generated from WSDL: Explorer View

Resolving References to Runtime Objects

For information on this task, see “Instantiating Objects and Resolving References” on page 54. This step is necessary only if you are using XML operations, and if the default references are not satisfactory.

Adding Environment Entries

Environment entries are data that you store in your web service’s EJB module deployment descriptor. Environment entries are available to your web service for use as parameters to the methods that create or find target objects.

If a method specified in the Method, Constructor, or Static Method fields of the Resolve Objects dialog box takes a parameter, you can map that parameter’s value to an environment entry. Because environment entries are stored in the deployment descriptor, they can be configured at deployment time to values appropriate for the runtime environment.

To add an environment entry:

1. **Right-click your web service node and choose Properties.**

The IDE displays the web service properties, as illustrated in FIGURE 2-2.

2. **Click the Environment Entries property value, then click the ellipsis (...) button.**

The Environment Entries dialog box is displayed, as illustrated in FIGURE 2-9.

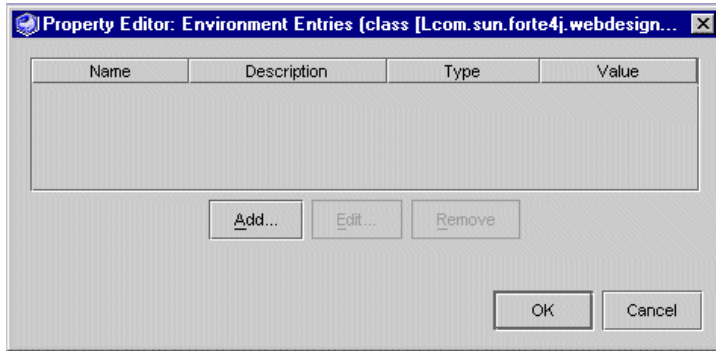


FIGURE 2-9 Environment Entries Dialog Box

3. Click Add.

The Add Environment Entry dialog box is displayed, as illustrated in FIGURE 2-10.

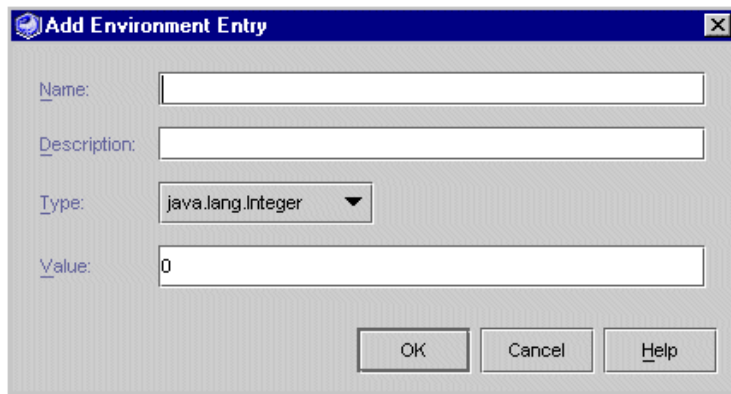


FIGURE 2-10 Add Environment Entry Dialog Box

4. Fill out the fields in the Add Environment Entry dialog box.

- **Name.** Assign a name to the environment entry.
- **Description.** Provide a description of the environment entry.
- **Type.** Enter the Java type of the method parameter to which you will map the environment entry.
- **Value.** Assign the value of the environment entry.

5. Click OK.

The environment entry is created. The next time you generate runtime classes for your web service, the environment entry will be propagated to the web service's EJB module deployment descriptor.

Generating Runtime Classes

Before you can assemble your web service as a J2EE application and deploy it for testing, the IDE must generate your web service's runtime classes.

The IDE places the generated runtime classes in the same package as the web service, under a node named *xxxGenServer*, where *xxx* is the name of the web service. The generated classes differ depending on whether the architecture is multitier or web-centric. For example, in the multitier case, some of the runtime classes are for the generated EJB component.

The IDE generates one additional class for each XML operation, if any. For more information about XML operations, see Chapter 4.

To generate a web service's runtime classes:

1. **Select your web service in the Explorer.**
2. **Right-click and choose Generate Web Service.**

Your web service's runtime classes are generated and compiled. The resulting Explorer display is illustrated in FIGURE 2-11.

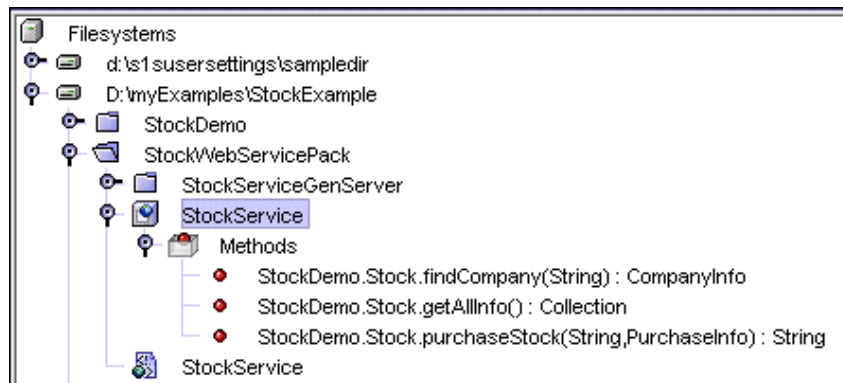


FIGURE 2-11 Web Service Hierarchy of Nodes

A WSDL node appears in the Explorer hierarchy, at the same level as the web service node and in the same package. The WSDL node has the same name as your web service, but it is distinguished by a different icon, with a green sphere (🟢).

You can also generate the WSDL file and node manually by selecting the Generate WSDL menu item from the web service node. For further information about using WSDL files, see "Generating WSDL" on page 37.

Assembling and Deploying a Web Service

The explanation in this section assumes that the EJB modules needed by your web service are available to the IDE, and that you have generated the runtime classes of your web service.

The deployment steps are described for the J2EE Reference Implementation. Deployment should be similar for other application servers, but you might need to consult the person responsible for your application server for information such as host name, port number, and any special requirements.

The assembly and deployment procedure is simplified if your web service has the web-centric architecture and does not reference EJB business methods. The following explanation starts with the simpler web-centric case and then presents the procedure for a web service that is assembled and deployed as a J2EE application.

A Web-Centric Application

If your web service has the web-centric architecture, the IDE creates only a WAR file. In this case, you do not have to create a J2EE application.

If a web service has the web-centric architecture, the web service node has the Deploy and Execute menu items enabled in the Explorer.

- **Right-click the web service node and choose Execute.**

If you are using the default Tomcat web server, the IDE assembles and deploys the WAR file and displays the default test web page in a browser. See “Creating a Test Client” on page 34 for further information.

Note – If you are using the Sun ONE Application Server 7 server, Execute does not automatically deploy the WAR file. You must right-click the web service node and choose Deploy and then Execute.

Alternatively, you can use the following procedure:

1. **Right-click the web service node and choose Assemble WAR File.**

The IDE creates a WAR file in the same package as your web service and displays the WAR node in the Explorer.

2. **Right-click the WAR file node and choose Unpack as Web Module.**

The IDE unpacks the WAR file into the directory of your choice and mounts the directory to the IDE filesystem.

3. **Navigate to the unpacked web module in the Explorer, right-click the `WEB-INF` node, and choose Properties.**

4. **Change the Context Root property to match the SOAP RPC URL property of your web service (see FIGURE 2-2).**

The context root value appears in the SOAP RPC URL, which has the following form: `http://hostname:portnum/contextRoot/webservicename`.

Note – You must give the Context Root property value of `WEB-INF` a leading slash (`/`), as follows: `/contextRoot`.

5. **Deploy the web module to a web server.**

See *Building Web Components* in the Sun ONE Studio 4 Programming series for instructions on how to do this.

The following procedure assumes that your web service uses EJB business components.

Assembling the J2EE Application

To create a J2EE application containing your web service and its referenced components:

1. **Right-click the package in which you want to create the J2EE application and choose New → J2EE → Application.**

The New Application wizard is displayed.

2. **Type a name for the application in the Name field and click Finish.**

A J2EE application node is added to the package.

3. **Right-click the application node and choose Add Module.**

The Add Module to Application dialog box is displayed.

4. **Select the web service and all EJB modules referenced by it, and click OK.**

FIGURE 2-12 shows a J2EE application node with these subordinate nodes:

- An EJB JAR node named `webserviceName_ejbJar`, where `webserviceName` is the name of your web service. This node corresponds to the generated EJB module of the web service.
- A WAR node named `webserviceName_war`, where `webserviceName` is the name of your web service. This node corresponds to the generated web module of the web service.
- An EJB JAR node for each EJB module you added to the application.

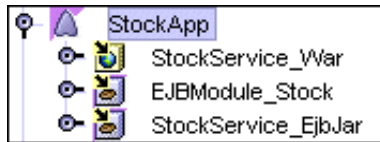


FIGURE 2-12 J2EE Application for a Web Service

5. Edit the Web Context property of the web service WAR node, if necessary.

A default web context property is assigned to the web service WAR node. The value of this property is part of the URL used to access your web service. The value must match the *context root* value in the SOAP RPC URL property of your web service (see FIGURE 2-2).

To edit the default setting:

- a. Right click the web service WAR node and choose Properties.**
- b. Click the Web Context property, type a new value, and press Enter.**

FIGURE 2-13 shows the properties of a web service WAR file in which the Web Context value has been changed to *StockApp*, to match the context root value in the SOAP RPC URL property of the web service.

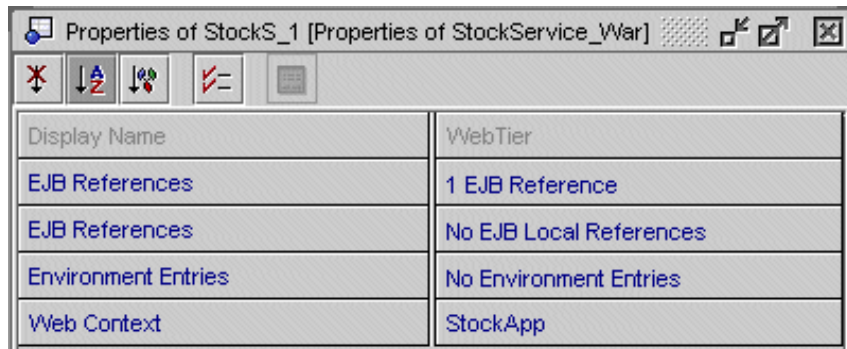


FIGURE 2-13 Web Service WAR File Properties With Web Context Set to StockApp

6. Edit the Application Server property of the J2EE application, if necessary.

This property should specify the desired application server instance for deployment. The property defaults to the default application server of the IDE, which you can set in the Explorer window's Runtime tab.

To edit the property, right-click the J2EE application node and choose Properties. Click the Application Server property value, and click the ellipsis (...). The IDE displays a dialog box in which you can select any available application server instance, as illustrated in FIGURE 2-14.

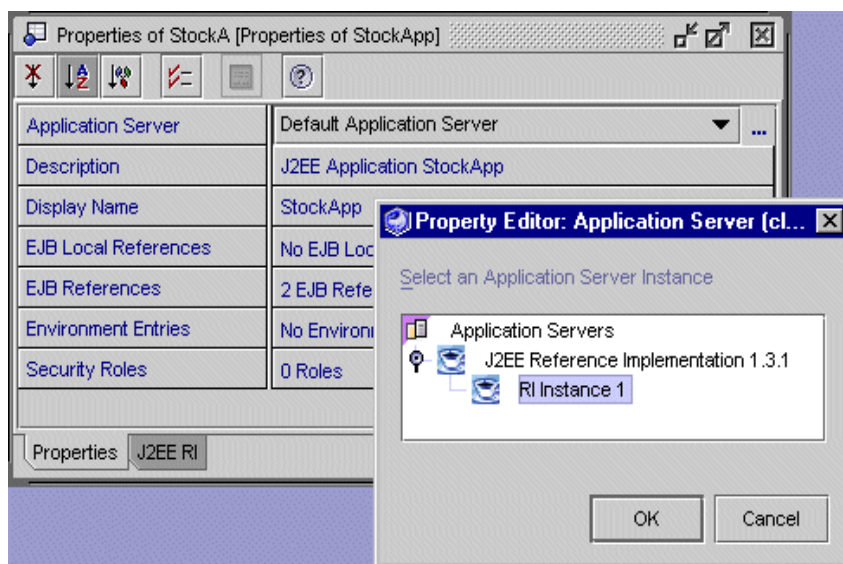


FIGURE 2-14 Application Server Instance for Deployment

Configuring the RI Server

Note – If you are using the RI server that comes installed with the Sun ONE Studio 4, Enterprise Edition for Java IDE, skip Steps 1-5 of the following procedure.

To configure an external RI application server:

1. Click the **Runtime** tab in the **Explorer** window.
2. Expand the **Server Registry** node, and then expand the **Installed Servers** node.
3. Add a **J2EE Reference Implementation** server instance by right-clicking the **J2EE Reference Implementation** node and choosing **Add Server Instance**.

The **Explorer** window's **Runtime** tab is illustrated in FIGURE 2-15.

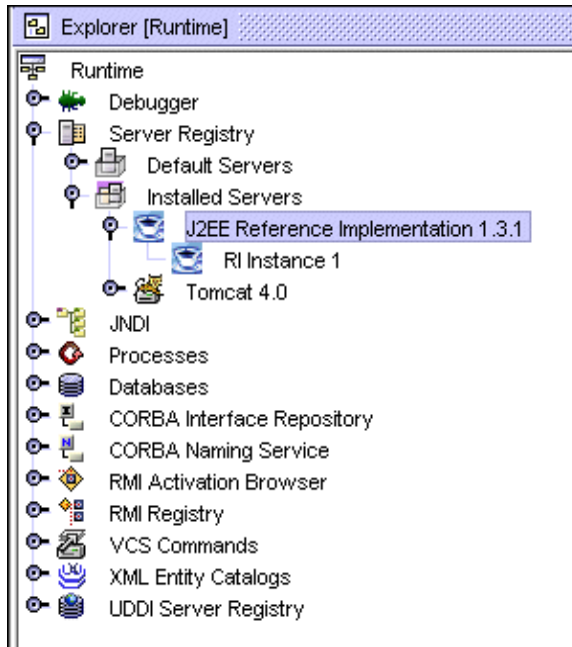


FIGURE 2-15 Creating an Application Server Instance

4. Open the property sheet for the J2EE Reference Implementation.

Right-click the J2EE Reference Implementation node again and choose Properties.

5. Set the value of the RIHome property to the directory where the J2EE SDK is installed.

6. (Optional) Configure database access.

Most EJB components require that you configure the RI server for database access. Edit the `J2EE_HOME/config/resource.properties` file and add the necessary drivers and data sources. (J2EE_HOME is an environment variable and refers to the directory where your J2EE SDK is installed.)

You can use RI server administration commands to edit the `resource.properties` file. For example, open a command window and enter the following commands:

```
j2eeadmin -addJdbcDriver yourDriver
j2eeadmin -addJdbcDatasource yourDatasource
```

See the *Sun ONE Studio 4, Enterprise Edition for Java Getting Started Guide* for a more complete description of this subject.

Deploying the J2EE Application

To deploy your web service application to the J2EE RI server:

1. **Right-click the J2EE application node and choose Deploy.**

This action starts the packaging process. When the IDE has finished packaging the application, it deploys the application to the RI server.

2. **Watch for packaging and deployment status messages.**

The process might take a few minutes. The IDE status line displays messages indicating the packaging progress. The console of the RI server indicates the deployment progress.

When deployment has completed, your web service is ready to be tested.

Note – It is not necessary for you to start the RI server. When you deploy an application, the IDE automatically starts the server. If you are using an external RI server, make sure that it is not running just before you begin deploying.

Creating a Test Client

This section lists the client development features of the Sun ONE Studio 4, Enterprise Edition for Java IDE, and describes how to create a test client. See Chapter 3 for more information and procedures on how to create web service clients.

Web Service Client Features

The IDE supports the following tasks through interactive dialog boxes and menu commands:

- Generating a client from a Sun ONE Studio 4, Enterprise Edition for Java web service
- Generating a client from WSDL
- Generating a client to access a service published in a UDDI registry

Setting a Default Test Client for a Web Service

When you use the IDE to generate a client from your web service, the New Client dialog box displays a checkbox that enables you to make that client the default test client for the service.

To change the default test client, right-click the web service node in the Explorer, choose Properties, and edit the Test Client property.

Note – When you make a client the default test client for a web service, the IDE automatically packages that client in the web service WAR file when the web service is assembled into a J2EE application.

Testing a Web Service

You can use a default test client to test your web service. There are two procedures, depending on the architecture of your web service.

Testing With a J2EE Application

If your web service has a multitier architecture or has a web-centric architecture and uses EJB business methods:

1. **Generate a client and make that client the default test client for your web service.**
2. **Assemble your web service into a J2EE application.**
3. **Right-click the application node and choose Execute. Alternatively, right-click the test client node and choose Execute.**

The IDE deploys the web service application, runs it in the J2EE application server that is specified in the J2EE application, and executes the default test client. If your web service is assembled without a default test client, the IDE displays a page saying that there is no input page.

Alternatively, you can deploy the web service application, then right-click the test client node and choose Execute. The IDE assembles the test client, deploys its WAR file to the IDE's default Tomcat web container, and executes the client.

Testing Without a J2EE Application

If your web service has a web-centric architecture and does not use any EJB business methods, there is no J2EE application. In this case, you can:

1. **Generate a client and make that client the default test client for your web service.**
2. **Make sure the SOAP RPC URL property of the web service refers to the port of the internal Tomcat web server installation (8081).**

To view or change the SOAP RPC URL property, right-click the web service node in the Explorer and choose Properties.

3. Right-click the web service node and choose Execute.

The IDE assembles a WAR file, deploys it to the default Tomcat web server, and runs the web service. To view the WAR properties, right-click the WAR node in the Explorer and choose Properties.

Note – If you are using the Sun ONE Application Server 7 server, Execute does not automatically deploy the WAR file. You must right-click the web service node and choose Deploy and then Execute.

Iterative Testing

Testing is an iterative process. Edit your web service, regenerate the runtime classes and client, and test until satisfied.

Working With UDDI Registries

This section explains how to use the IDE to make your web service available to other developers through a UDDI registry.

Several basic tasks are required:

1. Deploy your runtime web service to an application server or web server that is accessible to other developers and end-users over a network.
2. Generate WSDL from your web service and publish the WSDL to a web server that is accessible to other developers over a network.
3. Publish your web service to a UDDI registry that is accessible to other developers over a network.

The IDE does tasks 1 and 2 when you deploy your web service. The IDE provides a wizard for task 3, which is the main subject of this section.


Note – Client development is described in Chapter 3, including information about how to find a web service in a UDDI registry and how to generate a client that can use the web service.

Generating WSDL

You can share WSDL files with other developers without necessarily using a UDDI registry. WSDL files can be put on a shared drive or sent as email attachments. For information about how to generate a client from WSDL (without a UDDI registry), see “Creating a Client From WSDL” on page 68.

When you generate the runtime classes for your web service, the IDE automatically creates a WSDL file and displays it as a node in the Explorer. You can also generate and view a WSDL file as follows:

1. **Right-click your web service node in the Explorer and select Generate WSDL.**

A WSDL node appears in the Explorer hierarchy, at the same level as the web service node and in the same package. The WSDL node has the same name as your web service, but it is distinguished by a different icon, with a green sphere ().

2. **Right-click your WSDL node in the Explorer and select Open.**

The Source Editor window is displayed in read-only mode. You can read and copy the WSDL, which is an XML document. You can also find the WSDL file in your directory hierarchy, under the directory of your web service package. The file is *webserviceName.wsdl*.

Note – If you deploy your web application and use the IDE to publish the web service to a UDDI registry, the IDE’s deployment and publication processes manage the WSDL for you.

Managing UDDI Registry Options

The IDE keeps a list of known UDDI registries and information (such as URLs, userid, and password) needed to access each registry. The IDE also keeps default information (such as business, categories, and identifiers) that is used when you create a web service client or publish a web service to a registry.

Note – It is recommended that you set default values for registry information before you use the IDE wizards to access UDDI registries. By setting defaults, you avoid repetitive typing of values in the wizards.

To manage UDDI registry options, choose Tools → Options → Distributed Application Support → UDDI Registries from the IDE's main window. The Options dialog box is displayed, as illustrated in FIGURE 2-16.

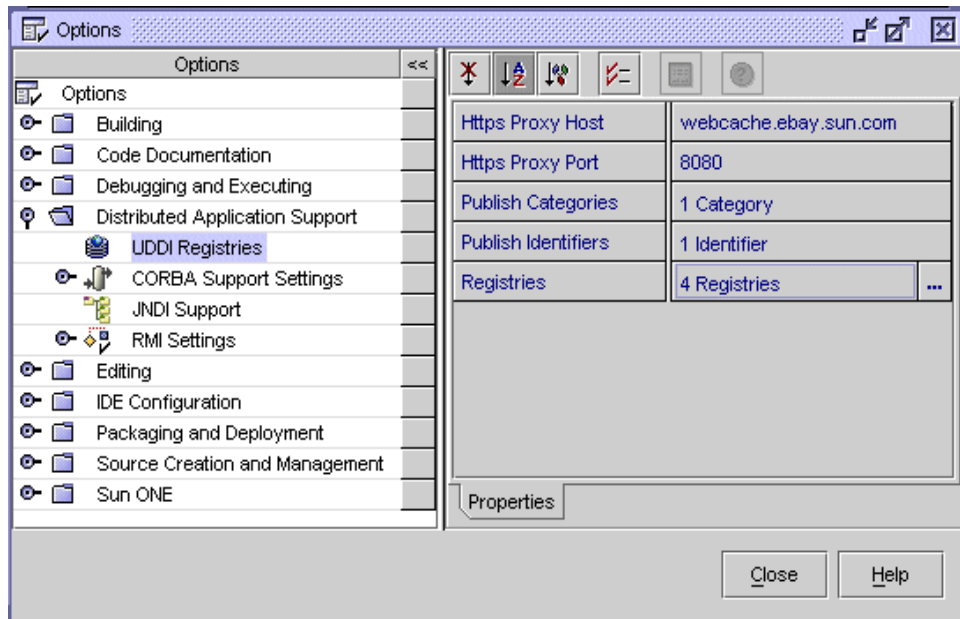


FIGURE 2-16 UDDI Registries Options Dialog Box

Setting Default Publish Categories and Identifiers

The publish categories and publish identifiers associated with your web service are saved in a UDDI registry when you publish the web service.

An identifier is a data element that is unique to a business or technical model (tModel). A category is a data element that classifies a business, service, or tModel. Categories or identifiers can be specified in user queries to locate a business, service, or tModel in a registry.

To set default publish categories, click the Publish Categories value in the UDDI registries Options dialog box (see FIGURE 2-16) and click the ellipsis (...) button that appears. The Publish Categories dialog box appears, as illustrated in FIGURE 2-17.

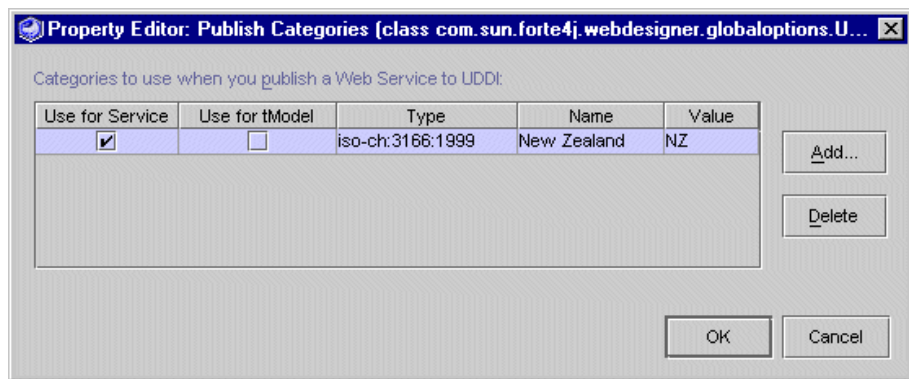


FIGURE 2-17 UDDI Publish Categories Dialog Box

Click the Use for Service checkbox or the Use for tModel checkbox to set a category as a default for services or tModels when you publish them to a registry.

Click Delete to delete a category.

Click Add to add a category. The UDDI Categories (taxonomies) dialog box appears, as illustrated in FIGURE 2-18. The categories are industry standards.

Note – When you publish a web service to a UDDI registry, the IDE automatically adds a hidden category to be associated with the tModel. The name of the category is `uddi-org:types` and the value is `wsdlSpec`. This category indicates that the tModel has a WSDL description of a web service. It only applies to tModels, and is not used to categorize web services. This category is not displayed in the IDE, but it can be found in searches by registry tools.

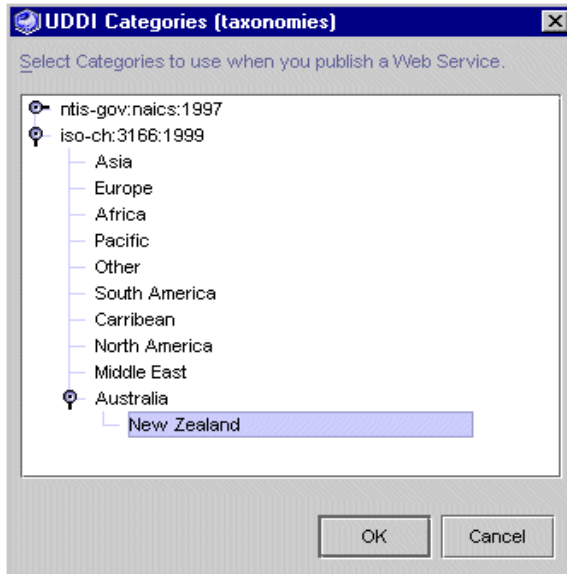


FIGURE 2-18 UDDI Categories (Taxonomies)

Expand the nodes, select the desired category, and click OK. The category appears in the Publish Categories dialog box. To select more than one item, hold down the Control key while clicking. To select a range of items, hold down the Shift key while clicking.

To set default publish identifiers, click the Publish Identifiers value in the UDDI registries Options dialog box (see FIGURE 2-16) and click the ellipsis (...) button that appears. The Publish Identifiers dialog box appears, as illustrated in FIGURE 2-19.

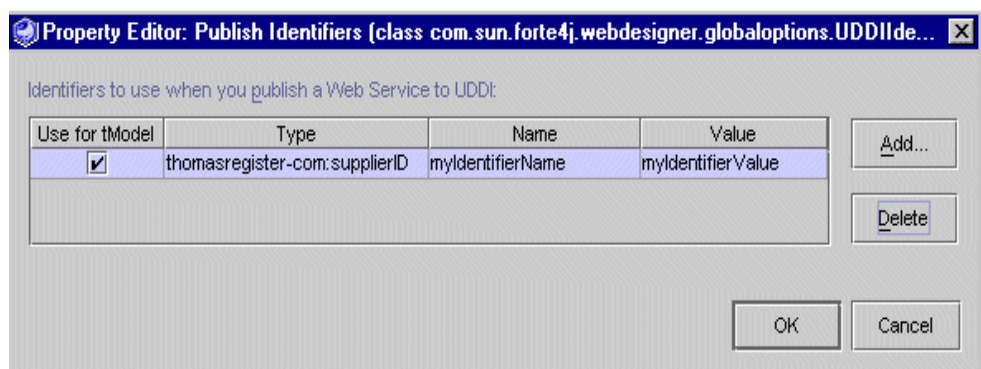


FIGURE 2-19 UDDI Publish Identifiers Dialog Box

You can delete an identifier, add an identifier, or set an identifier as a default for tModels.

To delete an identifier, select the identifier and click Delete.

To add an identifier, click Add. The UDDI Add Identifier dialog box appears, as illustrated in FIGURE 2-20. The identifier types and tModel UUIDs are industry standards.

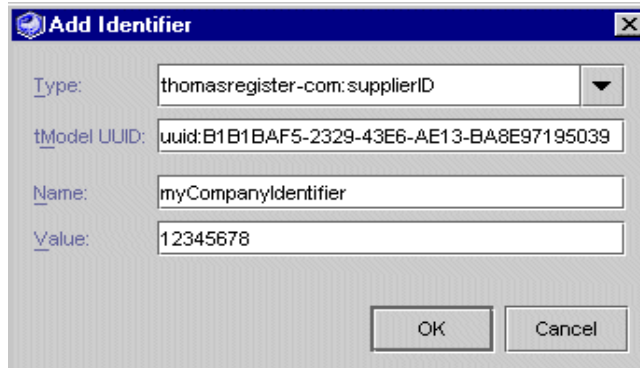


FIGURE 2-20 UDDI Add Identifier Dialog Box

Select Type. The IDE displays the tModel UUID for the selected type. You can specify your own user-defined type if you know and enter the tModel UUID (the registry key) of your user-defined type.

Specify Name and Value, and click OK. The IDE returns you to the Publish Identifiers dialog box.

Click OK. The IDE returns you to the UDDI Set Categories and Identifiers dialog box.

Editing Registries Information in the IDE

To edit information about registries known to the IDE, click the ellipsis (...) button of the Registries property in the UDDI registries Options dialog box (see FIGURE 2-16). The Property Editor dialog box is displayed, as illustrated in FIGURE 2-21.

The property editor displays known registries and detail information for the selected registry. The top portion of the screen is a table of known registries. Click a registry name to display its detail information in the bottom portion of the screen.

You can edit registry information as follows:

- To add a registry to the list, click Add. Provide a descriptive registry name and values for the Query URL, Publish URL, and Browser Tools URL.
- To delete a registry, select it and click Delete.

- To change the default registry, select the desired registry and click its checkbox in the Default column.
- To edit a registry name, double-click the current name and type the new name.
- To set a default login, specify values for Name and Password.

| UDDI Registry | Default |
|------------------------|-------------------------------------|
| Internal UDDI Registry | <input checked="" type="checkbox"/> |
| IBM | <input type="checkbox"/> |
| Microsoft | <input type="checkbox"/> |

Query URL:

Publish URL:

Browser Tools URL:

Default Login

Name:

Password:

Repeat Password:

Default Business

Name:

Key:

Add Delete OK Cancel

FIGURE 2-21 UDDI Registries Property Editor

You can also set a business that the IDE will use as a default in its UDDI wizards. To set a default business, specify values for Name and Key.

Note – You can set a default login name and password. This is convenient but it is a security risk in some environments because the password is stored in your IDE filesystem and is not encrypted. It is more secure to type the password each time.

Note – You can edit UDDI registry properties while you are publishing a web service or searching the registry for the purpose of generating a client. See “Creating a Client: Procedure” on page 69.

Gaining Access to an External UDDI Registry

In order to access a UDDI registry, you have to provide the IDE with certain information (see “Editing Registries Information in the IDE” on page 41).

To search a registry and generate a web service client, you need the Query URL. To publish a web service to a registry, you need the Publish URL. Get this information from the *registry operator*, that is, the organization managing the registry.

Security can vary from one registry to another. You might need a login name and password to publish to a registry. Get this from the registry operator. Some public registries provide tools that you can run from a web browser to set up your own account.

Publishing Your Web Service to a UDDI Registry

This section begins by explaining publication tasks and terminology, and then describes the publication procedure.

Publication Tasks and Terminology

A web service entry in a UDDI registry is associated with a *business* entry, a *technical model* (*tModel*) entry, and optional categories and identifiers. When you publish a web service to a UDDI registry, the IDE wizards help you to carry out the following tasks:

- Add a new business entry or find an existing business entry, and associate the business entry with the web service entry. (Businesses added by this wizard do not have associated categories and identifiers.)
- Add a new tModel entry or find an existing tModel entry, and associate the tModel entry with the web service entry
- Optionally associate one or more standard industry categories and identifiers with the web service entry or tModel entry

The business is the organizational entity that contains the web service. Each business entry in the registry has a unique key.

The tModel is a registry entry that contains a URL (called the overviewURL) pointing to your WSDL, which describes the external interfaces of your web service. This information is used by the IDE or other developer software to create clients that can call methods on your web service at runtime.

The web service entry in a UDDI registry specifies an *entry point*, also called the *endpoint URL* or *service URL*. The URL is used by a client to find a runtime instance of your web service. During publication, the IDE sets this URL by default to the value of the SOAP RPC URL property of the web service. You can edit the value.

Standard industry categories and identifiers can be associated with your web service entries and tModel entries in a UDDI registry to facilitate subsequent searches. For more information about UDDI registry searches, see “Creating a Client From a UDDI Registry” on page 69.

A business can publish many different services to a UDDI registry, where each service is described structurally by WSDL associated with a tModel entry. There can be many service entries that reference the same tModel. For example, as a service provider you might support one instance of a service for use within your enterprise, another instance for business partners, and another instance for the general public. The various instances have the same external interface, as described in the WSDL referenced by the tModel, but they might have different performance or availability characteristics.

When you publish a web service, the IDE automatically includes a reference to its tModel.

Publication Procedure

Before starting this procedure, you must deploy your web service application. The IDE places a copy of the service’s WSDL file in the deployed service. When you publish the service, the tModel refers to this WSDL file.

To publish a web service to a UDDI registry:

- 1. Right-click your web service node in the Explorer and select Publish to UDDI.**

The Publish New Web Service to UDDI dialog box is displayed, as illustrated in FIGURE 2-22.

In this dialog box, you can:

- Set the Service Name in the UDDI registry
- Set the Network Access Point Type to http (the default) or https
- Edit the Network Access Point Address URL, which defaults to the value of the SOAP RPC URL property of your web service
- Specify text for the Service Description

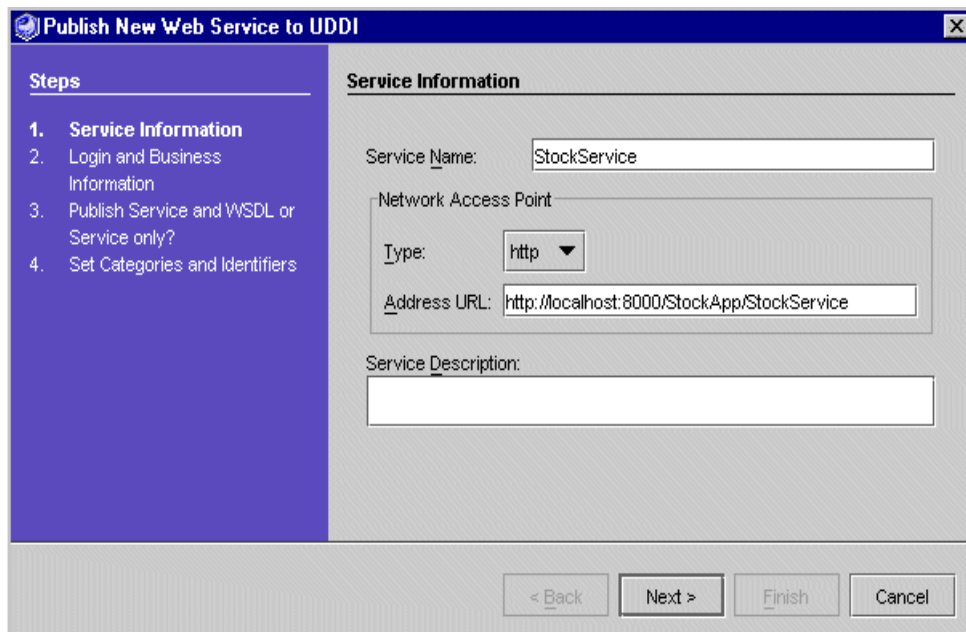


FIGURE 2-22 UDDI Publish New Web Service Dialog Box

Note – The host name in Address URL defaults to localhost. Change this to the network machine name of the host.

2. Click **Next** to display the UDDI Login and Business Information dialog box, as illustrated in FIGURE 2-23.

In this dialog box, you can:

- Select the target UDDI registry for the publication
- Type a userid and password that entitles you to publish to the selected registry
- Find an existing business in the registry, or add a new business

If you previously set default values for userid, password, and business for the selected UDDI registry, the default values are displayed. You can override a default value by typing or pasting a different value in each field.

To specify a business, type a value for Business Name. The Find and Add buttons become active.

- **Click Find for an existing business.** The IDE displays the existing value of Business Key. If more than one business in the registry matches the value of Business Name, the IDE displays the first Business Key value returned by the registry server.
- **Click Add for a new business.** The IDE adds a business entry to the UDDI registry, and displays the generated value of Business Key. A business added this way has no associated categories or identifiers.

Click Edit to edit information for the selected UDDI registry. The IDE displays the UDDI Registries Property Editor. This enables you to change registry default values during the publication process. See “Editing Registries Information in the IDE” on page 41.

Login and Business Information

Service Publication to a UDDI registry requires a login at the registry and a Business in which to publish the Service.

UDDI Registry: Internal UDDI Registry Edit...

User Id: testuser

Password: *****

Business Name: CompanyA Find Add

Business Key: PM240140-1221-8112-3722-3ef86c528da8

< Back Next > Finish Cancel

FIGURE 2-23 UDDI Login and Business Information Dialog Box

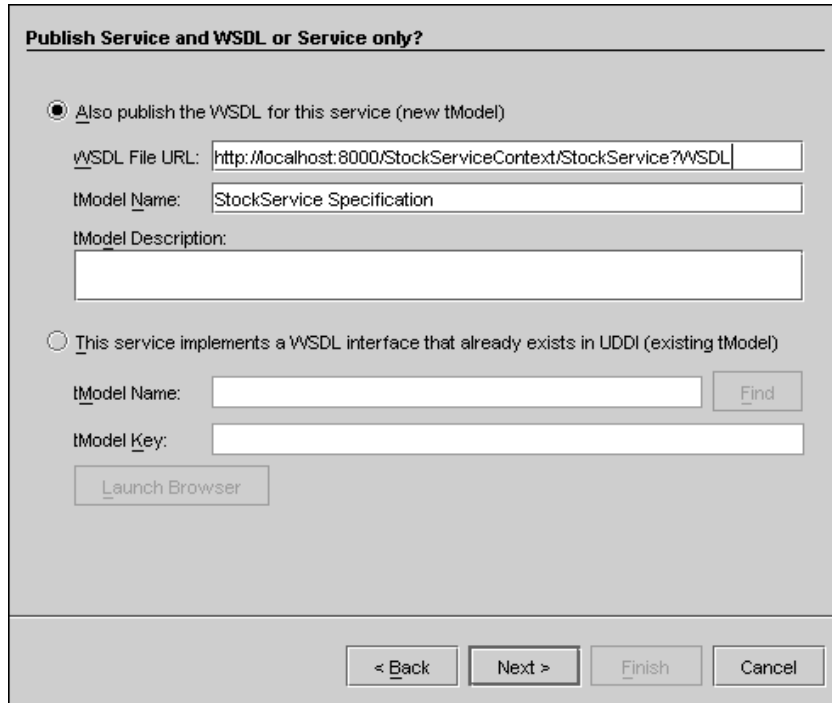
3. Click Next to display the UDDI tModel selection dialog box, as illustrated in FIGURE 2-24.

In this step you specify what you want to publish and provide details. You can create a new tModel in the registry or find an existing tModel, and associate the new or existing tModel with your web service.

If you create a new tModel, the IDE attempts to derive the WSDL File URL from the Address URL of your deployed web service (see FIGURE 2-22), based on IDE naming conventions. If you changed the Address URL, or if you deployed your web service outside the IDE, the IDE leaves the WSDL File URL value blank.

You can change the URL by typing a different value.

You can specify text for the tModel Description.



The dialog box is titled "Publish Service and WSDL or Service only?". It contains two radio buttons. The first radio button is selected and is labeled "Also publish the WSDL for this service (new tModel)". Below this, there are three text input fields: "WSDL File URL:" with the value "http://localhost:8000/StockServiceContext/StockService?WSDL", "tModel Name:" with the value "StockService Specification", and "tModel Description:" which is empty. The second radio button is labeled "This service implements a WSDL interface that already exists in UDDI (existing tModel)". Below this, there are two text input fields: "tModel Name:" and "tModel Key:". To the right of the "tModel Name:" field is a "Find" button. Below the "tModel Key:" field is a "Launch Browser" button. At the bottom of the dialog box are four buttons: "< Back", "Next >", "Finish", and "Cancel".

FIGURE 2-24 UDDI tModel Selection Dialog Box

If you use an existing tModel in the UDDI, type the tModel Name and click Find. The IDE will display the tModel Key. If more than one tModel in the registry matches the value of tModel Name, the IDE displays the first tModel Key value returned by the registry server. If you already have the key value, type or paste it into the tModel Key field.

Note – If you are publishing your web service to a UDDI registry that provides browser tools, you can use those tools to search for the tModel key. Click Launch Browser. The IDE opens your default web browser at the web page specified by the Browser Tools URL of the UDDI registry (see FIGURE 2-21).

4. Click Next.

The IDE displays the UDDI Set Categories and Identifiers dialog box, as illustrated in FIGURE 2-25.

The dialog box shows your default categories and identifiers. Click Edit to add or delete categories or identifiers, as described in “Setting Default Publish Categories and Identifiers” on page 39.

Set Categories and Identifiers

Set Categories and Identifiers for this Service and tModel (used later in UDDI searches).

Categories (taxonomies):

| Use for Service | Use for tModel | Type | Name | Value |
|-------------------------------------|--------------------------|---------------------|------------------|-------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | iso-ch:3166:1999 | New Zealand | NZ |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | ntis-gov:naics:1997 | Manufacturing-31 | 31 |

Edit...

Identifiers:

| Use for tModel | Type | Name | Value |
|-------------------------------------|-------------------------------|------------------|-------------------|
| <input checked="" type="checkbox"/> | thomasregister-com:supplierID | myIdentifierName | myIdentifierValue |

Edit...

< Back Next > Finish Cancel

FIGURE 2-25 UDDI Set Categories and Identifiers

5. Click Finish.

The IDE publishes your web service to the UDDI registry. This can take several minutes. A success message is written to the IDE’s top window status line.

The Internal UDDI Registry

The UDDI registry server from Sun's Java™ Web Services Developer Pack (Java WSDP) is integrated with the IDE as a single-user, internal UDDI registry is bundled with the IDE as a convenience for end-to-end testing of your development process. This registry consists of a servlet that runs in a dedicated Tomcat web server (different from the internal Tomcat server used for web application development) and uses an XML database for persistent storage. The IDE automatically starts and stops the Tomcat server and the database server when you start and stop the registry server.

Note – The internal UDDI registry is configured with a single user. The name is testuser and the password is testuser. Set this name and password as the default for the internal registry.

Starting and Stopping the Internal UDDI Registry Server

To start the internal UDDI registry server:

1. **Expand the UDDI Server Registry node in the Explorer Runtime tabbed pane.**
You should see the Internal UDDI Registry node.
2. **Right-click the Internal UDDI Registry node and choose Start Server, as shown in FIGURE 2-26.**

The IDE output window displays server startup messages. You might also see messages stating that the IDE is stopping a previous Tomcat server process.

Note – If the internal UDDI registry server is already running, the Start Server menu item is inactive.

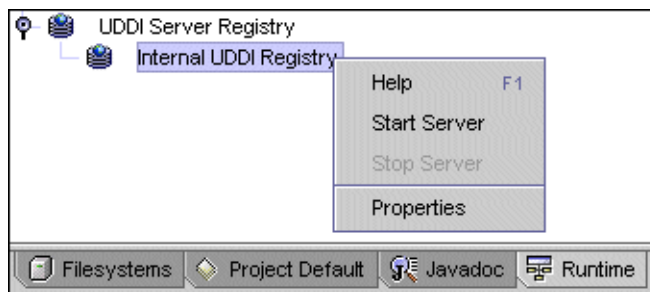


FIGURE 2-26 Start Internal UDDI Registry Server

To stop the internal UDDI registry server:

1. **Expand the UDDI Server Registry node in the Explorer Runtime tabbed pane.**

You should see the Internal UDDI Registry node.

2. **Right-click the Internal UDDI Registry node and select Stop Server.**

The IDE output window displays server startup messages. You might also see messages stating that the IDE is stopping a previous Tomcat server process.

Note – If the internal UDDI registry server is not running, the Stop Server menu item is inactive.

Using the Sample Registry Browser

The Java WSDP provides a *sample registry browser*, which has additional features beyond those included in the Sun ONE Studio 4, Enterprise Edition for Java IDE.

The sample registry browser enables you to delete a service from the internal registry, to add business information beyond name and key, and other actions not provided in the IDE wizards. However, as its name suggests, the sample registry browser is not a complete, full-function tool. For example, it does not enable you to view or delete tModels.

Java WSDP software and sample programs are in the `jwsdp` directory under your IDE home directory. Sample program source is in the `wsdp/samples` directory.

For further information and a downloadable tutorial, see the Java WSDP web site at <http://java.sun.com/webservices/webservicespack.html>.

The purpose of this section is to help you get started setting up and using the sample registry browser, not to describe all of its features.

Note – The sample registry browser runs in its own window, outside the Sun ONE Studio 4, Enterprise Edition for Java IDE. However, its main use is to access the internal registry server. An external registry typically will have its own set of tools for searching and publishing.

Before starting the sample registry browser, use the IDE to start the internal UDDI registry, as explained in “Starting and Stopping the Internal UDDI Registry Server” on page 49.

To start the sample registry browser:

1. **Open a command window and change your current directory to `s1studio_HOME/jwsdp/bin`.**
2. **Type the following command:**
 - `jaxr-browser` if you are on a Microsoft Windows system
 - `./jaxr-browser.sh` if you are in a Solaris environment

The sample registry browser is displayed, as illustrated in FIGURE 2-27.

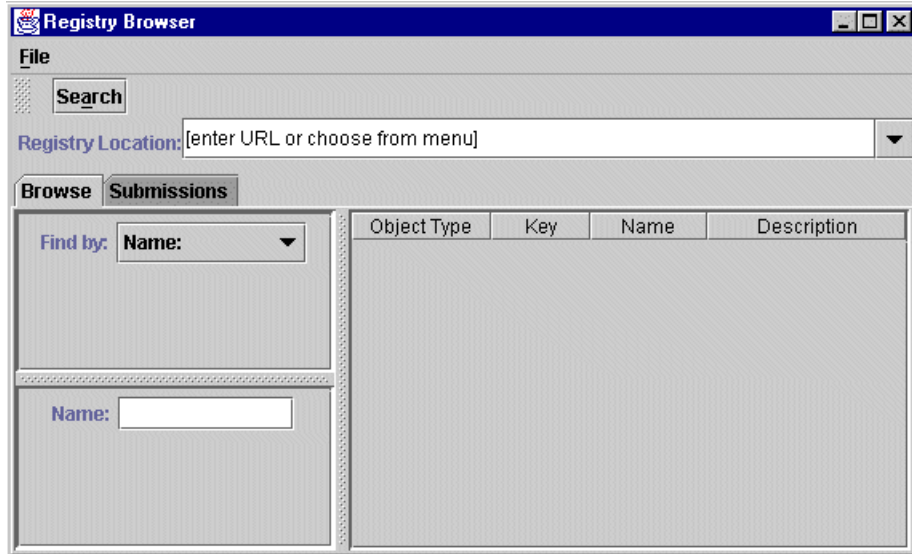


FIGURE 2-27 JWS DP Sample Registry Browser

To configure the sample registry browser for your internal registry:

1. **Select the last URL in the Registry Location list, as illustrated in FIGURE 2-28.**



FIGURE 2-28 JWS DP Sample Registry Browser URL Selection

2. Change the port number in the URL from 8080 to the correct port number for the internal registry Tomcat web server, as illustrated in FIGURE 2-29.

The IDE is configured with 8089 as the default port number for this server.

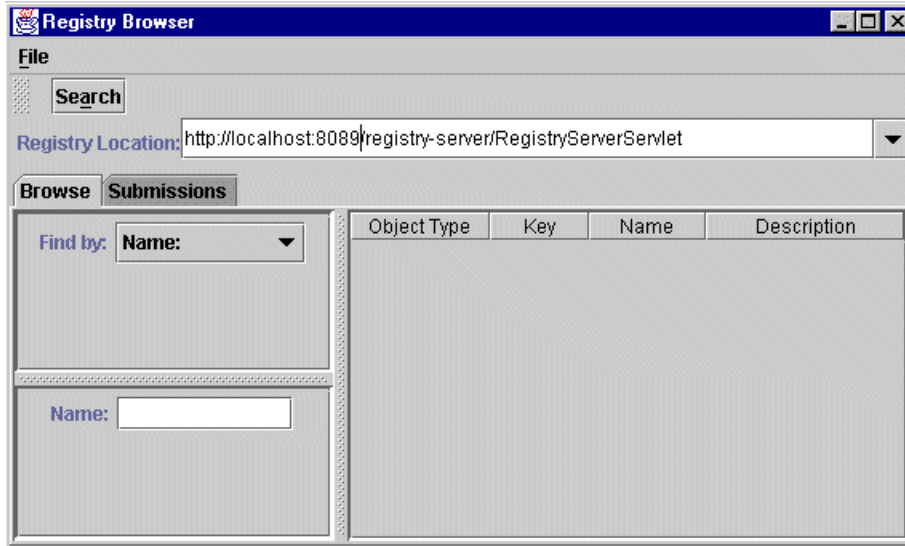


FIGURE 2-29 JWSDP Sample Registry Browser With Internal Registry URL

The sample registry browser main window has, on the left side, a tabbed pane with Browse and Submission tabs. The main window has, on the right side, a pane that displays detail information about the object that is selected in the tabbed pane. The upper left area of the main window has buttons for Search, Submit, and other context-sensitive actions.

An example showing the result of a search by company name is illustrated in FIGURE 2-30.

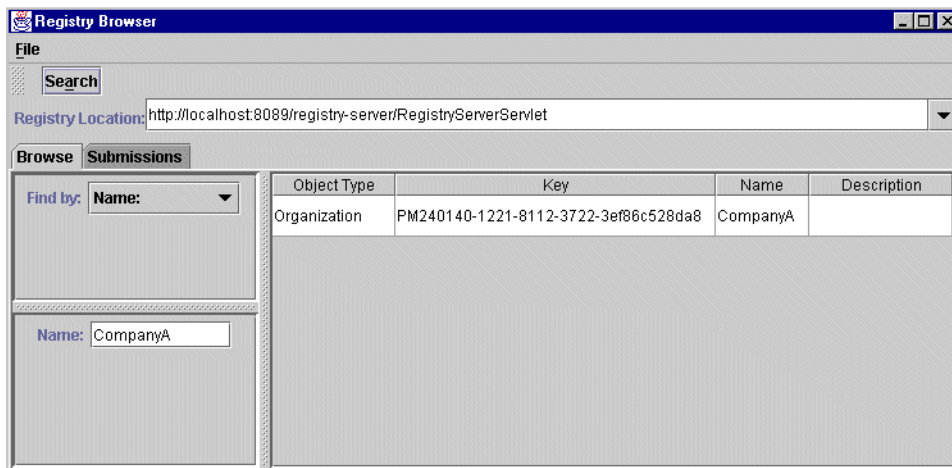


FIGURE 2-30 JWS DP Sample Registry Browser Displaying Selected Business

An example using the Submissions tabbed pane is illustrated in FIGURE 2-31.

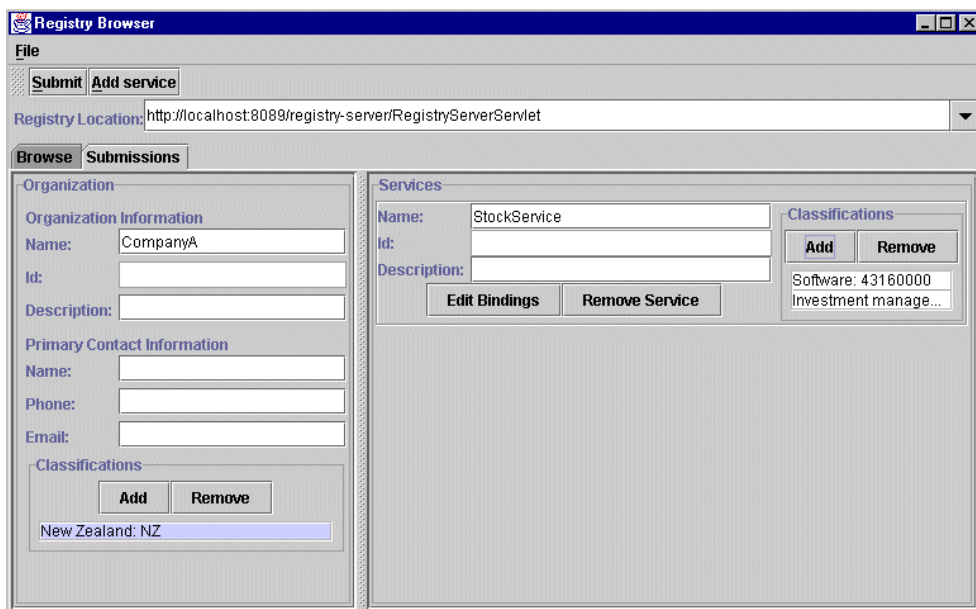


FIGURE 2-31 JWS DP Sample Registry Browser Displaying Submissions Tabbed Pane

You can explore additional features of the sample registry browser on your own, or download and work through the tutorial provided at the Java WSDP web site: <http://java.sun.com/webservices/webservicespack.html>.

Instantiating Objects and Resolving References

When developing an XML operation, you specify which methods the XML operation calls. To call these methods at runtime, the web service requires certain objects. For each method call, it locates or instantiates:

- An instance of the class in which the method is defined
- An instance of each class required by the method as a parameter

To perform this task, the web service maintains a reference to each of these target objects and a definition of how to instantiate an object of the appropriate class should the target object not exist. As you add method calls to an XML operation, default object references and target object definitions are automatically added to the web service. These defaults are usually appropriate and do not need editing.

However, you can manually specify the target of an object reference, and you can edit and create new target object definitions to suit your requirements. You might need to manually resolve object references to enterprise beans that were created outside of the IDE.

For further information on this subject, see Appendix B.

Deployment Descriptors

When you generate runtime classes for your web service, a web module and EJB module deployment descriptor are also generated. When you assemble your web service J2EE application, these deployment descriptors are included in the application. The deployment descriptors are XML files used to configure runtime properties of the application. The J2EE specification defines the format of these descriptors.

You can view the deployment descriptors in the Source Editor at any time during development. You can also edit the deployment descriptors. However, deployment descriptors that you edit are not regenerated if you regenerate runtime classes. Deployment descriptor edits are thereby preserved. However, changes you make to your web service after editing a deployment descriptor will not be propagated to your deployment descriptor. You should, therefore, take care to edit deployment descriptors only at the end of your development cycle.

For further information on this subject, see Appendix C.

Support for Arrays and Collections

This section summarizes the support for input and output data of types `Array` and `Collection`. For further information on this subject, see Chapter 4.

Arrays

The IDE supports arrays as follows:

- Array output is supported for direct method calls and for XML operations.
- Array input is supported for direct method calls.
- Array input is *not* supported for XML operations.

Collections

A collection is a Java class that implements `java.util.Collection`. The IDE supports collections as follows:

- Collection output is supported for direct method calls and for XML operations.
- Collection input is supported for direct method calls on the server and in a client proxy, but *not* in the generated JSP pages.
- Collection input is *not* supported for XML operations.

A collection might contain objects of types defined by user classes that are not otherwise referenced in the web service. For a user-defined object, you must add the serialization data type to the JAX-RPC type registry, so that it is handled properly by the JAX-RPC runtime. The IDE provides a web service property for adding serialization classes.

To add a serialization class:

1. **Right-click the web service node, choose Properties, click the value of the Serialization Classes property, then click the ellipsis (...) button.**

The IDE displays any existing serialization classes and provides Add and Remove buttons, as illustrated in FIGURE 2-32.

2. **Click Add.**

The IDE opens a browser in which you can select one or more classes from any mounted packages. The selected classes should be JavaBean components.

3. Select the desired classes, using ctrl-click to select more than one, then click OK.
The IDE displays the serialization classes.

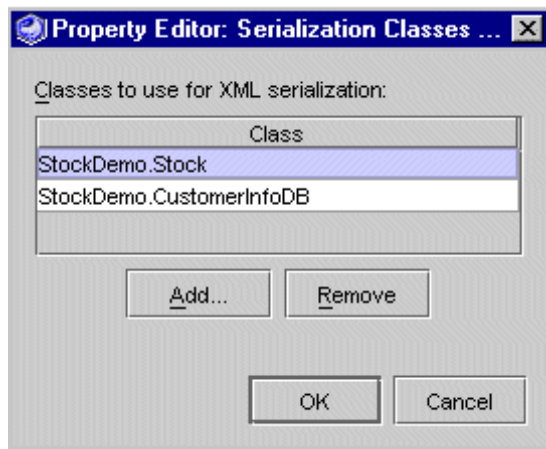


FIGURE 2-32 Serialization Classes Property Editor

Creating a Web Service Client

The Sun ONE Studio 4, Enterprise Edition for Java IDE gives you several ways to create a web service client without writing client code. This chapter describes the IDE tools and procedures that you can use to do each of the following tasks:

- Generate a client from a Sun ONE Studio 4, Enterprise Edition for Java web service
- Generate a client from WSDL
- Generate a client from a UDDI registry entry

If you are creating your own web service and clients in the IDE, you can set the default test client for the service. For further information, see “Setting a Default Test Client for a Web Service” on page 34.

Creating a Client From a Local Web Service


This section assumes that you start with an existing web service developed in the IDE.

You can automatically generate a simple client and client SOAP proxy that is adequate for testing your web service during development. You can then edit the client or replace it entirely by a more sophisticated client that can be used for business purposes.

Creating the Client

You can create a client starting at a package node, a web service node, or the IDE’s main window menu.

To start at a web service node:

1. **Open the New Web Service Client wizard from the Explorer by right-clicking the web service node (the one with the blue sphere icon ) and choosing New Client.**

The New Client dialog box is displayed, as illustrated in FIGURE 3-1.

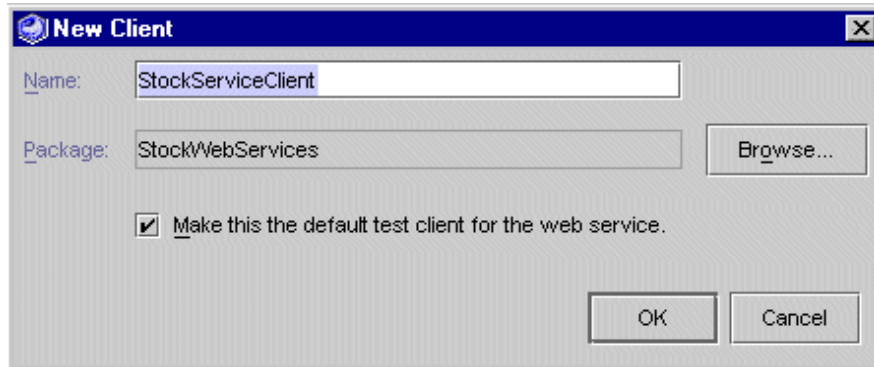


FIGURE 3-1 New Client From Web Service Dialog Box


2. **Ensure that the Package field specifies the desired location in which to create the web service client.**

Click Browse to find and select a package of your choice.

3. **In the Name field, type a name for the new web service client.**

If you enter the name of a web service client that already exists in the package designated in the wizard, the IDE displays a highlighted message: *A web service client with this name already exists in package.*

4. **Click OK.**

Your new web service client is displayed in the Explorer. A client node () appears under the designated package.

To start at a Java package node:

1. **Right-click a Java package node and choose New → Web Services → Web Service Client.**

Alternatively, Choose File → New Template → Web Services → Web Services → Web Service Client from the IDE's main window.

In this case the IDE displays the Web Service Client dialog box, as illustrated in FIGURE 3-2.

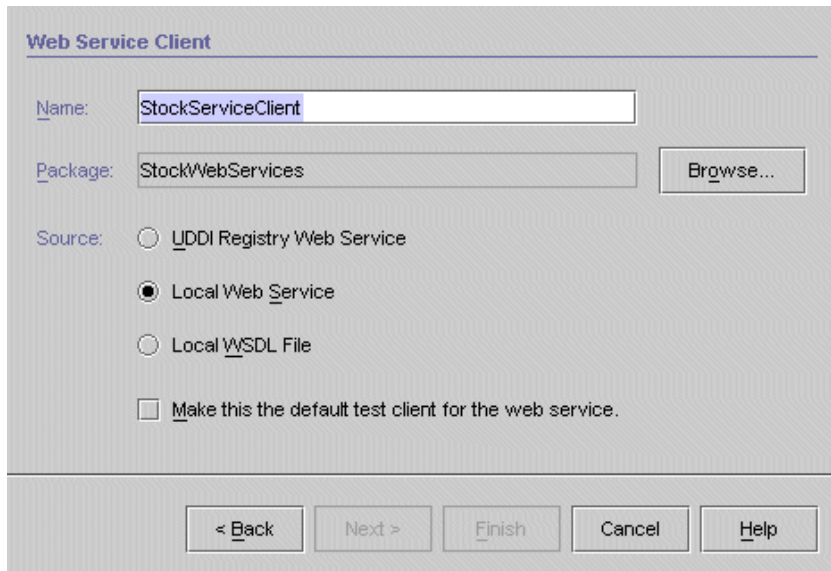



FIGURE 3-2 New Web Service Client Dialog Box

2. Set the desired name and package.
3. Select Local Web Service for Source.
4. Click Next to open a browse window, select the desired web service in the browse window, and click Finish.

Your new web service client is displayed in the Explorer. A client node () appears under the designated package.

Setting the Client's SOAP Runtime Property

The IDE can generate a client that uses the JAX-RPC runtime (the default) the kSOAP runtime, or the Apache SOAP runtime. kSOAP is a SOAP API suitable for the Java 2 Microedition. Apache SOAP clients are supported for compatibility with the previous release of the IDE.

To change the client type:

1. Create the client.
2. Right-click the client node and choose Properties. The Soap Runtime property has the default value JAX-RPC.

3. Click the property value. The choices JAXRPC, Apache SOAP, and kSoap are displayed, as shown in FIGURE 3-3. Choose kSoap.

When you generate the client, a kSOAP proxy is created.

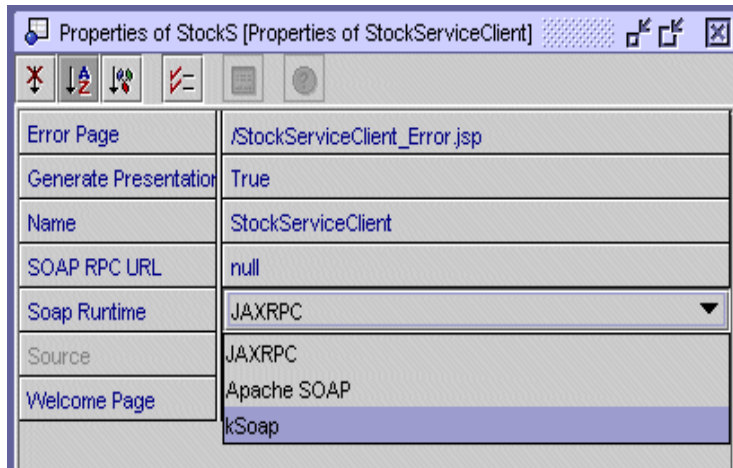


FIGURE 3-3 Client SOAP Runtime Property

Note – The rest of this chapter assumes a JAX-RPC client, unless otherwise specified.

Generating the Client

To generate the client's SOAP proxy and test user interface (HTML and JSP pages), right-click the web service client node in the Explorer, and choose Generate Client Proxy.

During the generation process, the IDE analyzes the web service's WSDL and issues validation warning and error messages if there are problems such as invalid data types (for example, a user object that is not a valid JavaBean component). Validation is particularly important when you are creating a client from someone else's web service, such as a web service accessed through a UDDI registry. IDE validation is based on the JAX-RPC reference implementation.

The IDE puts the generated objects in the same package as the client. A Generated Documents node contains HTML and JSP pages that invoke the SOAP proxy classes.

If the client's SOAP Runtime property is kSoap or Apache SOAP, the IDE generates a single proxy class and puts it in a node named `xxxProxy`, where `xxx` is the client name. If the client's SOAP Runtime property is JAXRPC, the IDE generates many proxy classes and puts them in a node named `xxxGenClient`, where `xxx` is the client name.

The Explorer hierarchy for a JAX-RPC client is illustrated in FIGURE 3-4.

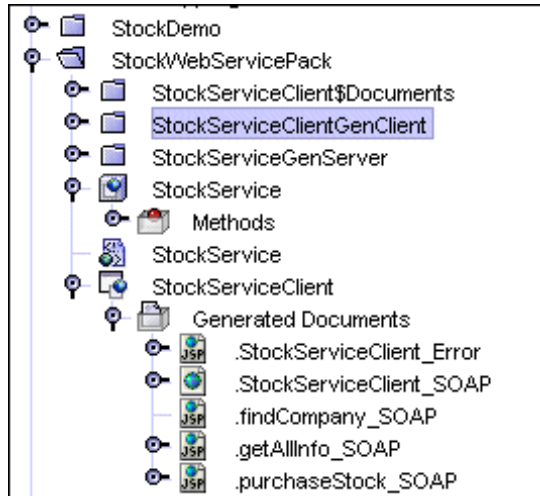


FIGURE 3-4 Client Documents and GenClient Nodes in Explorer Hierarchical Display

The Client HTML Pages and JSP Pages

The following client documents are generated:

- **One JSP page for each method or XML operation.** Each of these JSP pages provides the functionality to execute its corresponding business method or XML operation and format the results in HTML. During development, you can use the JSP pages as a client for testing your business methods and XML operations. Web page designers can customize the generated JSP pages and use them as the basis for more sophisticated, user-friendly clients.
- **One HTML welcome page.** The welcome page displays one HTML form for all generated JSP pages. The HTML form provides the URLs needed to invoke the JSP pages. The forms includes fields for user input when input parameters are required.
- **One HTML error page.**

The names of the welcome page and error page appear as properties of the client node in the IDE.

To view the HTML or JSP code, right-click the node of the desired document and choose Open. Code samples for a welcome page and a JSP page are illustrated in FIGURE 3-5 and FIGURE 3-6.

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>StockService</title>
</head>
<center><h2>StockService</h2></center>
<hr> </hr>
<center><h2>CompanyInfo findCompany(String string_1)</h2></center>
<form method="post" action="findCompany_SOAP.jsp">
  <input type="hidden" name="method" value="findCompany">
  <b>string_1:</b><input type="text" name="string_1"/>
  <br/><br/>
  <input type="submit" value="Invoke"/>
  <input type="reset" value="Reset"/>
</input>
</form>
<hr> </hr>
<center><h2>Collection getAllInfo()</h2></center>
<form method="post" action="getAllInfo_SOAP.jsp">
  <input type="hidden" name="method" value="getAllInfo">
  <input type="submit" value="Invoke"/>
  <input type="reset" value="Reset"/>
</input>
</form>

```

FIGURE 3-5 Client Sample HTML Welcome Page

```

<center><h2>findCompany</h2></center>
<%
// Get our port interface
StockWebServicePack.StockServiceClientGenClient.StockService service = new StockWebServicePack.StockS
StockWebServicePack.StockServiceClientGenClient.StockServiceServantInterface port = service.getStockS

// Get the stub and set it to save the HTTP log.
StockWebServicePack.StockServiceClientGenClient.StockServiceServantInterface_stub stub = (StockWebSer
java.io.ByteArrayOutputStream httpLog = new java.io.ByteArrayOutputStream();
stub.setTransportFactory(new com.sun.xml.rpc.client.http.HttpClientTransportFactory(httpLog));
// Get the end point address and save it for the error page.
String endPointAddress = (String) stub.getProperty(stub.ENDPOINT_ADDRESS_PROPERTY);
request.setAttribute("ENDPOINT_ADDRESS_PROPERTY", endPointAddress);

// Get parameters
java.lang.String p_string_1 = request.getParameter("string_1");

// Call the web service
try {
  StockWebServicePack.StockServiceClientGenClient.CompanyInfo result = port.findCompany(p_string_1);

```

FIGURE 3-6 Client Sample JSP Page

The Client SOAP Proxy

FIGURE 3-7 shows an Explorer view of some of the proxy classes generated for a JAX-RPC client. To view the Java source code, right-click a class or method node and choose OPEN.

The IDE lets you edit the proxy source, but if you regenerate the client in the IDE, your proxy source changes are not preserved.

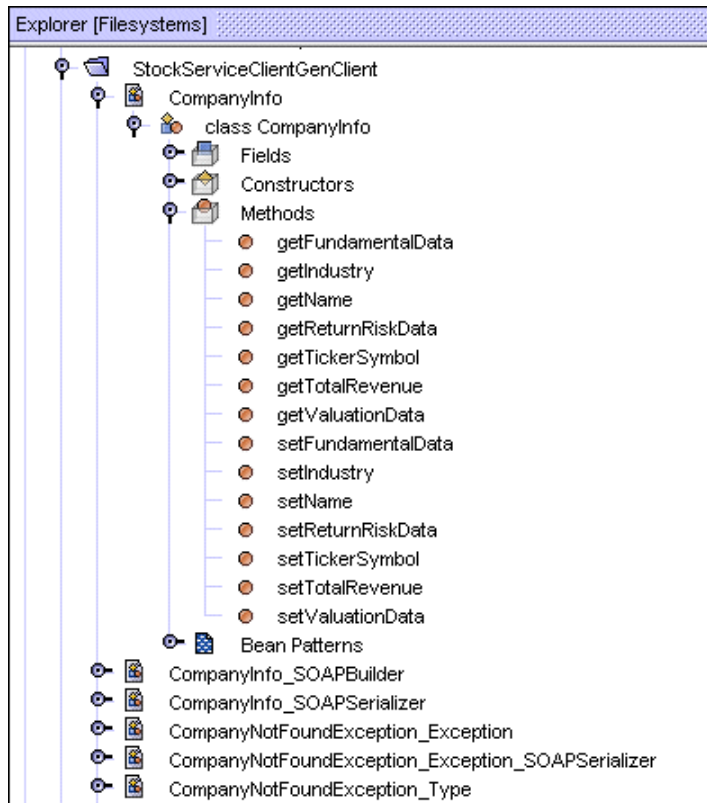


FIGURE 3-7 Client SOAP Proxy GenClient Node

You can use the generated proxy with a Java Swing client or more sophisticated HTML and JSP pages.

Using Your Own Front End Client Components

Clients that use the kSOAP runtime execute on small J2ME devices such as cell phones and PDAs, whose user interfaces are typically constrained by the display limitations. Developers often use Java AWT, WML, and other presentation technologies to create the user interfaces.

If you set a client's SOAP Runtime property to `kSoap`, the IDE does not create a Generated Documents node and does not generate HTML or JSP pages.

If the client's SOAP Runtime property is `JAX-RPC` or `Apache SOAP`, you might want to suppress the generated documents under certain circumstances. For example, if you already have a front-end business client, you might want the IDE to generate only the client SOAP proxy, and to display your own client components. To do this:

1. **Create the client.**
2. **Right-click the client node and choose Properties. The Generate Presentation property has the default value `True`.**
3. **Change the Generate Presentation property value to `False`.**

When you generate the client, only the client SOAP proxy is created.

4. **(Optional) Right-click the client node and choose Add References to → Documents.**

You can add references to documents created outside the IDE, such as HTML and JSP pages created in a web design tool, if you mount the directory containing the documents so they are visible in the IDE Explorer.


The IDE displays the Add Reference to Documents browser. Navigate to a folder that contains your own HTML and JSP pages and click OK in the browser to add the references to your client. The IDE creates a Documents folder containing the references. The Documents folder appears in the Explorer under the client node, on the same hierarchy level as the client's Generated Documents folder.

You can also right-click the client node and add references to Libraries and Classes. The IDE includes them in your test application's WAR file. The libraries and classes can be used to build a Swing test client.



Caution – Nodes within a referenced folder are links that represent actual files or folders. Deleting a reference to file or folder deletes only the reference, leaving the referenced file or folder intact. However, deleting a node inside a referenced folder deletes the file or folder that the node is linked to. Do not delete nodes within a referenced folder unless you want to delete actual files or folders.

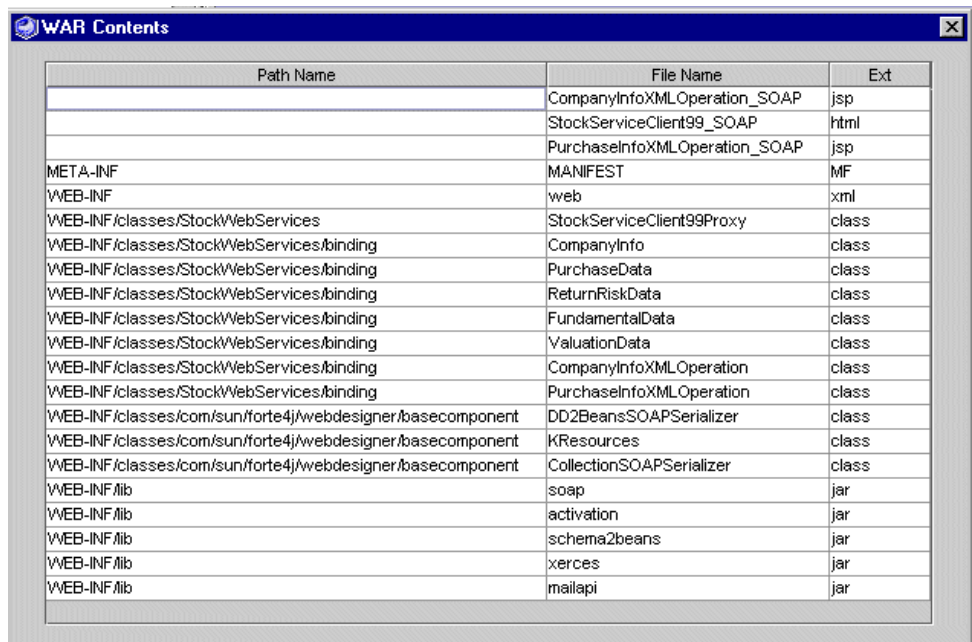
Assembling the Client Application

To assemble the client application, right-click the client node in the Explorer and choose Assemble. The IDE creates a WAR file and displays a node in the Explorer with the WAR icon () and a name of the form *clientName_Client*.

To view the contents of the WAR file, right-click its node in the Explorer and choose View WAR Content. The display is illustrated in FIGURE 3-8.

If you change your web service and regenerate its runtime classes, you can cause the IDE to automatically regenerate the client. To do this, right-click the client in the Explorer and choose Refetch WSDL. This causes the client proxy and documents to be regenerated. (The Refetch WSDL command name refers to an internal WSDL that is based on the service and associated with the client. This WSDL is not displayed in the Explorer.)

You can then reassemble the client. Alternatively, the client will be automatically reassembled the next time you execute it.



| Path Name | File Name | Ext |
|---|-------------------------------|-------|
| | CompanyInfoXMLOperation_SOAP | jsp |
| | StockServiceClient99_SOAP | html |
| | PurchaseInfoXMLOperation_SOAP | jsp |
| META-INF | MANIFEST | MF |
| WEB-INF | web | xml |
| WEB-INF/classes/Stock/WebServices | StockServiceClient99Proxy | class |
| WEB-INF/classes/Stock/WebServices/binding | CompanyInfo | class |
| WEB-INF/classes/Stock/WebServices/binding | PurchaseData | class |
| WEB-INF/classes/Stock/WebServices/binding | ReturnRiskData | class |
| WEB-INF/classes/Stock/WebServices/binding | FundamentalData | class |
| WEB-INF/classes/Stock/WebServices/binding | ValuationData | class |
| WEB-INF/classes/Stock/WebServices/binding | CompanyInfoXMLOperation | class |
| WEB-INF/classes/Stock/WebServices/binding | PurchaseInfoXMLOperation | class |
| WEB-INF/classes/com/sun/forte4j/webdesigner/basecomponent | DD2BeansSOAPSerializer | class |
| WEB-INF/classes/com/sun/forte4j/webdesigner/basecomponent | KResources | class |
| WEB-INF/classes/com/sun/forte4j/webdesigner/basecomponent | CollectionSOAPSerializer | class |
| WEB-INF/lib | soap | jar |
| WEB-INF/lib | activation | jar |
| WEB-INF/lib | schema2beans | jar |
| WEB-INF/lib | xerces | jar |
| WEB-INF/lib | mailapi | jar |

FIGURE 3-8 Client Sample WAR File Display

Deploying and Executing the Client

The web service referenced by your client must be currently deployed.

- To deploy the client, right-click its node in the Explorer and choose Deploy.
- To execute the client, right-click its node in the Explorer and choose Execute.

If you are using a Sun ONE Application Server 7 server, deploy and then execute.

If you are using the built-in Apache Tomcat web server it is not necessary to deploy as a separate step. When you choose Execute, the IDE automatically starts the web server, deploys the client application, and displays the client's welcome page in your default web browser. The result is illustrated in FIGURE 3-9.

Alternatively, you can make the client the default test client, as explained in “Setting a Default Test Client for a Web Service” on page 34. Then you can start the client by right-clicking the J2EE application of your web service and choosing Execute.

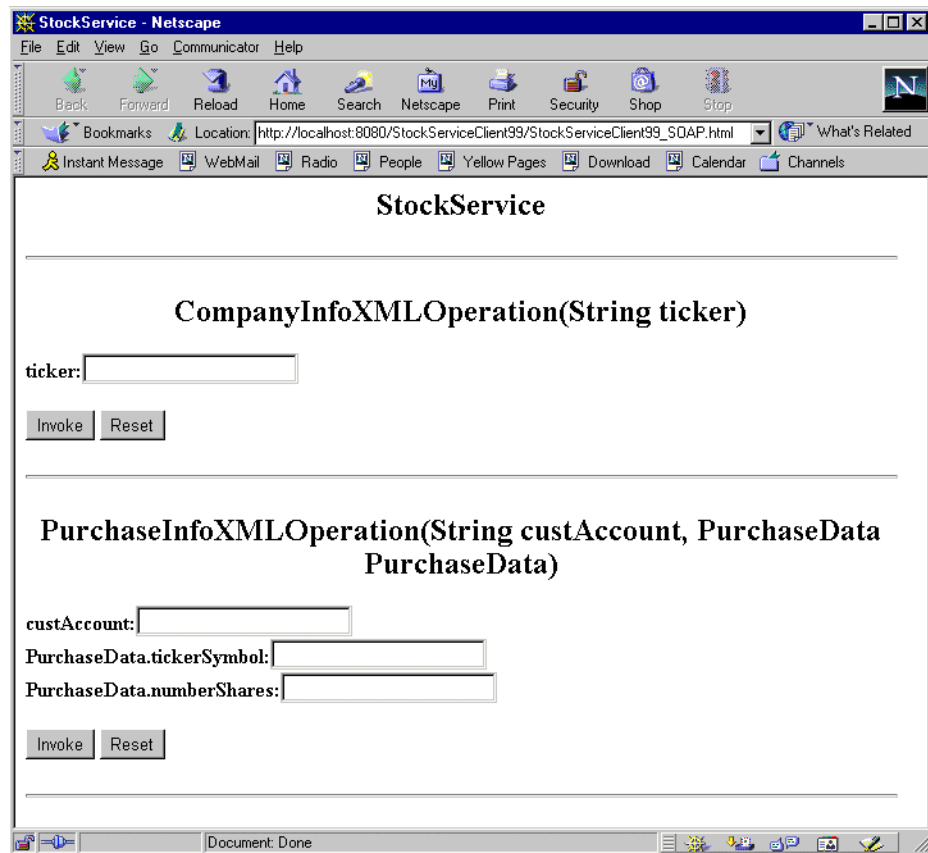


FIGURE 3-9 Client Welcome Page

FIGURE 3-10 illustrates a display from the generated test client of a restaurant review application.




FIGURE 3-10 Client Output Display of Collection of Customer Reviews

Note – The generated test client displays a collection of restaurant reviews as an XML document, including the SOAP envelope. In a production client intended for end users, you would probably eliminate the display of technical information and format the output in a more user-friendly manner.

Creating a Client From WSDL

WSDL provides a portable representation of the external interfaces of a web service. See “Generating WSDL” on page 37 for the procedure to create a WSDL description of your web service. See “WSDL” on page 5 and “Working With UDDI Registries” on page 36 for other information about WSDL and UDDI registries.

WSDL can be useful even if you are not using a UDDI registry. Several people on a project might be developing different kinds of clients, perhaps in different locations or on different schedules. They can be given copies of the WSDL to generate clients. A WSDL file is an XML document and can be put on a shared network drive, sent as an email attachment, or passed around on a disk. The developers must also have network access to a runtime instance of the web service to test their clients.

The following procedure assumes that you have a WSDL file, that the file name extension is `.wsdl`, and that the directory containing the file is mounted in the IDE. The `.wsdl` file appears as a node in the Explorer with a green sphere icon (). If someone gives you a WSDL file that does not satisfy this naming convention, you can rename it or make a copy with a file-name extension of `.wsdl`.

To generate a client from WSDL:

1. **Choose File → New Template → Web Services → Web Services → Web Service Client from the IDE’s main window.**

The IDE displays the Web Service Client dialog box, as illustrated in FIGURE 3-2.

2. **Set the desired name and package, and select Local WSDL File for Source.**
3. **Click Next to open a browse window.**
4. **Select the WSDL node in the browse window.**
5. **Click Finish.**

The IDE creates a client node.

6. **Right-click the client node in the Explorer and choose Generate Client Proxy.**

The IDE creates the client SOAP proxy with its own node. The IDE also creates a set of HTML and JSP pages in the `Generated Documents` node under the client node.

7. **The remaining steps, including assembly, deployment, and execution, are the same as in “Creating a Client From a Local Web Service” on page 57.**

To generate only the client SOAP proxy, without the Generated Documents node:

1. **Right-click the WSDL node in the Explorer, and choose Generate Client Proxy.**

The IDE creates the client SOAP proxy with its own node.

2. **Create your own client component that makes calls to the client SOAP proxy.**

This might be the best approach if you already have fully developed clients that you can easily customize to use the client SOAP proxy, or if you want to use a type of client that is different from the HTML and JSP pages generated by the IDE, such as a Java Swing client.

Creating a Client From a UDDI Registry

Some of the business needs and scenarios that might lead you to create a client from a UDDI registry are described in Chapter 1.

Creating a Client: Planning and Implementation

The work flow consists of these tasks:

- Deciding your criteria for selecting a web service
- Selecting a UDDI registry to search
- Searching the UDDI registry for a web service
- Downloading information about the web service
- Generating the client
- Customizing the client

Task 1 requires advance planning and is part of the process of designing your application. It is not enough to consider the characteristics of the web service. You also have to think about the business that is providing the service, what kind of support the business offers, and how much your use of the service will cost. The analysis is simplified if you are using a private registry and the service provider is a part of your own project or enterprise, or perhaps a business partner collaborating on your project.

Creating a Client: Procedure

This section describes how to use the IDE to select a registry, search the registry for a service, and generate a client that can access a runtime instance of the service.

1. Right-click a Java package node and choose New → Web Services → Web Service Client.

Alternatively, Choose File → New Template → Web Services → Web Services → Web Service Client from the IDE's main window.

The IDE displays the Web Service Client dialog box, as illustrated in FIGURE 3-11.

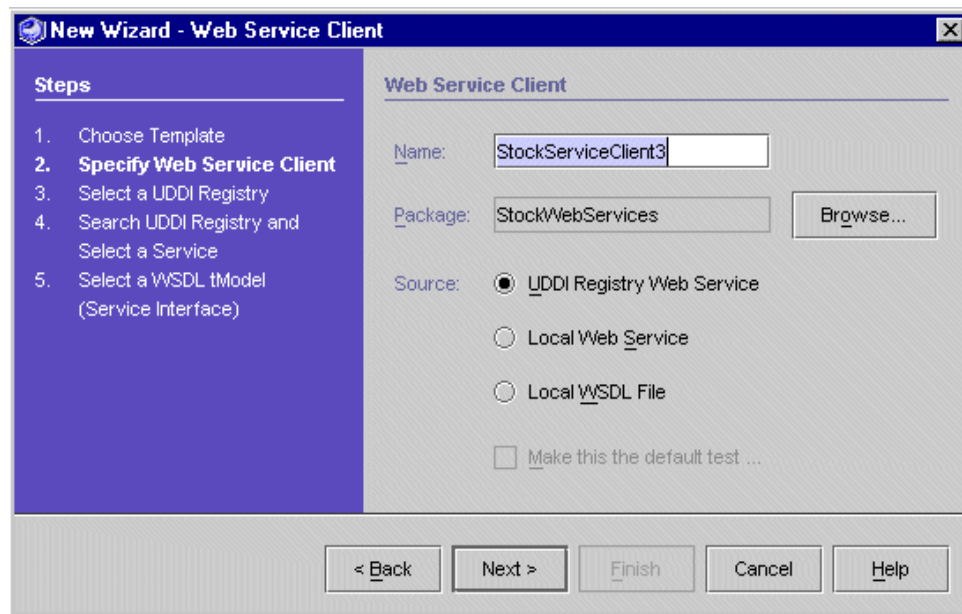


FIGURE 3-11 New Web Service Client Wizard

2. Set the desired name and package, and select the UDDI Registry Web Service radio button for Source.
3. Click Next to display the Select a UDDI Registry dialog box, as illustrated in FIGURE 3-12.

The dialog box displays a list of UDDI registries. Select the desired registry or click Edit to edit registry information or change the default registry (see “Editing Registries Information in the IDE” on page 41).

Note – The initial default registry in the IDE is the internal UDDI registry that is bundled with the IDE. For further information about this registry, see “The Internal UDDI Registry” on page 49.

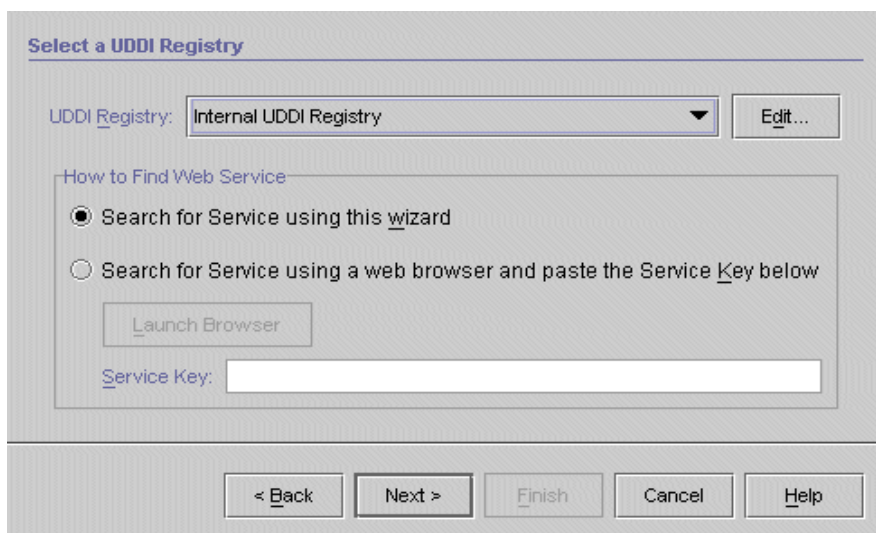


FIGURE 3-12 UDDI Registry Selection Dialog Box

Most UDDI registries provide browser-based tools for registry searches and for adding, editing, and deleting registry entries. The Browser Tools URL for a registry is one of the properties that you can edit in the IDE (see “Editing Registries Information in the IDE” on page 41).

To search a selected registry with its own browser tools:

a. Select the “Search for service using a web browser” radio button.

The Launch Browser button becomes active.

If you already have the web service’s registry key from a previous search, paste the value into the Service Key field and click Next. The IDE displays the final wizard step in which you can click Finish to create the client, bypassing intermediate UDDI registry search steps.

b. Click Launch Browser.

The IDE opens your default web browser at the Browser Tools URL web page.

c. Use the browser tools to locate the web service that you want to access with your client.

d. Copy the web service registry key and paste it into the Service Key field in the Select a UDDI Registry dialog box.

e. Click Next.

The IDE displays the final wizard step in which you can click Finish to create the client, bypassing intermediate UDDI registry search steps.

The remainder of this procedure assumes that you do your search through the IDE wizard. This is the default radio button under “How to Find Web Service.”

4. Select a registry from the UDDI Registry list and click Next to display the UDDI Registry Search dialog box, as illustrated in FIGURE 3-13.

Search UDDI Registry and Select a Service

Search For: Containing:

Search Results

Matching Businesses (0):

| Name | Description |
|------|-------------|
|------|-------------|

Services with WSDL for Selected Business (0):

| Name | Description |
|------|-------------|
|------|-------------|

Enter query text and press Search

FIGURE 3-13 UDDI Registry Search Dialog Box

You can search for businesses whose names contain a string that you specify. You can also base your search on a type other than business name. The list of search types is shown in FIGURE 3-14.

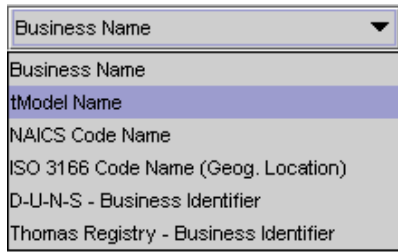


FIGURE 3-14 UDDI Registry Search Types

You can search on tModel name, NAICS code name, or ISO 3166 code name.

You can also search for a D-U-N-S business identifier or Thomas Registry business identifier, in which case the search is for a value “Equal To” rather than “Containing.” You might know the identifier from a previous search. If you are using a private UDDI registry, someone in your project or enterprise might give you the identifier.

Note – If the search type requires an exact value, the label in the UDDI Registry Search dialog box (FIGURE 3-13) changes from Containing to Equal To.

For NAICS or ISO 3166 code name searches, you can enter either a name or a code in the Containing field. For example:

- Select ISO 3166 and specify a search string of `United Kingdom` or `GB`.
- Select NAICS and specify a search string of `Other Financial Vehicles` or `52599`.

A search on tModel Name returns all businesses that contain services that are based on the tModels matching the search string.

A search on NAICS Code Name or ISO 3166 Code Name returns all businesses that have categories matching the search string.

Note – When you specify a string in the Containing field, the IDE wraps your string in a wildcard: `%yourstring%`. You can optionally insert more wildcards. For example, you can search for a business whose name contains the string `bio%tech`, where `%` matches any string. String searches are case-insensitive.

The remainder of this procedure assumes that you search on business name.

5. Specify a string in the Containing field and click Search.

The Matching Businesses table displays all businesses in the registry whose names contain your search string, as illustrated in FIGURE 3-15. Each business is listed with its name and an optional description. The number of matching businesses is also displayed.

You can select a business from the table and proceed to Step 7 of this procedure.

Depending on your search string and the size of the registry, your search might return a large number of businesses. This is most likely to occur with public registries, which are expected eventually to grow to tens of thousands or even millions of entries. If the list returned by your search is too large, you can enter a more carefully chosen search string or you can use the registry browser tools to perform a more advanced search.

Note – In a realistic scenario, a development team or planners probably have a pretty good idea what businesses they are interested in. The main problem finding a business in the registry might be getting the correct spelling or (in the case of a large enterprise with many divisions, departments, and projects) finding out which of its names is used in the registry. Registry search features can help, but in many cases this information is managed by the appropriate project leaders.

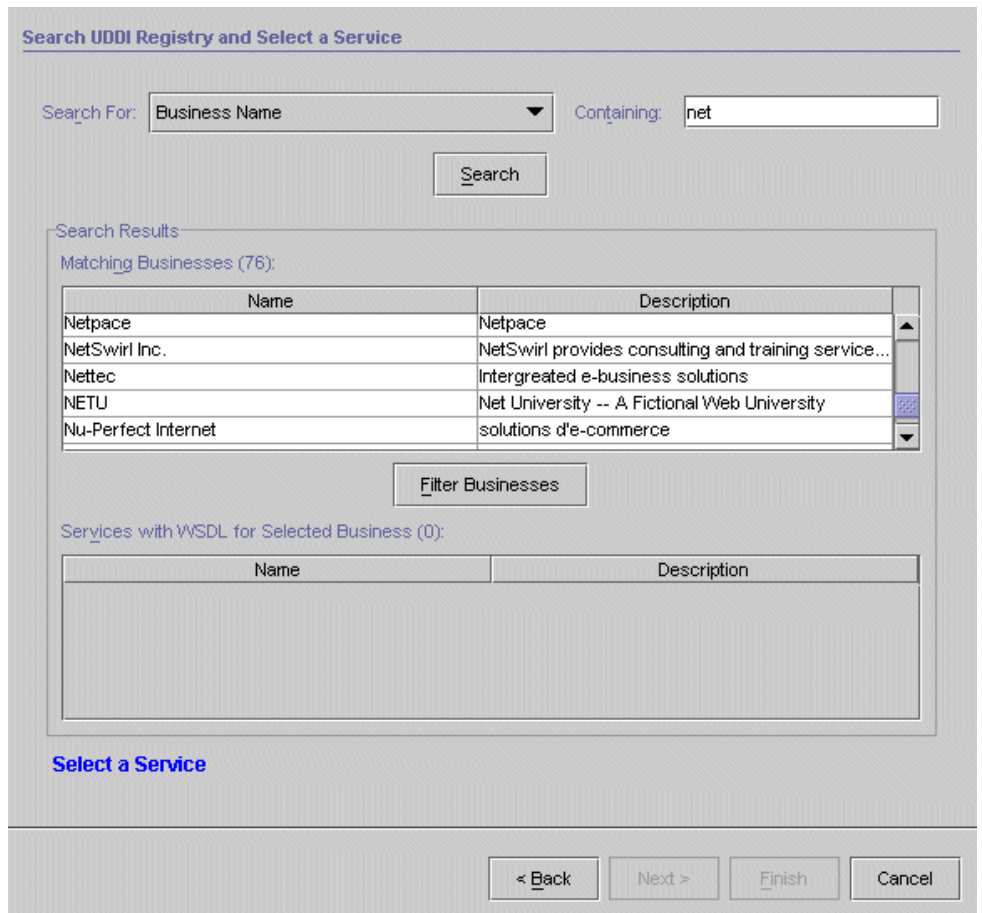


FIGURE 3-15 UDDI Registry Search Dialog Box With Matching Businesses

6. Click Filter Businesses to further refine your search.

There might be services in the registry that have no WSDL tModel entries. Since you can't create clients for those services, the IDE enables you to filter them out of the Matching Businesses table. When you click Filter Businesses, the IDE searches the UDDI registry for services associated with the businesses in your result set. The IDE checks the services for tModels that have an OverviewDoc with an OverviewURL field that begins with "http://" and ends with either "wsdl" or "asmx". Businesses that fail this test are filtered out of the Matching Businesses table.

The IDE displays a progress monitor window with a Cancel button, as illustrated in FIGURE 3-16. The filtering process can take some time. If it takes too long, you can click Cancel and start over with a more refined search string.

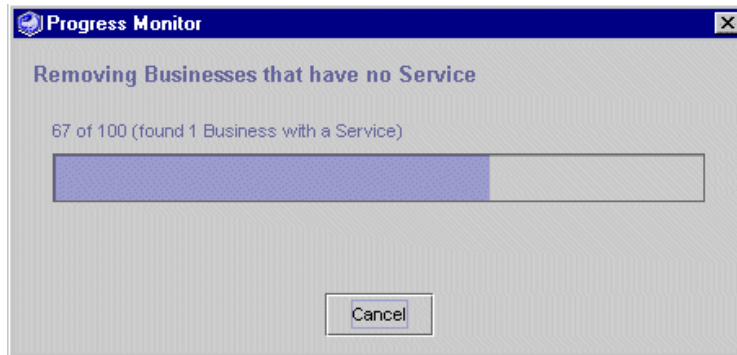


FIGURE 3-16 UDDI Registry Filter Business Progress Monitor

7. Select a business from the Search Results list.

The IDE displays services associated with the selected business, as illustrated in FIGURE 3-17, and activates the Next button. Only services with WSDL tModel entries are displayed.

Note – The registry might not have enough information about a business or service to guide your decision. Public registries are still in their infancy and can be viewed as elements of a broader web service infrastructure in which usage patterns and practices will emerge over time. A private registry gives the registry owner more control of the criteria for publishing services. For example, a private registry might have standards for documenting entries and requirements about the runtime availability of services that are published to the registry. There might be a distinction between services available for development and testing, and services available for use in production applications.

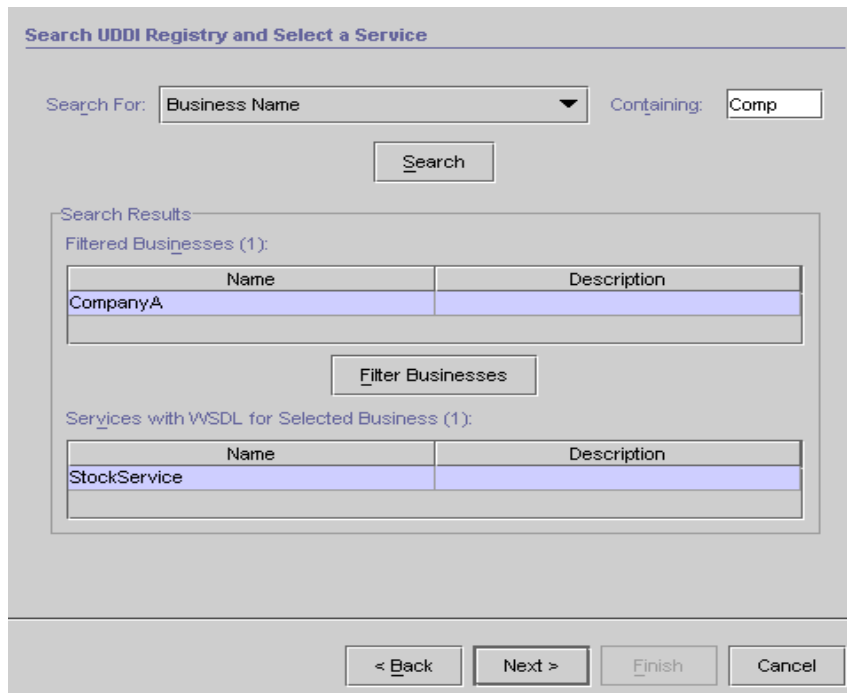


FIGURE 3-17 UDDI Registry Select Service

8. Select a service and click Next to display the Service Interface dialog box, as illustrated in FIGURE 3-18.

The dialog box shows detail information about the service and the tModels and WSDL that it references.

For the service, you see the name, network endpoint, and key. The network endpoint is the URL that a client can use to access the runtime service instance.

For each WSDL tModel, you see name and overview URL. The overview URL is the locator for the WSDL that the IDE uses to create your client.

Note – A web service provider might publish several tModels for a given service instance. For example, one tModel might allow full use of the service methods and another tModel might provide access to a subset of the service methods. This is reflected in clients generated from the two different tModels.

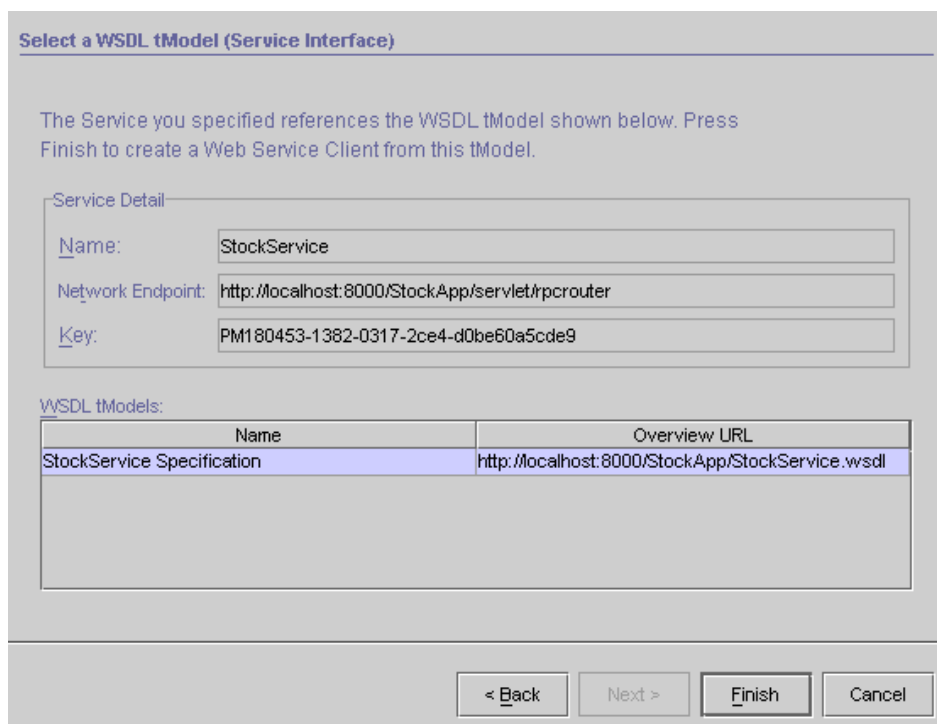


FIGURE 3-18 UDDI Registry Display Service Details and tModel

9. Select a WSDL tModel and click Finish to complete the client creation procedure.

The IDE creates your web service client. It appears in the Explorer under the designated package as a new client node.

10. The remaining steps, including assembly, deployment, and execution, are the same as in “Creating a Client From a Local Web Service” on page 57.

The Service Endpoint URL

For your client to access a service instance at runtime, the client proxy must have the endpoint URL of the service. The IDE can set a default URL in the proxy, or the URL can be passed to the proxy at runtime in its `serviceURL` parameter, which overrides the default.

The client JSP pages generated by the IDE do not pass the URL at runtime, but assume that the client proxy has a default value for `serviceURL`. The default URL comes from the WSDL that is used to generate the client proxy.

As explained earlier in this chapter, you can generate a client from your own web service, from a WSDL file displayed in the IDE, or from a service that you find in a UDDI registry. In all of these cases, the IDE creates a hidden file containing WSDL that describes the web service, and the IDE creates the client based on the WSDL. The hidden file, which has a file name extension of `.wsdx`, cannot be displayed or modified in the IDE. However, you can find the hidden file in the same directory as the client, make a copy of the file with a file name extension of `.wsdl`, and pass it around so that other developers can generate clients in their instances of the IDE.

Note – If you edit the WSDL in the hidden file outside the IDE, your changes will be lost the next time you generate the client.

The default URL in the hidden WSDL file is derived as follows:

- If you generate a client from your own web service, the IDE sets the service URL equal to the SOAP RPC URL property of your service.
- If you explicitly generate WSDL from your own web service, the WSDL is visible in the Explorer, and the IDE sets the service URL equal to the SOAP RPC URL property of your service. If someone gives you WSDL directly as an XML document, it might or might not contain the service URL. The WSDL is visible in the Explorer if the file containing it has a name with a `.wsdl` extension and the directory containing it is mounted in the IDE.
- If you select a visible WSDL file in the Explorer and generate a client, the IDE copies the WSDL to a hidden file with a `.wsdx` extension, and uses the latter to generate the client. If the WSDL does not contain a value for service URL, the proxy is generated without a default value for the service URL.
- If you generate a client from a UDDI registry, the WSDL is downloaded to your IDE and stored as a hidden file with a `.wsdx` file extension. If the UDDI service entry provides a URL, the IDE puts that URL into the downloaded WSDL, regardless of whether a URL is coded in the original WSDL.
- If the generated proxy does not have a default value because none was provided in the scenarios described above, the service URL must be passed to the proxy at runtime.

Developing XML Operations

XML operations provide an efficient way to create a web service interface for existing business components that were not designed for web service access.

For example, suppose you are creating a web service to enable a customer to place an order, but the business component has methods to check inventory, check customer credit, ship an order, and process billing information. You can combine those methods into a single XML operation to place an order.

XML operations are, along with direct method calls, the building blocks of your web service. Each XML operation defines a response to a particular client request.

This chapter explains how you create and edit XML operations. It also provides a description of the tools you use for this job. For conceptual information on XML operations, see “XML Operations” on page 13.

Overview of Tools

The tools you use to develop XML operations are available to you from the Explorer and Source Editor.

You perform the initial step of creating an XML operation definition in the Explorer by selecting a node, to indicate where you want the operation created and then opening the New From Template wizard. This procedure is the same as the procedure for creating a class or any other object in the Explorer. (See “Creating an XML Operation” on page 87 for explanation of how to do this.)

To further develop your XML operation, you access tools from either the Explorer or the Source Editor. The same commands are available from either window. The Source Editor is generally more convenient to use because it lets you issue commands and view results of the commands in the same window. FIGURE 4-2 and FIGURE 4-1 show XML operations displayed in the Source Editor.

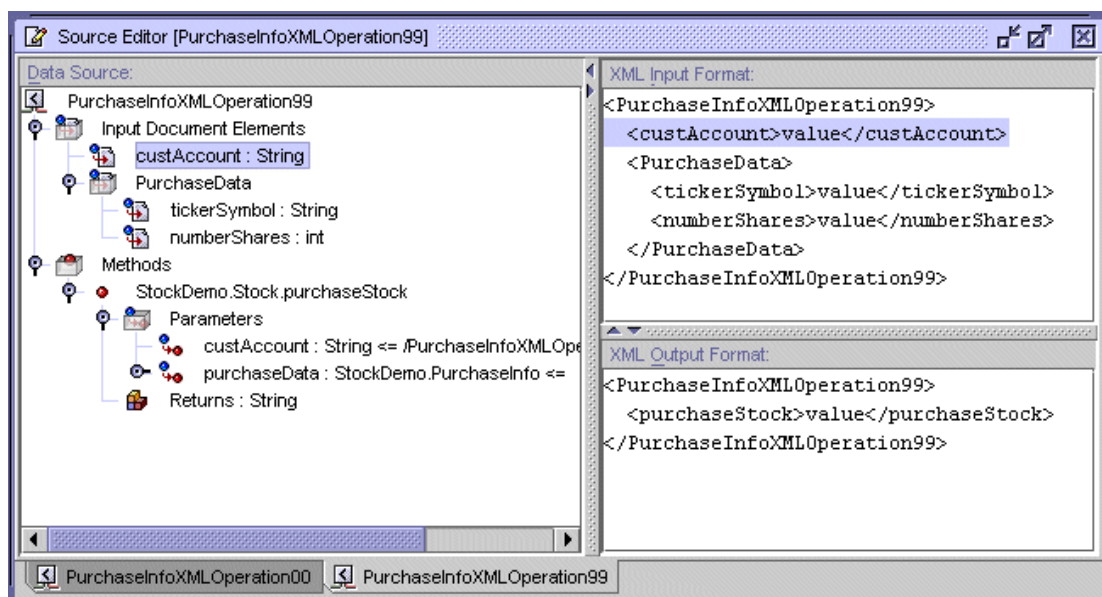


FIGURE 4-1 XML Operation Source Editor, Showing Complex Input

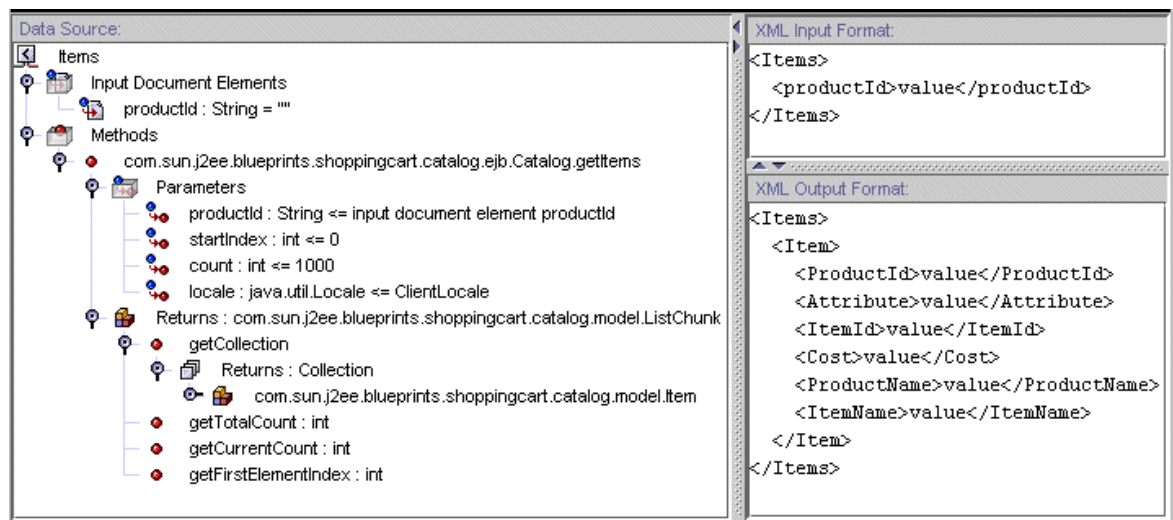


FIGURE 4-2 XML Operation Source Editor, Showing Complex Output

The Source Editor displays an XML operation in three panes:

- **Data Source pane.** Use this pane to view and issue editing commands on the XML operation.
- **XML Input Format pane.** This view-only pane displays the format of the XML input document.
- **XML Output Format pane.** This view-only pane displays the format of the XML output document.

The Data Source Pane

The Data Source pane displays an XML operation in a tree view. The nodes of the structure represent XML input document elements, methods called by the XML operation, parameters to these methods, return values of the methods, and organizational nodes. Each type of node has its own commands and properties.

You can edit an XML operation by:

- Selecting a node and then choosing a menu command
- Double-clicking a node to display its property sheet and then editing its properties

At the top level, the Data Source pane contains two organizational nodes: the `Input Document Elements` node and the `Methods` node.

Input Document Elements Node

The XML input document contains the data that specifies the client request. It is represented as the `Input Document Elements` node. By expanding this node, you can browse and edit the XML input document elements. These elements are represented as subnodes. FIGURE 4-3 shows a cropped view of the Source Editor with the `Input Document Elements` node expanded.

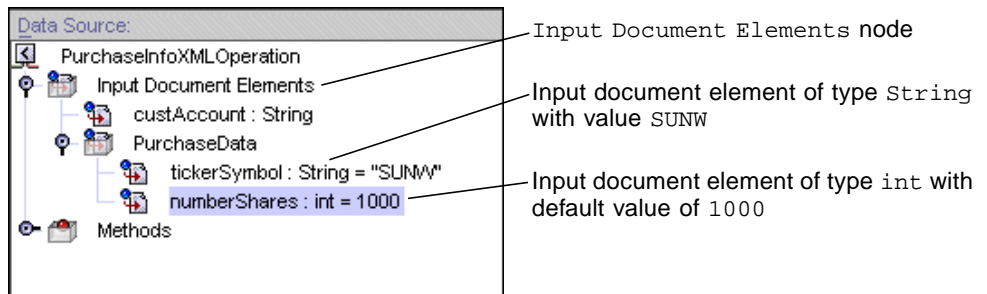


FIGURE 4-3 The Input Document Elements Node

From these nodes, you can perform the following types of editing operations on the XML input document:

- Add elements (see “Adding an Input Document Element” on page 92)
- Delete elements (see “Deleting a Method or Input Document Element” on page 97)
- Rename elements (see “Renaming an Input Document Element” on page 94)
- Rearrange elements (see “Reordering a Method or Input Document Element” on page 97)
- Provide a default value for an element (see “Giving an Input Document Element a Default Value” on page 95)
- Make an element permanent (see “Making an Input Document Element Permanent” on page 96)

Methods Node

From the Methods node, you can:

- Specify the methods called by the XML operation, including overloaded methods
- Specify which data the XML operation retrieves by adding calls to methods on your data sources
- Provide values for method parameters by mapping method parameters to sources
- Retrieve more or less detailed data by expanding or collapsing classes returned by the XML operation’s method calls
- Trim the amount of data sent to the client by selectively excluding fields of the returned classes from the XML output document
- Make an object returned by a method available to other XML operations in the web service
- Cast a method return value
- Display and select inherited methods

Executing Methods and Returning Data

XML operations execute methods on other runtime objects. You specify which methods your XML operation executes by adding methods calls to the `Methods` node. By adding method calls, you can program your XML operation to return data or perform other types of processing.

You can also delete and rearrange the order in which methods are called. The XML operation executes the methods in order from top to bottom.

Adding, reordering, or deleting methods affects the XML output document by adding, reordering, and deleting the elements corresponding to the return values. Such changes are displayed in the XML output pane as you make them. For more information on these topics, see “Adding a Method to an XML Operation” on page 91, “Reordering a Method or Input Document Element” on page 97, and “Deleting a Method or Input Document Element” on page 97.

Providing Parameter Values

If a method takes parameters, the parameters are listed under the `Parameters` node for that method. By default, the XML operation obtains a value for each parameter by mapping elements of the XML input document to like-named parameters.

You can, however, remap XML input elements to method parameters in any way you want. You might need to do this if two methods in your XML operation take like-named parameters. In such a situation, the XML operation by default maps both parameters to the same XML input element. If this is not appropriate, you can create a new input element and remap one of your parameters to it.

You can also map parameters to types of sources other than input elements. For example, you can map a parameter to:

- A fixed value
- The return value of another method call in your XML operation
- A target object that you define in your web service and that is instantiated on demand
- An object returned by a method call in another XML operation and which you have explicitly shared
- A system shared object

For a description of how to map a method parameter to a source, see “Mapping a Method Parameter to a Source” on page 98. For information on target objects, see “Instantiating Objects and Resolving References” on page 54. For information on system shared objects, see “System Shared Objects” on page 102.

Retrieving More or Less Data

If you add a method call that returns an object (or an array or collection of objects), the object’s class type is shown as a subnode to the method. If this class contains methods that begin with the string `get`, the methods are shown as subnodes to the class. By default, an object returned by any of these “getter” methods is also shown as a class node, but without subnodes representing its methods.

You can, however, choose to expand such a class node. Expanding the class node adds nodes for all of the getter methods in the class and likewise adds elements corresponding to the return values of these methods to the XML output.

Conversely, you can collapse a class so that its getter methods are not displayed in the Data Source pane. You can also individually delete methods. In both of these cases, the methods are not called when the XML operation is executed. The elements of the XML output corresponding to these methods are automatically removed.

Whenever you expand or collapse a class, corresponding changes to the XML output are displayed in the XML Output Format pane as you make them. For more information on this topic, see “Expanding a Class” on page 102.

Trimming the Data Returned to the Client

You might find that a returned class provides some data that you don’t want to include in the XML output. For example, a customer account class might have an account ID field and a corresponding getter method that is used only for internal purposes. In such a situation, you can selectively choose to exclude the element corresponding to this method from the XML output.

Excluding an element from the XML output does not affect the methods called by the XML operation or the data set returned by these methods to the XML operation. The exclusion affects only the data set returned to the client by way of the XML output document. By excluding unnecessary elements, the data passed between application containers is minimized, optimizing performance. For more information on this topic, see “Excluding an Element From the XML Output” on page 100.

Development Work Flow

The following steps outline the work flow for developing an XML operation.

1. Create an XML operation.

This procedure results in an XML operation that has:

- One method call
- A default XML input document based on the method parameters
- A default XML output document based on the return value of the method
- A default mapping of XML input elements to method parameters

2. (Optional) Edit the XML operation by performing some or all of these procedures:

- Add or delete method calls.
- Add, delete, or rename input document elements.
- Map method parameters to sources.
- Share returned objects.
- Expand or collapse returned classes.
- Rename or exclude elements from the XML output document.

3. Include the XML operation in a web service.

If you don't already have a web service, you must create one. For more information, see "Creating a JAX-RPC Web Service from Java Methods" on page 18 and "Adding References to Methods, XML Operations, and JAR Files" on page 20.

4. Test the XML operation in the web service.

For more information on this topic, see "Creating a Test Client" on page 34.

5. Edit the XML operation, regenerate the web service's runtime classes, and test until satisfied.

Creating XML Operations

You can create XML operations individually or generate a group of XML operations based on an enterprise bean.

Creating an XML Operation

To create an XML operation:

- 1. In the Explorer, right-click the folder in which you want to create the XML operation and choose New → Web Services → XML Operation.**

The New from Template XML Operation dialog box is displayed, as illustrated in FIGURE 4-4.

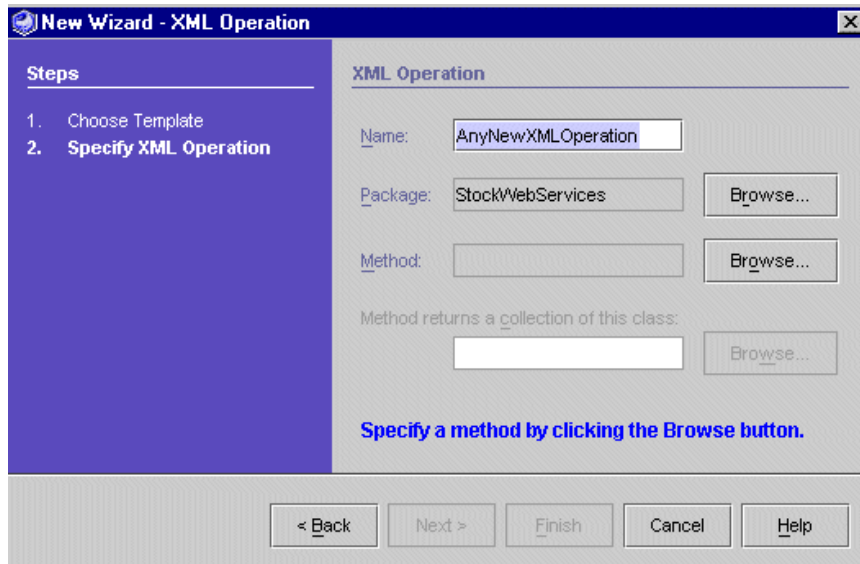


FIGURE 4-4 New XML Operation Dialog Box

2. In the Name field, type a name for the XML operation.
3. Ensure that the Package field specifies the correct location in which to create the XML operation.
4. Click the Browse button next to the Method field.

The Select Method dialog box is displayed, as illustrated in FIGURE 4-5.

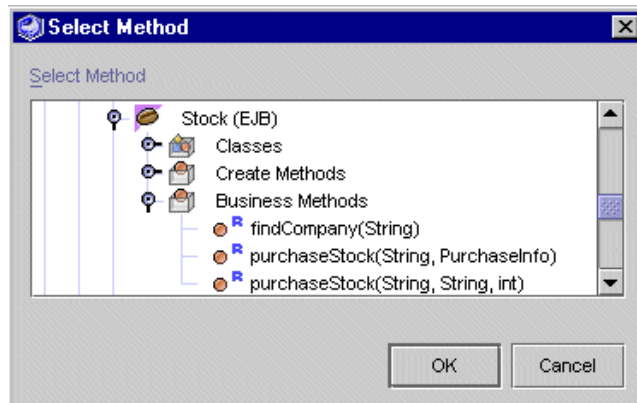



FIGURE 4-5 Select Method Dialog Box

5. Navigate to a method you want to include in the XML operation and click OK.

You can add methods from a class, a bean, an interface, or an EJB component.

You can include methods defined in an EJB's home and remote interfaces, but not methods defined only in the EJB local interface.

If you are adding an EJB method, be sure to browse to the logical EJB node (the node with the bean icon ), not the node that represents the EJB bean class or the home or remote interface. By adding the method from the logical EJB node, you provide the runtime information needed to call the method. When you create an XML operation, you include one method in it. If your XML operation requires additional methods, you can add them later. For information on adding additional methods to an XML operation, see "Adding a Method to an XML Operation" on page 91.

6. If the method you have selected returns an array or collection, select the class, parent class, or interface of the objects contained in it.

Click the Browse button in the New XML Operation dialog box, next to the field labeled Method Returns a Collection of This Class. A file browser opens. Use the file browser to select the class or interface.

7. Click Finish.

The XML operation is created and displayed in the Source Editor ready for editing (as shown in FIGURE 4-2). For information on editing your XML operation, see "Editing an XML Operation" on page 90.

Generating XML Operations From an Enterprise Bean

As an alternative to creating XML operations individually, you can generate a group of XML operations based on an enterprise bean, an EJB module, or a package containing one or more enterprise beans. Doing so generates one XML operation for each method on the home and remote interfaces of each enterprise bean. References to the generated XML operations are automatically added to your web service.

To generate XML operations from enterprise beans, you must have already created a web service. For information on creating a new web service, see "Creating a JAX-RPC Web Service from Java Methods" on page 18.

To generate XML operations from an enterprise bean:

1. Right-click your web service and choose Generate Operations from EJB.

A file browser is displayed.

2. Browse to an enterprise bean, an EJB module, or package containing an enterprise bean and click OK.

3. Click Finish.

4. Specify the class, parent class, or interface of objects contained in any array or collection returned by methods in the enterprise beans.

If any method returns an array or collection, the Collection of What? dialog box is displayed, as illustrated in FIGURE 4-6.

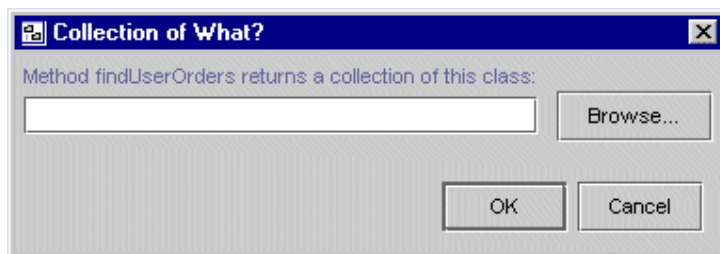


FIGURE 4-6 Collection of What? Dialog Box

The name of the method that returns the collection is indicated in the dialog box.

To specify the object type:

a. Click the **Browse** button.

A file browser is displayed.

b. **Navigate to the class, parent class, or interface of objects contained in the array or collection and click OK.**

Note – This dialog box is automatically displayed once for each method that returns an array or collection.

The XML operations are generated and references to them are added to your web service. You can now delete the XML operations you don't need and edit the others according to your requirements.

Editing an XML Operation

This section describes the ways you can edit an XML operation.

Adding a Method to an XML Operation

To add a method to an XML operation:

1. **Open your XML operation in the Source Editor.**
2. **In the Data Source pane, right-click the `Methods` node and choose `Add Method`.**

The Add Method dialog box is displayed, as illustrated in FIGURE 4-7.

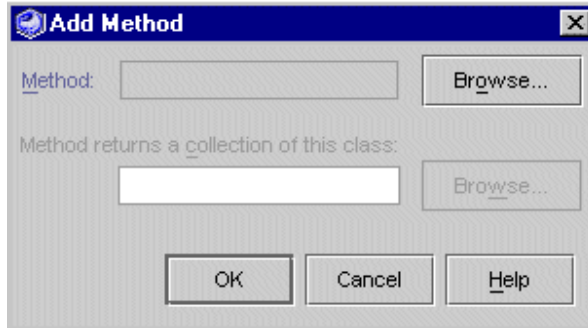


FIGURE 4-7 Add Method to XML Operation Dialog Box

3. **Click `Browse` to select a method.**

The Select Method dialog box is displayed, as illustrated in FIGURE 4-8.

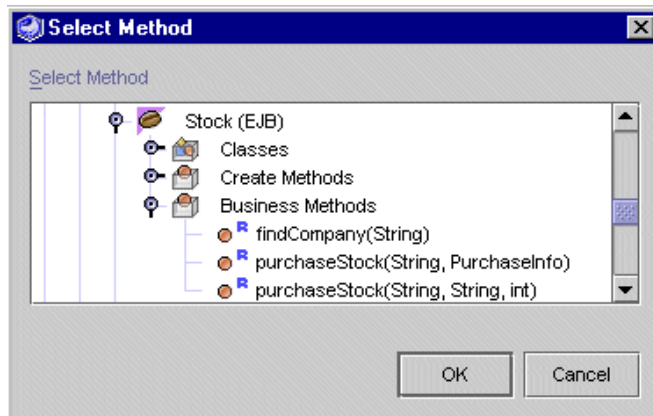



FIGURE 4-8 Select Method Dialog Box

4. Navigate to a method you want to include in the XML operation and click OK.

You can add methods from a class, a bean, an interface, or an EJB component.

You can include methods defined in an EJB's home and remote interfaces, but not methods defined only in the EJB local interface.

If you are adding an EJB method, be sure to browse to the logical EJB node (the node with the bean icon ), not the node that represents the EJB bean class or the home or remote interface. By adding the method from the logical EJB node, you provide the runtime information needed to call the method.

5. If the method you have selected returns an array or collection, select the class, parent class, or interface of the objects contained in it.

Click the Browse button in the Add Method dialog box, next to the field labeled Method Returns a Collection of This Class. A file browser opens. Use the file browser to select the class or interface.

6. Click OK.

This action results in the following:

- The method is added to the `Methods` node.
- Parameters to the method are added to the `Parameters` node.
- Elements corresponding to the parameters are added to the `Input Document Elements` node and to the XML Input Format pane. These elements are mapped to supply values for the parameters.
- Elements corresponding to the return value of the method are added to the XML Output Format pane.

Note – If you later move the method to another package or edit the method's class in such a way that the signature of the method is altered, you must remove the method call from the XML operation and add it back. Altering the method signature includes changes to the name, parameters, class of the return value, and list of exceptions.

Adding an Input Document Element

To add an input document element:

1. Open your XML operation in the Source Editor.

2. In the Data Source pane, right-click the `Input Document Elements` node and choose **Add Input Document Element**.

The Add Input Document Element dialog box is displayed, as illustrated in FIGURE 4-9.

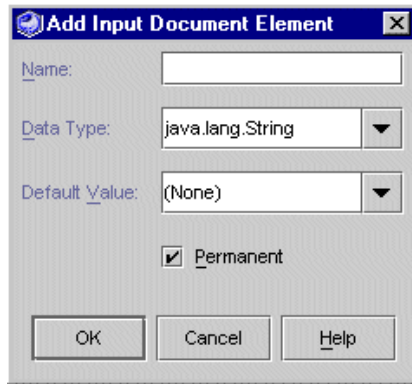


FIGURE 4-9 Add Input Document Element Dialog Box

3. Type a name for the element in the Name field.

4. Click the Data Type combo box and select a data type for the element.

If you intend to use this input document element as a parameter for instantiating a target object, you must choose a class (such as `String`) as a data type. Primitives (such as `int` or `double`) will not work.

5. (Optional) Specify a default value for the element.

If you want to specify a default value for the element, type it in the Default Value field.

This value is used if the client request does not provide this element.

6. If you want this element automatically deleted when it is no longer mapped to a method parameter, deselect the Permanent checkbox.

Whenever you remap a method parameter's source, input elements that are not currently mapped to a method parameter are deleted.

7. Click OK.

The new input element is added to the `Input Document Elements` node in the Data Source pane. The XML Input Format pane displays the updated XML input document.

Renaming an Input Document Element

To rename an input document element:

1. **Select the input document element, and open the Properties window, as illustrated in FIGURE 4-10.**
2. **Click the Name property.**

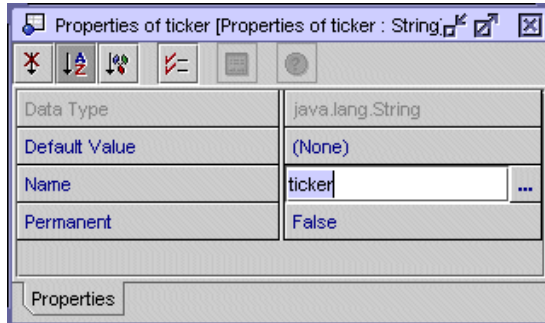


FIGURE 4-10 Input Document Element Properties Dialog Box

3. **Type a name for the input document element and press Enter.**

The input document element is now renamed. The new name is displayed both on the node inside the Input Document Elements folder and in the XML Input Format pane.

Renaming an Output Document Element

Each element in the XML output document is, by default, named after the method that returns the element's value. For example, adding a call to a method named `append` adds an element to the XML output document named `append`. Adding a call to a method named `getName` adds an element named `Name`. You can rename these elements by changing the value of the method call's `TagName` property.

To rename an output document element:

1. In the methods folder, select the method that returns the value for the element, select the Returns node of the method, and open the Properties window, as illustrated in FIGURE 4-11.
2. Click the Tag Name property.

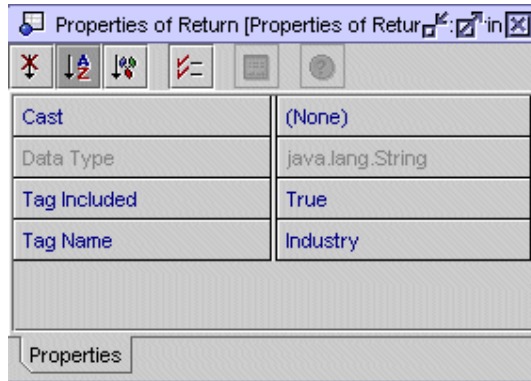


FIGURE 4-11 Output Document Element Properties Dialog Box

3. Type a name for the element and press Enter.

The output document element is now renamed. The new name is displayed in the XML Output Format pane.

Giving an Input Document Element a Default Value

To specify a default value for an input document element:

1. Open the property sheet for the input document element, as illustrated in FIGURE 4-12.

2. Click the Default Value property.

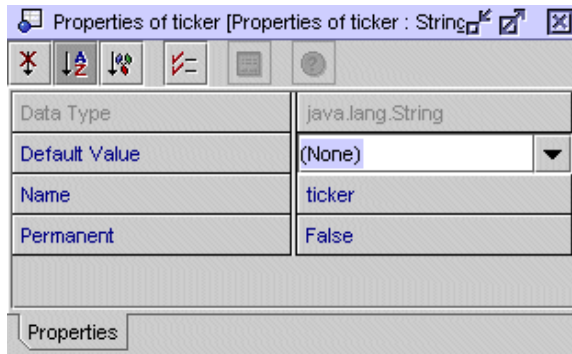


FIGURE 4-12 Input Document Element Properties (Default Value)

3. Type a value or, to specify a value of null, select (None).

This value is used if the client request does not provide this element.

Making an Input Document Element Permanent

The IDE automatically deletes any input document element that is not mapped to a method parameter unless the Permanent property of the input document element is enabled.

For example, if you add a method that takes a parameter to your XML operation, the IDE automatically adds an input document element and maps it to the method parameter. If you then remap the method parameter to a different source (for example, to the return value of another method), the IDE deletes the input document element because it is no longer mapped to a method parameter.

To prevent an input document element from being automatically deleted if it is not mapped, enable its Permanent property.

To enable an input document's Permanent property:

1. Open the property sheet for the input document element, as illustrated in

FIGURE 4-12.

2. Click the Permanent property.

3. Select True in the combo box and press Enter.

The input document element is now permanent.

Reordering a Method or Input Document Element

You can reorder the input document elements and methods in an XML operation.

Reordering methods changes the order in which the methods are called and changes the order of the elements in the XML output document. Methods are called in the order they are listed in the Source Editor, from top to bottom.

The return value of one method in an XML operation can be used as a parameter to another method in the XML operation (see “Mapping a Method Parameter to a Source” on page 98 for information on this topic). If you have such a dependency in your XML operation, you must ensure that the method supplying the parameter is called before the method that requires the parameter.

The ability to reorder input document elements is a development convenience. At runtime the order of input document elements has no significance to the web service.

To reorder a method or input document element:

1. **Open your XML operation in the Source Editor and locate the method or input document element you want to reorder.**
2. **Right-click the method or input document element and choose Move Up or Move Down.**

Deleting a Method or Input Document Element

You can delete input document elements and methods from an XML operation. When you delete a method, it means that the method is not called when the XML operation is executed. The method’s corresponding XML output elements are also removed.

To delete a method or input document element:

1. **Open your XML operation in the Source Editor and locate the method or input document element you want to delete.**
2. **Right-click the method or input document element and choose Delete.**

Mapping a Method Parameter to a Source

To map a method parameter to a source:

1. **Open your XML operation in the Source Editor and locate the method parameter you want to map.**
2. **Right-click the parameter and choose Change Source.**

The Method Parameter Source dialog box is displayed, as illustrated in FIGURE 4-13.

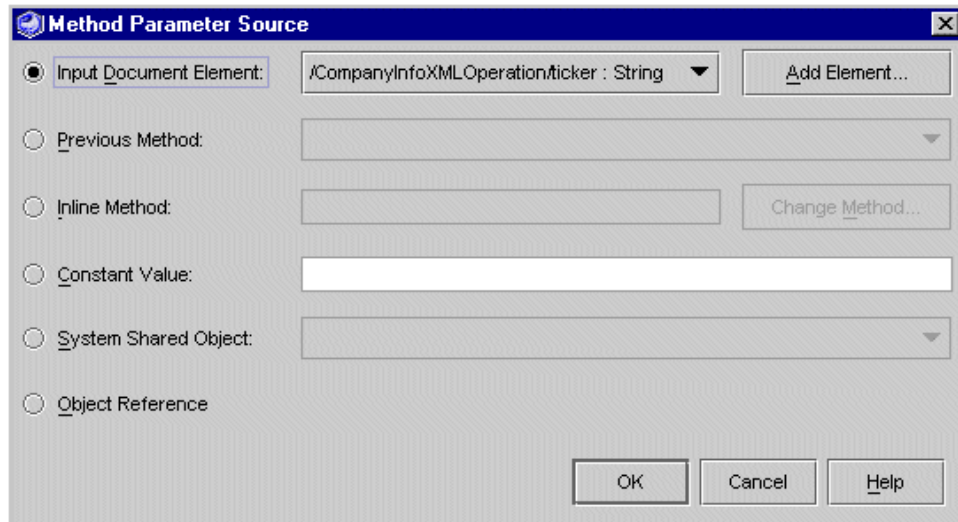


FIGURE 4-13 Method Parameter Source Dialog Box

3. **Select a source type.**

The following table describes the available source types.

| Source Type | Description |
|------------------------|---|
| Input Document Element | Select this radio button if you want to map the parameter to an element in the XML input document. |
| Previous Method | Select this radio button if you want to map the parameter to the return value of a previous method call in this XML operation. |
| Inline Method | Select this radio button if you want to map the parameter to the return value of an inline method call. Click Change Method to select the desired method. |
| Constant Value | Select this radio button if you want to map the parameter to a constant value. |

| Source Type | Description |
|----------------------|---|
| System Shared Object | <p>A web service maintains <i>system shared objects</i> at runtime containing data about web browser clients that access the web service. System shared objects are instantiated by the web service and populated with data obtained from the HTTP request and J2EE security mechanism.</p> <p>System shared objects maintain data about the user name. The data is provided as both a <code>String</code> object and a <code>java.security.Principal</code> object.</p> <p>Select this radio button if you want to map the parameter to one of these data types.</p> |
| Object Reference | <p>Select this radio button if you want to map the parameter to a target object defined in your web service. This option is available only if the parameter's type is a class (which includes <code>String</code>).</p> <p>Selecting this radio button adds a default object reference in your web service that maps the parameter to a new object of the required class. You can reconfigure this reference to resolve to an object of your choice. For more information on this topic, see "Instantiating Objects and Resolving References" on page 54.</p> |

4. Specify the source of the parameter's value.

If the source type is set to Input Document Element, Returned by Method, Constant Value, or System Shared Object, use the enabled field to specify a source. The following table describes the action to perform depending on the specified source type.

| Field Name | Action |
|----------------------------------|--|
| Input Document Element | Select an input document element from the enabled combo box. The combo box lists all input document elements in the XML operation of the type required by the parameter. Selecting an input document element maps its value to the parameter. |
| Previous Method or Inline Method | Select a method from the enabled combo box. The combo box lists methods called before the current method that return the appropriate type for the parameter. Selecting a method maps its return value to the parameter. |
| Constant Value | In the enabled field, type a string indicating the value. For values of type <code>String</code> or <code>char</code> , do not type a quotation mark unless it is part of the value. |
| System Shared Object | <p>Select an object from the list. The list of objects available depends on your parameter's type.</p> <p>For a parameter of type <code>java.security.Principal</code>, the combo box lists one object, <code>UserPrincipal</code>.</p> <p>For a parameter of type <code>java.lang.String</code>, the combo box lists one object, <code>UserName</code>.</p> |

5. Click OK.

Casting a Method Return Value

To cast a method return value:

1. **Open your XML operation in the Source Editor and locate the method `Returns` node.**
2. **Right-click the node and choose Properties.**
One of the properties is Cast, with a default of (None). Another property is Data Type. You can change the value of Cast to any type consistent with Java rules.
3. **Click OK.**

Displaying and Selecting Inherited Methods

To display inherited methods:

1. **Open your XML operation in the Source Editor and locate the method `Returns` node.**
Alternatively, you can locate the `Returns` node in the Explorer.
2. **Right-click the node and choose Expand.**
The Expand window is displayed, with Getter methods automatically selected.
3. **Select the Show Inherited Methods checkbox.**
Inherited methods are displayed.
4. **Select the desired methods and click OK.**
The selected methods are displayed in the Explorer and in the Data Source pane.

Excluding an Element From the XML Output

To exclude an element from the XML output document:

1. **Open your XML operation in the Source Editor.**
2. **Open the `Methods` node located in the Data Source pane.**
3. **Identify the node in the Data Source pane that corresponds to the element you want to exclude.**

When you select a node in the Data Source pane, its corresponding element is highlighted in the XML Output Format pane. For example, FIGURE 4-14 shows a class node selected in the Data Source pane and its corresponding element highlighted in the XML Output Format pane.

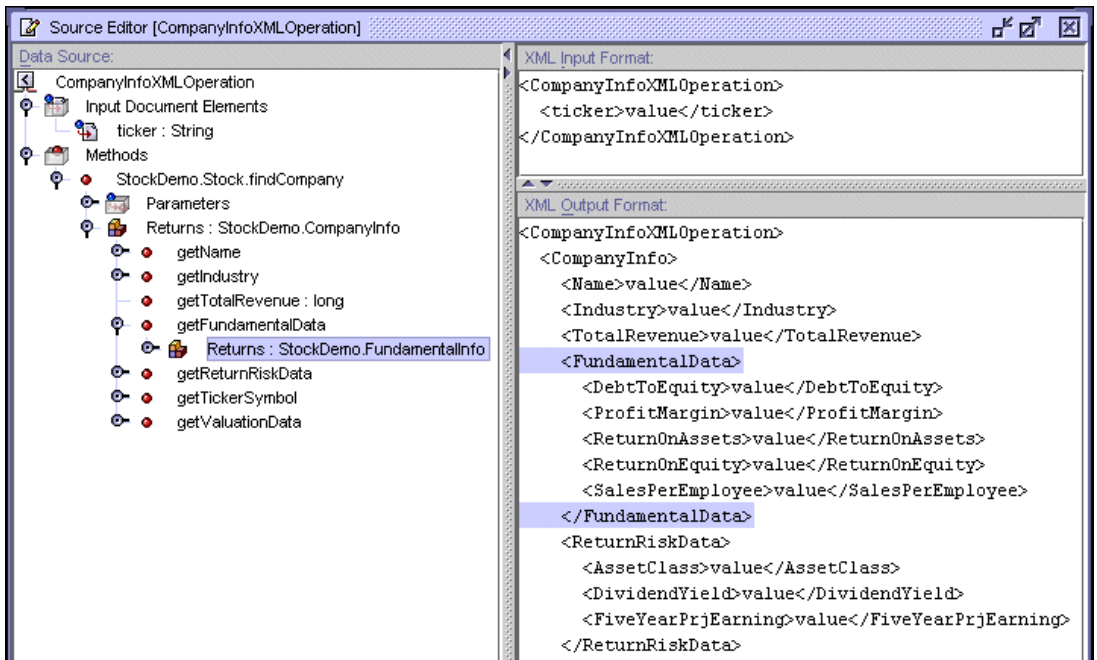


FIGURE 4-14 Source Editor: (Excluding an Output Element)

4. Right-click the node and choose **Exclude Tag from Output**.

The corresponding element is removed from the XML Output Format pane.

Including an Element in the XML Output

By default, all return values of your XML operation's method calls are included in the XML output document. If you exclude any of the elements representing these returned values, you can add them back to the XML output document.

To include an element in the XML output document:

1. Open your XML operation in the Source Editor.
2. Open the **Methods** node located in the Data Source pane.
3. Identify the node that corresponds to the element you want to include.

The types of nodes you can include are those that represent an array or collection, a class, or a method that returns a primitive.

4. Right-click the node and choose **Include Tag in Output**.

An element corresponding to the selected node is added to the XML Output Format pane.

Expanding a Class

To expand a class:

1. **Open your XML operation in the Source Editor.**
2. **Open the `Methods` node located in the Data Source pane.**
3. **Right-click the class and choose `Expand`.**

All getter methods on the class are added as subnodes to the class node. Elements corresponding to the class and to the return value of the getter methods are added to the XML Output Format pane.

Collapsing a Class

To collapse a class:

1. **Open your XML operation in the Source Editor.**
2. **Open the `Methods` node located in the Data Source pane.**
3. **Right-click the class and choose `Collapse`.**

The nodes representing the getter methods of the class are deleted from the Data Source pane. Their corresponding elements, as well as the element corresponding to the class, are deleted from the XML Output Format pane.

System Shared Objects

A web service maintains *system shared objects* at runtime containing data about web browser clients that access the web service. System shared objects are instantiated by the web service and populated with data obtained from the HTTP request and J2EE security mechanism. For information about how to use a system shared object, see “Mapping a Method Parameter to a Source” on page 98 and FIGURE 4-13.

System shared objects are limited to user name, which is obtained from the J2EE security mechanism and provided as both a `String` object and `java.security.Principal` object.

For a parameter of type `java.security.Principal`, the System Shared Object selection in the Method Parameter Source dialog box lists one object, `UserPrincipal`.

For a parameter of type `java.lang.String`, the System Shared Object selection lists one object, `UserName`.

Static Utility Methods

Data returned by a business method might require some type of processing before being displayed on a web page. For example, an EJB method might return a value of type `double`, but you would like to display the value in a monetary format. You can best accomplish processing of this sort by writing or reusing static utility methods rather than by adding new methods to your business components.

Organizing Static Utility Methods

For ease of use, you should organize static utility methods that you write into a small number of classes in the IDE.

Place the utility classes that are specific to the web service under development in a service-specific location, for example, in the package containing your web service. Place general purpose utility classes in a package that can be easily reused in other services and by other web service developers.

Using Static Utility Methods

To use a static utility method in an XML operation:

- 1. Mount the static utility class in the Explorer.**
- 2. Open your XML operation in the Source Editor.**
- 3. Add a call to the utility method to your XML operation.**

See “Adding a Method to an XML Operation” on page 91 for information on how to do this.

The utility method call must be positioned after the method call that returns the data to be processed. To reposition the method, right-click it and choose Move Up or Move Down.

When you add the utility method, the IDE adds an element to the XML output document corresponding to the return value of the method. This element is displayed in the XML Output Format pane.

The IDE also adds a new input document element corresponding to the method’s input parameter. You can ignore this input document element; the IDE will delete it for you after you remap the method parameter source in the next step.

4. **Map your utility method's input parameter to the value returned by the method call on the data component.**

The method call on the data component returns the value that you want to process with the utility method. So, you must map the output from the data component to the input of your utility method.

a. **Expand the utility method node and then the Parameters node.**

b. **Right-click the parameter and choose Change Source.**

The Method Parameter Source dialog box is displayed, as illustrated in FIGURE 4-15.

c. **Select Previous Method.**

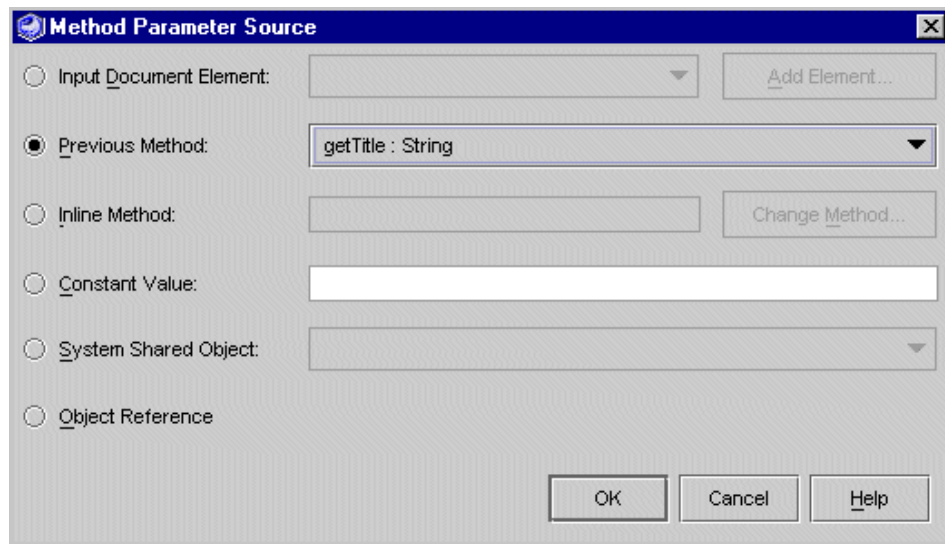


FIGURE 4-15 Method Parameter Source Dialog Box

d. **In the enabled combo box, select the method that returns the data from the data component and click OK.**

See FIGURE 4-13 and "Mapping a Method Parameter to a Source" on page 98 for more information on mapping parameters.

5. **Exclude the return value of the data-component method call from the XML output document, by right-clicking the method that calls the data component and choosing Exclude Tag from Output.**

The element is removed from the XML Output Format pane. Your client needs the processed data, but in most cases not the raw data returned by the data component.

6. (Optional) Set the utility method's Tag Name property to a more appropriate name.

7. (Optional) Rename the output document element.

When you added the utility method to the XML operation, the IDE added a corresponding element to the XML output document. By default, this element is named after the method (for example, `formatAsDollars`). In most cases, some other name would be more appropriate (for example, `Price`). To rename the element, change the value of the utility method's Tag Name property. See "Renaming an Output Document Element" on page 94 for information on how to do this.

Integration of C++ Shared Libraries

This appendix explains how to use the functionality of C and C++ native libraries in a web service. The explanation assumes that you have general familiarity with the Sun ONE Studio 4, Enterprise Edition for Java IDE and that you have read the earlier parts of this book, especially Chapter 2 and Chapter 3, which explain how to create web services and web service clients.

There are two aspects to the integration process, roughly corresponding to the roles of the C++ developer and the Java web service developer. This book is addressed to the Java web service developer. In that role, you might find it helpful to have a general understanding of what the C++ developer does to prepare a C++ component so that its methods can be exposed in a Sun ONE Studio Java web service. The following section provides a summary of the C++ side of the process. For more information on that subject, see the Native Connector Tool online help in the IDE.

Tasks of the C++ Developer

The role of the C++ developer is to provide a native connector file and native connector components.

The *native connector file* is used by the Java developer in the IDE to select methods and add them to a web service, either as direct method invocations or as part of more complex XML operations. Native connector file names end with the `.cdf` extension. They are displayed in the Explorer with an icon containing an orange sphere (🟠). The content of a native connector file is an XML document.

The *native connector components* are used by the web service at runtime. Java wrapper classes are generated, along with required descriptors and interfaces. C++ methods in a native shared library are accessed as Java methods in the generated wrappers.

The *Native Connector Tool (NCT)* is used by the C++ developer in the IDE to create and edit the native connector file, and to generate the native connector components.

The NCT provides a wizard to create native connectors, an editor to select native functions and data objects to be visible to Java components, and a code generator to create the interfaces between the native code and Java components, using the Java Native Interface (JNI). The resulting Java wrapper classes can be used anywhere that the Java language can be used.

The basic steps in the process are these:

1. **Create an enhanced native library by compiling the library source code using the `-xnativeconnect` compiler option.**

You must use compilers in the Sun[™] Open Net Environment (Sun ONE) Studio 7, Compiler Collection for this step.

2. **Create a native connector file for the enhanced native library.**
3. **Edit the native connector file (with the NCT editor) to select the objects, methods, and properties to expose in the Java API.**

Note – If you add methods individually, you must also add the class's no-arg constructor. This is a Sun ONE Studio 4, Enterprise Edition for Java web services requirement. If you add methods by clicking Add All, the constructor is added automatically.

4. **Generate the native connector file to create the native connector (wrapper) components.**

The process is displayed in FIGURE A-1.

Using the Native Connector Tool to Create Wrapper Components

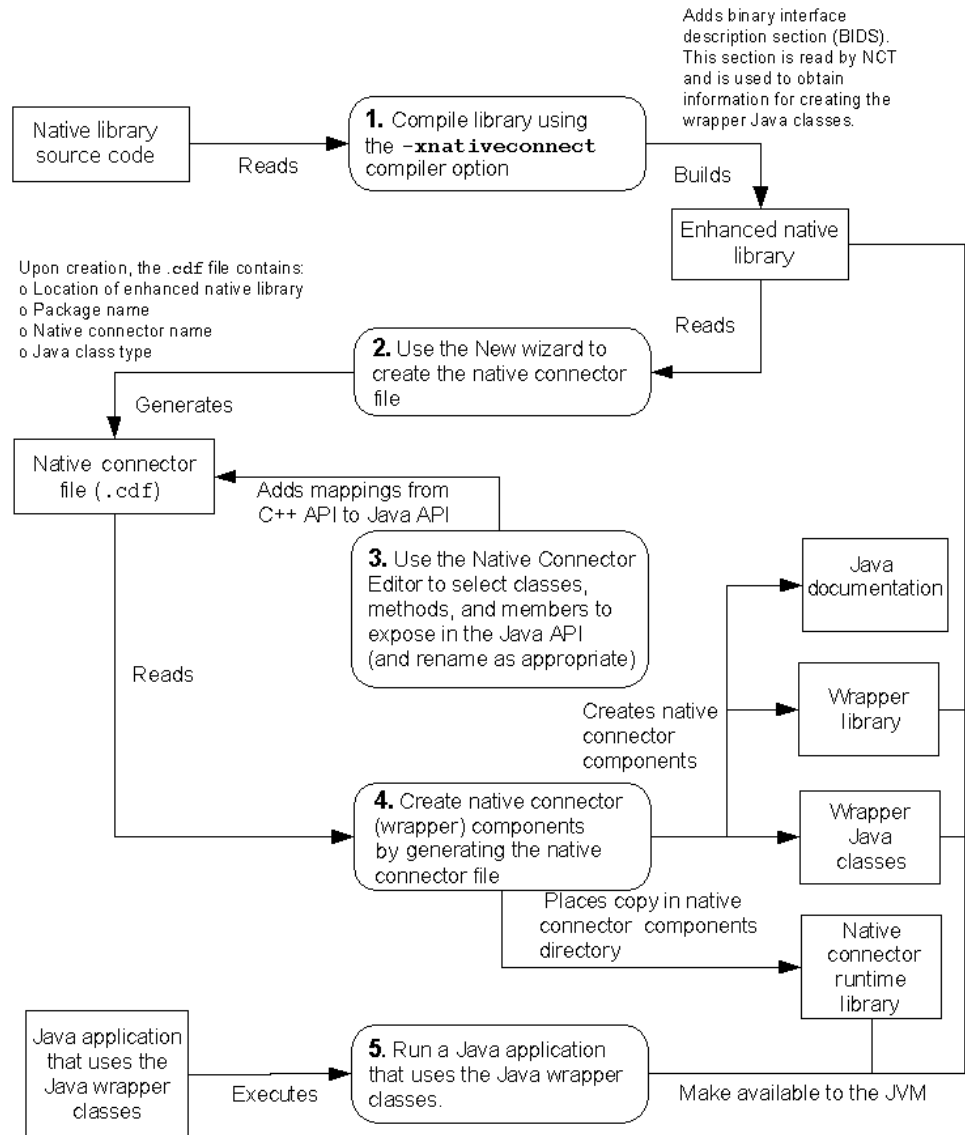


FIGURE A-1 Native Connector Tool Work Flow

Tasks of the Java Web Service Developer

The Java web service developer can expose methods of C and C++ shared libraries in direct method calls or as part of XML operations. The same web service can also expose the methods of EJB components and (if the web-centric architecture is used) web tier components. See Chapter 2 for a complete explanation of how to build, test, assemble, and deploy a web service.

The following procedure assumes that a C++ developer has provided you with the required C++ shared libraries and the native connector file. To add a C or C++ method to your web service:

1. Mount the directory containing the native connector file.

The file name should have a .cdf extension. FIGURE A-2 shows a native connector node named `myConnector` expanded in the Explorer. The available methods are displayed in the `Methods and Field Accessors` node.

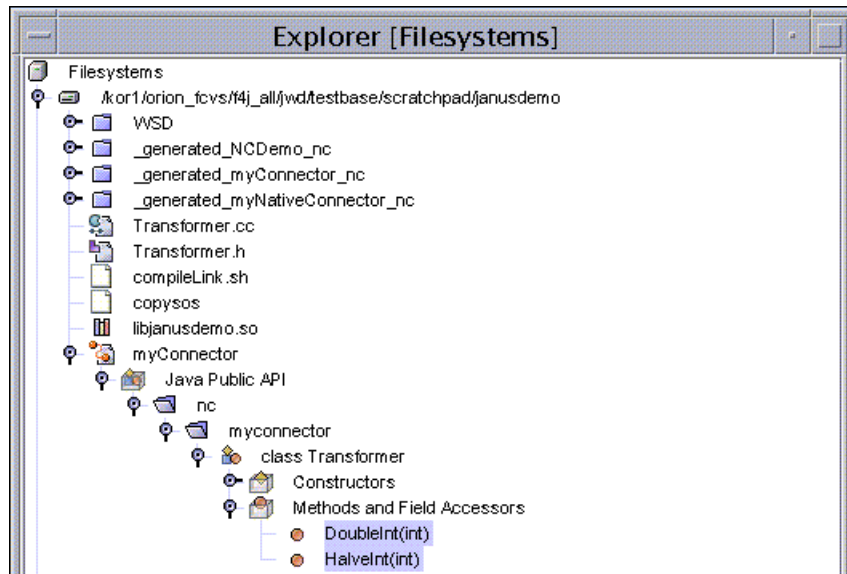


FIGURE A-2 Native Connector Node

2. Right-click your web service in the Explorer, and choose Add Reference.

A file browser is displayed.

3. **Select the native connector node, expand it in the file browser tree view, and select the desired methods.**

You can select more than one method by holding down the Control key while clicking.

4. **Click OK in the file browser.**

The selected methods are added to your web service.

You can also add methods from the New Web Service wizard, which displays the file browser as part of the process of creating a new web service.

Note – Apart from Step 1 and Step 3, this is the same procedure, described in Chapter 2, that you use to expose Java component methods in a web service.

Software Requirements

For information about the required platforms and operating systems, see the release notes on the following web site:

<http://forte.sun.com/ffj/documentation/index.html>

When you install the Sun ONE Studio 4, Enterprise Edition for Java IDE, use the installation wizard and install everything, including Core Platform and Modules, and Solaris Developer.

Customizing Your J2EE RI Installation

When you install the IDE, the installer puts the J2EE Reference Implementation in a directory directly under your top-level directory for Sun ONE Studio 4, Enterprise Edition for Java: `$s1studio_HOME/j2sdlkeel.3.1`. You can also use an external RI. If you are using

an external RI, modify the following procedure to refer to the corresponding files in your RI location.

1. **Edit the file `$J2EE_HOME/lib/security/server.policy`.**

Near the bottom of this file is a section labeled “default permissions granted to all domains.” The first permission grants the `queuePrintJob` runtime permission to all domains. Add the following line after the first permission:

```
Permission java.lang.RuntimePermission "loadLibrary.*";
```

2. Edit the file `$J2EE_HOME/bin/setenv.sh`.

Find the line that defines the `SYSTEM_LIB_DIR` variable. Insert the following two lines after it:

```
LD_LIBRARY_PATH=$J2EE_HOME/nativelib
export LD_LIBRARY_PATH
```

Installing the Shared Libraries in Your Application Server

Three shared libraries are used at runtime, as illustrated in FIGURE A-1. The libraries must be installed on the system that hosts the application server, so that they can be found by the application server.

The J2EE RI has a directory named `nativelib` that is intended for native libraries. Copy your three shared libraries to this directory. The J2EE RI uses the value of `LD_LIBRARY_PATH` to find the libraries.

From a terminal window, install the shared libraries by executing commands such as the following:

```
cd /mydisk1/NativeConnectorDemo/_generated_NCDemo_nc/sparc-SunOS
cp *.so $s1studio_HOME/j2sdkee1.3.1/nativelib
```

Instantiating Objects and Resolving References

When developing an XML operation, you specify which methods the XML operation calls. To call these methods at runtime, the web service requires certain objects. For each method call, it locates or instantiates:

- An instance of the class in which the method is defined
- An instance of each class required by the method as a parameter

To perform this task, the web service maintains a reference to each of these target objects and a definition of how to instantiate an object of the appropriate class should the target object not exist. As you add method calls to an XML operation, default object references and target object definitions are automatically added to the web service. These defaults are usually appropriate and do not need editing.

However, you can manually specify the target of an object reference, and you can edit and create new target object definitions to suit your requirements. You might need to manually resolve object references to enterprise beans that were created outside of the IDE.

This section provides instructions that explain how to:

- Specify the target of an object reference
- Define a new target object
- Edit a target object definition

Specifying the Target of an Object Reference

To specify the target of an object reference:

1. Open the Resolve Object References dialog box.

In the Explorer, right-click your web service and choose Resolve Object References. The Resolve Object References dialog box (see FIGURE B-1) is displayed.

| Target Class | Usage | Target Object |
|-----------------|-----------------|---------------|
| StockDemo.Stock | *.findCompany | Stock |
| StockDemo.Stock | *.purchaseStock | Stock |

Target Object

Name:

Scope: ☒ Session ☐ Message

Source: ▼

EJB Name:

EJB Type: ☐ Entity ☒ Session

EJB Ref Name:

EJB Remote Interface:

EJB Home Interface:

Method:

FIGURE B-1 Resolve Object References Dialog Box

If your web service has references to XML operations, the table in the dialog box has an additional column displaying XML operation names, as illustrated in FIGURE B-2.

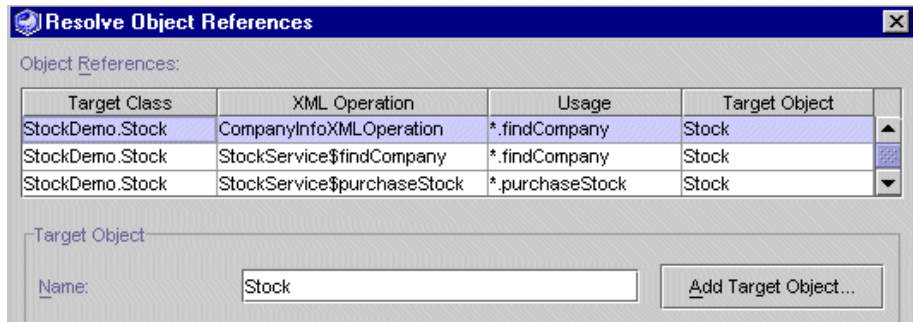


FIGURE B-2 Resolve Object References Dialog Box With XML Operations

2. Locate the object reference you want to edit.

Object references are listed at the top of the dialog box in table format. Each row represents one reference. The following table describes the columns.

| Column Name | Description |
|---------------|---|
| Target Class | Class of object required by the reference. |
| XML Operation | Name of XML operation that holds the reference to the target object. |
| Usage | <p>Indicates how the XML operation uses the target object. There are two possible uses. The XML operation can:</p> <ul style="list-style-type: none"> • Call a method on the target object • Pass the target object as a parameter to a method call <p>This column provides the name of the method that is called and shows (by use of an asterisk) how the target object is used to facilitate the method call.</p> <p>For example, a value of <code>*.getCustomer</code> indicates that a method named <code>getCustomer</code> is called on the target object. A value of <code>updateCustomer(customerInfo:*)</code> indicates that the target object is passed as the <code>customerInfo</code> parameter to a method named <code>updateCustomer</code>.</p> |
| Target Object | Name of object that resolves the reference. |

3. Select an object to resolve your reference.

In the row representing the object reference you are editing, click in the Target Object column and select an object from the list that drops down. This list displays all available objects of the required class, which includes target objects already defined in this dialog box.

Defining a New Target Object

If none of the target objects already defined is appropriate for your object reference, you can define a new target object.

To define a new target object:

- 1. Open the Resolve Object References dialog box.**

In the Explorer, right-click your web service and choose Resolve Object References. The Resolve Object References dialog box is displayed (see FIGURE B-1).

- 2. Select the reference for which you want to define a new target object.**

The dialog box displays the definition for the current target object.

- 3. Click Add Target Object.**

The New Target Object dialog box is displayed.

- 4. Type a name for the target object in the Target Object Name field.**

- 5. Click OK.**

The dialog box automatically resolves your reference to the newly defined target object. Your new target object uses the same definition as the previous one. Only the name by which it is referenced is changed. For instructions on editing the definition, see “Editing a Target Object Definition” on page 116.

Editing a Target Object Definition

The target object definition specifies how the web service:

- Locates a target object that is already instantiated
- Instantiates a new target object

A web service can have multiple references that resolve to the same target object. For example, the same session bean might be used in many XML operations in the web service. In such a scenario, editing the target object definition affects each of these references because they all resolve to the same object. If this behavior is inappropriate for your application, you might need to create a new target object definition for one or more of your references. For instructions on creating a new definition, see “Defining a New Target Object” on page 116.

To edit a target object definition:

1. **Open the Resolve Object References dialog box.**

In the Explorer, right-click your web service and choose Resolve Object References. The Resolve Object References dialog box is displayed (see FIGURE B-1).

2. **Edit the Name field, if necessary.**

This field specifies the name by which the target object can be referenced. The target object's value is also displayed in the Target Object column in the list of object references at the top of the dialog box. This field is required.

3. **Edit the Scope field, if necessary.**

This required field defines the scope within which the target object can be referenced. Options are:

- **Session.** The target object can be referenced by all XML operations in the session for the duration of the session.
- **Message.** The target object can be referenced only during the execution of the client request that instantiated it. Because only one XML operation is executed per request, access to the target object is limited to that one XML operation. Each time the XML operation is executed, a new target object is created.

Note – Your choice of scope has no effect on the runtime characteristics of a web service that you create in the current release of the Sun ONE Studio 4, Enterprise Edition for Java IDE.

4. **Edit the Source field, if necessary.**

This field specifies the type of mechanism by which the target object is obtained or instantiated. Options are:

- **Enterprise JavaBean.** The target object is the remote or home interface of an enterprise bean. A home interface is obtained through a JNDI lookup. A remote interface is obtained through a method call to the corresponding home interface.
- **Constructor.** The target object is returned by a call to a constructor method on the target class.
- **Static Method.** The target object is returned by a call to a static method.

5. Edit the remaining fields, if necessary.

The Source field affects which other fields in the dialog box are displayed and enabled for input. Edit the remaining fields in the dialog box by referring to the descriptions in whichever of the following tables is appropriate for your target object's source.

If the source is an enterprise bean, refer to TABLE B-1.

TABLE B-1 Fields Enabled When Source Field Is Set to Enterprise JavaBean

| Field | Description |
|----------------------|---|
| EJB Name | Canonical name of the enterprise bean as defined in the EJB module deployment descriptor. |
| EJB Type | Type of enterprise bean, either entity or session. Required. |
| EJB Ref Name | String in the JNDI lookup that specifies the target object. The default value is the value of the Name field, prefixed with the string <code>ejb/</code> . Required. |
| EJB Remote Interface | Remote interface of the enterprise bean. Required even if the target object is the home interface. |
| EJB Home Interface | Home interface of the enterprise bean. Required. |
| Method | Find or create method called on the home interface to return the remote interface. A find method is used for an entity bean and a create method is used for a session bean. Required only if the target object is a remote interface. |

If the sources is a constructor, refer to TABLE B-2.

TABLE B-2 Fields Enabled When Source Field Is Set to Constructor

| Field | Description |
|-------------|---|
| Class | The class in which the constructor of the target object is defined. This field is automatically filled in when you select a constructor for the Constructor field. Read-only. |
| Constructor | Constructor method used to instantiate the target object. The class of the constructor is specified in the Class field. Required. |

If the source is a static method, refer to TABLE B-3.

TABLE B-3 Fields Enabled When Source Field Is Set to Static Method

| Field | Description |
|---------------|---|
| Class | The class in which the static method that returns the target object is defined. This field is automatically filled in when you select a static method for the Static Method field. Read-only. |
| Static Method | Static method used to instantiate the target object. Required. |

6. Map method parameters to sources, if necessary.

If the Map Parameters button is enabled, follow these steps to provide values for parameters of the method specified in the Method, Constructor, or Static Method field.

a. Click Map Parameters.

The Map Parameters dialog box is displayed.

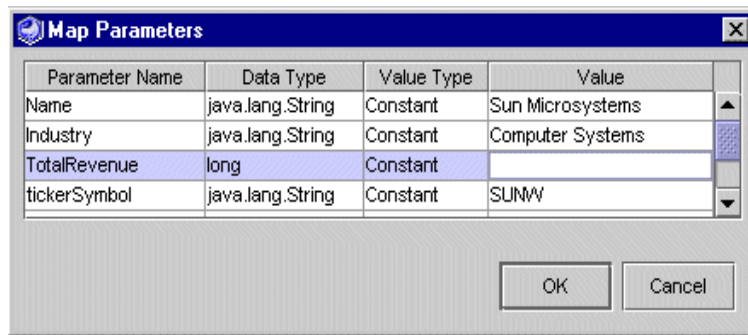


FIGURE B-3 Map Parameters Dialog Box

b. Locate the row that represents the parameter you want to map.

Each row represents one method parameter. If your method takes multiple parameters, use the Parameter Name column to identify the parameters. This column gives the names of the parameters if the method is provided as source code. If the method is provided in a JAR file, the Parameter Name column indicates the order in which the method requires the parameters by displaying names such as param1, param2, and so on.

c. Specify a value type for your parameter.

Click in the Value Type column and select an option from the list. The following table describes the options.

| Option | Description |
|------------------------|---|
| Constant | Maps the parameter to the value specified in the Value field. |
| Environment Entry | Maps the parameter to the value of the environment entry specified in the Value field. For information on setting environment entries, see “Adding Environment Entries” on page 26. |
| Target Object | Maps the parameter to the object specified in the Value field. The object can be a target object defined in the web service’s Resolve Object References dialog box. |
| Input Document Element | Maps the parameter to the input document element specified in the Value field. The data type of the input document element must be an object (for example, <i>String</i>); it cannot be a primitive (for example, it cannot be <i>int</i>). For information on declaring the data type of an input document element, see “Adding an Input Document Element” on page 92. |

d. Specify a value for your parameter in the Value field.

The following table explains how to specify a value depending on your parameter’s value type.

| Value Type | User Action |
|------------------------|--|
| Constant | Type a literal value in the Value field. |
| Environment Entry | Type the name of an environment entry in the Value field. For information on setting environment entries, see “Adding Environment Entries” on page 26. |
| Target Object | Click the Value field and select an object from the list. The list includes all target objects of the class specified in the Data Type field that are defined in the web service. |
| Input Document Element | Click the Value field and select an input document element from the list. The list includes input document elements that are both: <ul style="list-style-type: none">• Of the specified data type• Defined in each XML operation that references the target object The second requirement ensures that the parameter you are mapping has a source for its value in all circumstances. For example, two XML operations could reference the same target object, but you might not know which XML operation will be executed first. Therefore, each of these XML operations must provide a value for the parameter so that no matter which is executed first, a parameter value is available. For more information on input document elements, see “Input Document Elements Node” on page 83. |

Deployment Descriptors

When you generate runtime classes for your web service, a web module and EJB module deployment descriptor are also generated, depending on whether you chose multitier or web-centric architecture. When you assemble your web service J2EE application, these deployment descriptors are included in the application. The deployment descriptors are XML files used to configure runtime properties of the application. The J2EE specification defines the format of these descriptors.

You can view the deployment descriptors in the Source Editor at any time during development. You can also edit the deployment descriptors. However, deployment descriptors that you edit are not regenerated if you regenerate runtime classes. Deployment descriptor edits are thereby preserved. However, changes you make to your web service after editing a deployment descriptor will not be propagated to your deployment descriptor. You should, therefore, take care to edit deployment descriptors only at the end of your development cycle.

Fields Propagated to Deployment Descriptors

This section lists the fields in the IDE that are normally propagated to your web service's deployment descriptors. If you edit one of these deployment descriptors and afterward change the value of any field that is normally propagated to that deployment descriptor, you must edit the deployment descriptor manually and insert the new value.

Fields Propagated to the EJB Module Deployment Descriptor

The Resolve Objects dialog box contains several fields that are propagated to the EJB module deployment descriptor. The corresponding elements in the deployment descriptor are all subelements of an `ejb-ref` element. The following table lists those fields.

| Fields in Resolve Objects Dialog Box | Elements in EJB Module Deployment Descriptor |
|--------------------------------------|--|
| EJB Ref Name | <code>description</code> |
| EJB Ref Name | <code>ejb-ref-name</code> |
| EJB Type | <code>ejb-ref-type</code> |
| EJB Home Interface | <code>home</code> |
| EJB Remote Interface | <code>remote</code> |
| EJB Name | <code>ejb-link</code> |

Note – Adding an EJB method call to one of your XML components might create a new target object definition. The first time you add a method call to a particular EJB component, a new target object definition is created in the Resolve Objects dialog box. Normally, this definition is propagated to the EJB module deployment descriptor as a new `ejb-ref` element.

Your web service's property sheet contains a property named Environment Entries. The fields in this property are propagated to the EJB module deployment descriptor. The corresponding elements in the deployment descriptor are all subelements of an `env-entry` element. The following table lists those fields.

| Fields in the Environment Entries Property | Elements in EJB Module Deployment Descriptor |
|--|--|
| Name | <code>env-entry-name</code> |
| Description | <code>description</code> |
| Type | <code>env-entry-type</code> |
| Value | <code>env-entry-value</code> |

Fields Propagated to the Web Module Deployment Descriptor

If you set a default test client for a web service, the client's welcome page and error page are propagated to the web module's deployment descriptor. The following table lists those fields.

| Properties | Elements in Web Module Deployment Descriptor |
|--------------|--|
| Error Page | location (a subelement of error-page) |
| Welcome Page | welcome-file (a subelement of welcome-file-list) |

See “Setting a Default Test Client for a Web Service” on page 34 and “The Client HTML Pages and JSP Pages” on page 61 for information about default test clients.

Viewing a Deployment Descriptor

To view the web module or EJB module deployment descriptor:

- **Right-click your web service node and choose one of these menu items:**
 - Deployment Descriptor → Web Module → View
 - Deployment Descriptor → EJB Module → View

The deployment descriptor is displayed in the Source Editor in read-only mode.

Editing a Deployment Descriptor

To edit the web module or EJB module deployment descriptor:

1. **Right-click your web service node and choose one of these menu items:**
 - Deployment Descriptor → Web Module → Final Edit
 - Deployment Descriptor → EJB Module → Final Edit

The Final Edit dialog box is displayed, as illustrated in FIGURE C-1. The dialog box reminds you that after editing the deployment descriptor, the descriptor will no longer be regenerated by the IDE when you regenerate the runtime classes of your web service.

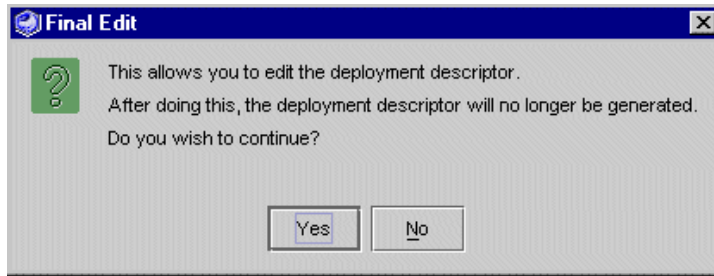


FIGURE C-1 Deployment Descriptor Final Edit Dialog Box

2. **If you are sure you will not need to regenerate the deployment descriptor, click Yes. Otherwise click No.**

The deployment descriptor is displayed in the Source Editor in edit mode. The deployment descriptor will not be regenerated if you regenerate runtime classes for your web service.

Index

A

- adding input document elements, 92
- adding method references
 - to an XML operation, 91
- adding references
 - to a web service, 20
- Apache SOAP
 - version, 12
- architecture
 - multitier, 10
 - web-centric, 11, 18, 29
- arrays, 55
- assembling a web service, 29

C

- C++ shared libraries, 107
- casting a method return value, 100
- classes returned by XML operations
 - collapsing, 102
 - expanding, 85, 102
- clients
 - creating from a UDDI registry, 69
 - creating from a web service, 57
 - creating to test a web service, 34
 - generating
 - HTML error page, 61
 - HTML welcome page, 61
 - JSP pages, 61
 - SOAP proxy, 60, 63

collections

- serialization classes, 55
- user-defined object types, 55

D

- data
 - retrieving more or less, 85
 - returned by XML operation, 84
 - returning less to client, 86
- deleting references, 22, 64
- deploying a web service, 29
- deployment descriptors
 - editing, 54, 121
 - generated for web service, 54
 - IDE fields propagated to, 121
 - viewing, 54, 121
- development work flow
 - Native Connector, 107, 110
 - web service clients, 57
 - web services, 17
 - XML operations, 86

E

- elements
 - adding to XML input document, 92
 - deleting from input document, 97
 - excluding from XML output document, 86, 100
 - including in XML output document, 101
- environment entries, 26

- error pages, generating, 61
- example applications
 - where to download, xviii
- execution of XML operations, 13

G

- generating an XML operation from an EJB component, 89
- generating runtime classes, 28

I

- inherited methods, 100
- input document elements
 - adding, 92
 - deleting, 97
 - reordering, 97
- Input Document Elements node, 83
- instantiating objects at runtime, 54, 113

J

- Javadoc technology
 - using in the IDE, xviii
- JAX-RPC
 - client proxy, 59, 63
 - client runtime property, 59
 - version, 12
- JSP pages, generating, 61

K

- kSOAP
 - client front end, 64
 - client proxy, 59
 - client runtime property, 59
 - version, 12

M

- mapping parameters, 98

- method calls in XML operations
 - adding, 91
 - deleting, 86, 97
 - description, 84
 - order of execution, 84
 - reordering, 97
- Methods node, 84
- multitier architecture, 10

N

- Native Connector
 - C++ developer tasks, 107
 - development work flow, 107, 110
 - Java web service developer tasks, 110
- Native Connector Tool, 107, 109

O

- object references
 - resolving, 54, 113
 - specifying target object, 113
- objects
 - instantiating at runtime, 54, 113
 - scope of references, 117
- operations, *See* XML operations

P

- parameters
 - mapping, 85
 - mapping in web service, 119
 - mapping in XML operation, 85, 98, 119
 - source type, 85
- Parameters node, 85

R

- references
 - adding to web service, 20
 - deleting from web service, 22
 - resolving, 54, 113
- refetch WSDL, 65
- request-response mechanism, 13

- Resolve Object References dialog box, 114, 117
- resolving object references, 54, 113
- runtime classes, generating, 28

S

- scope of object references, 117

- servers

- configuring and starting the Reference Implementation server, 32
 - starting and stopping the internal UDDI registry server, 49

- SOAP

- Apache SOAP implementation, 12
 - client front end, 64
 - client proxy, 63
 - client runtime property, 59
 - description, 4
 - IDE features, 9
 - JAX-RPC implementation, 12
 - kSOAP implementation, 12
 - specification, 12

T

- target objects

- defining new, 116
 - editing definitions, 116
 - instantiating objects, 54, 113
 - resolving references, 54, 113
 - specifying an object reference, 113

U

- UDDI

- creating a client from a registry, 69
 - internal registry
 - overview, 49
 - starting and stopping, 49
 - using the sample browser, 50
 - managing registry options
 - default categories and identifiers, 39
 - editing the table of registries, 41
 - overview
 - industry standards, 4, 6

- specification, 13

- Sun ONE Studio 4 IDE features, 9

- using a registry, 7

- publishing a web service to a registry, 7, 43

- utility methods, static, 103

W

- web services

- adding references, 20
 - assembling, 29
 - bottom-up development, 9, 17
 - creating, 18, 22
 - creating clients
 - for testing, 34
 - from local local service, 57
 - from UDDI registry, 69
 - from WSDL, 68
 - deleting references, 22
 - deploying, 29
 - description, 1
 - development work flow, 17
 - endpoint URL, 78
 - generating runtime classes, 28
 - scope of object references, 117
 - test clients, 34
 - top-down development, 9, 17

- web-centric architecture, 11, 18, 29

- welcome page, generating, 61

- WSDL

- creating clients, 68
 - creating web services, 9, 17, 22
 - description, 4, 5
 - generating from web service, 37
 - refresh, 65
 - specification, 12
 - Sun ONE Studio 4 IDE features, 9

X

- XML operations

- adding input document elements, 92
 - adding method calls, 91
 - collapsing classes, 85
 - creating, 87
 - Data Source pane, 83

- deleting method calls, 86, 97
- description, 13
- developing, 81
- development work flow, 86
- editing, 90
- excluding elements from XML output
 - document, 86
- expanding classes, 85
- generating from EJB, 89
- including elements in output document, 101
- mapping parameters, 85, 98
- reordering input document elements, 97
- reordering methods, 97
- request-response mechanism, 13
- specifying return data, 84
- XML Input Format pane, 83
- XML Output Format pane, 83
- XML output documents
 - excluding elements, 100
 - including elements, 101