



Building Web Services

Sun™ ONE Studio 5 Programming Series

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

Part No. 817-2324-10
June 2003, Revision A

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

Copyright © 2003 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.

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.

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 © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels 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 est un document protégé par un copyright et distribué avec des licences qui est 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.

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.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE 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.



Contents

Before You Begin 17

1. Web Services: A Conceptual Overview 25

What Are Web Services? 25

Industry Problems 25

Applications Within an Enterprise 26

Applications Shared Across Enterprises 26

A Standardized Solution 27

Web Services Standards 27

SOAP 28

WSDL 29

UDDI 29

Web Services Interoperability Organization 31

Publishing and Using a Web Service 31

Sun ONE Studio 5, Standard Edition Web Services 34

Web Services Architectures 35

Multitier Architecture 35

Web-centric Architecture 37

Deployment Architecture 38

Software Versions 39

XML Operations (Deprecated)	39
2. Building a Web Service	41
Web Service Development Tasks	41
Creating a JAX-RPC Web Service From Java Methods	42
Developing XML Operations (Deprecated)	44
Adding Operations to a Web Service	45
Deleting Operations From a Web Service	46
Resolving References to Runtime Objects	47
Adding Environment Entries	47
Creating a JAX-RPC Web Service From WSDL Source	48
Generating Runtime Classes	52
Assembling and Deploying a Web Service	53
A Web-centric Application	53
Assembling the J2EE Application	55
Deploying the J2EE Application	57
Testing a Web Service	58
Setting a Default Test Client for a Web Service	58
Testing a Web Service	58
Creating a Stateful Web Service	59
Setting Up the Business Component	60
Setting Up the Web Service and Test Client	64
Testing a Stateful Web Service	66
Working With UDDI Registries	68
Generating WSDL	69
Managing UDDI Registry Options	69
Setting Default Publish Categories and Identifiers	70
Editing Registries Information in the IDE	73
Gaining Access to an External UDDI Registry	74

Publishing Your Web Service to a UDDI Registry	75
Publication Tasks and Terminology	75
Publication Procedure	76
Using the Internal UDDI Registry	81
Starting and Stopping the Internal UDDI Registry Server	82
Using the Sample Registry Browser	82
Using Arrays and Collections	86
Arrays	86
Collections	87
Attachments	88
SOAP Message Handlers	88
Security	88
Deployment Descriptors	89
3. Creating a Web Service Client	91
Creating a Client From a Local Web Service	91
Creating the Client	91
Setting the Client Type: JAXRPC or kSoap	93
Generating a JAX-RPC Client	95
The Generated Files	95
The JSP Custom Tags	96
The Client HTML and JSP Pages	97
The Client Proxy Classes	99
Using Your Own Front-End Client Components	100
Validation During Client Generation	102
Refreshing a Client From Its Web Service	102
The Service Endpoint URL	103
Deploying a JAX-RPC Client	103
Executing a JAX-RPC Client	105

Firewalls	108
Distributed Applications	109
Generating a kSOAP Client	110
Executing a kSOAP Client	111
Creating a Client From a WSDL File	112
Creating a Client From a UDDI Registry	113
Creating a Client—Planning and Work Flow	113
Creating a Client—Procedure	114
Attachments	123
Attachments Example	123
The DataHandler Property	128
Stateful Web Services and Clients	129
SOAP Message Handlers	129
Security	129
4. Using Message Handlers	131
SOAP Message Handlers	132
SOAP Message Headers and Header Blocks	132
Request Handlers and Response Handlers	132
Handler chains	134
SOAP Actor Roles	135
Handler Properties.	135
Getting Header Blocks at Handler Initialization	136
Adding a Header Block to a SOAP Message	137
Using Handlers In the IDE	138
Adding Handlers To a Web Service or Client	138
Adding a Handler Property	141
Adding a SOAP Header Block To a Handler	144
Setting a SOAP Actor Role For a Handler Chain	145

Enforcing the Processing of Header Blocks	147
Sample Handler Code	148
5. Developing XML Operations (Deprecated)	153
Overview of XML Operations	153
What Is an XML Operation?	154
Request-Response Mechanism	155
Overview of Tools	157
The Data Source Pane	158
Input Document Elements Node	159
Methods Node	159
Executing Methods and Returning Data	160
Providing Parameter Values	160
Retrieving More or Less Data	161
Trimming the Data Returned to the Client	161
Development Work Flow	162
Creating XML Operations	162
Creating an XML Operation	163
Generating XML Operations From an Enterprise Bean	165
Editing an XML Operation	166
Adding a Method to an XML Operation	166
Adding an Input Document Element	168
Renaming an Input Document Element	169
Renaming an Output Document Element	170
Giving an Input Document Element a Default Value	170
Making an Input Document Element Permanent	171
Reordering a Method or Input Document Element	172
Deleting a Method or Input Document Element	172
Mapping a Method Parameter to a Source	172

Casting a Method Return Value	175
Displaying and Selecting Inherited Methods	175
Excluding an Element From the XML Output	175
Including an Element in the XML Output	176
Expanding a Class	177
Collapsing a Class	177
System Shared Objects	177
Static Utility Methods	178
Organizing Static Utility Methods	178
Using Static Utility Methods	178
Instantiating Objects and Resolving References	180
Specifying the Target of an Object Reference	181
Defining a New Target Object	183
Editing a Target Object Definition	183
A. Securing a Web Service	189
Security Overview	189
HTTP Basic Authentication	189
HTTPS/SSL Authentication and Encryption	191
Public Key Encryption	191
Public Key Certificates	192
SSL Handshaking	193
Using HTTP Basic Authentication	194
Mapping Roles to Users and Groups	194
Mapping Roles at the Level of the Web Service	196
Mapping Roles at the Level of the J2EE Application	198
Use Case for Multiple Roles	198
Mapping Roles: Order of Priority	199
Mapping Roles In the Application Server	199

Redirecting Role Names	199
Testing Security: HTTP Basic Authentication	201
Basic Authentication and WSDL	204
Basic Authentication and UDDI Publication	204
Using HTTPS/SSL Authentication and Encryption	204
Setting Properties for the Web Service and Test Client	206
Setting Up the Test Client Trust Store	206
Creating the Trust Store	206
Setting the IDE's Global Trust Store Option	207
Importing the Server's Certificate Into the Test Client Trust Store	208
Setting up the Server	210
Testing Security: HTTPS/SSL Authentication and Encryption	212
B. Deployment Descriptors	215
Fields Propagated to Deployment Descriptors	215
Fields Propagated to the EJB Module Deployment Descriptor	216
Viewing a Deployment Descriptor	217
Editing a Deployment Descriptor	217
Index	219

Figures

FIGURE 1-1	Publishing and Using a Web Service	33
FIGURE 1-2	Web Service and Client (Multitier Model)	36
FIGURE 1-3	Web Service and Client (Web-centric Model)	38
FIGURE 2-1	New Web Service Wizard	42
FIGURE 2-2	Web Service Properties With Context Root Set to <code>StockService</code>	44
FIGURE 2-3	Logical EJB Nodes in the Explorer	45
FIGURE 2-4	Web Service Operations	46
FIGURE 2-5	Environment Entries Dialog Box	47
FIGURE 2-6	Add Environment Entry Dialog Box	48
FIGURE 2-7	New Web Service Wizard	49
FIGURE 2-8	New Web Service Wizard: Select WSDL File	50
FIGURE 2-9	New Web Service Wizard: Specify New EJB	51
FIGURE 2-10	Web Service and EJB Nodes Generated From WSDL: Explorer View	52
FIGURE 2-11	Web Service Hierarchy of Nodes	53
FIGURE 2-12	J2EE Application for a Web Service	55
FIGURE 2-13	Web Service WAR File Properties With Web Context Set to <code>StockService</code>	56
FIGURE 2-14	Application Server Instance for Deployment	57
FIGURE 2-15	Stateful Session Bean: Wizard	61
FIGURE 2-16	Stateful Session Bean <code>Bean Type</code> Property	61
FIGURE 2-17	Explorer View of Business Methods	62

FIGURE 2-18	Add New Business Method Dialog Box	63
FIGURE 2-19	Web Service <code>Conversational</code> Property	64
FIGURE 2-20	Web Service Conversation Initiator and Terminator Method Properties	65
FIGURE 2-21	Web Service Test Client <code>Conversational</code> Property	65
FIGURE 2-22	Stateful Test Client Welcome Screen	67
FIGURE 2-23	UDDI Registries Options	70
FIGURE 2-24	UDDI Publish Categories Dialog Box	71
FIGURE 2-25	UDDI Categories (Taxonomies)	71
FIGURE 2-26	UDDI Publish Identifiers Dialog Box	72
FIGURE 2-27	UDDI Add Identifier Dialog Box	73
FIGURE 2-28	UDDI Registries Property Editor	74
FIGURE 2-29	UDDI Publish New Web Service Dialog Box	77
FIGURE 2-30	UDDI Login and Business Information Dialog Box	78
FIGURE 2-31	UDDI <code>tModel</code> Selection Dialog Box	80
FIGURE 2-32	UDDI Set Categories and Identifiers	81
FIGURE 2-33	Start Internal UDDI Registry Server	82
FIGURE 2-34	Java WSDP Sample Registry Browser	84
FIGURE 2-35	Java WSDP Sample Registry Browser URL Selection	84
FIGURE 2-36	Java WSDP Sample Registry Browser With Internal Registry URL	85
FIGURE 2-37	Java WSDP Sample Registry Browser Displaying Selected Business	85
FIGURE 2-38	Java WSDP Sample Registry Browser Displaying Submissions Tabbed Pane	86
FIGURE 2-39	Serialization Classes Property Editor	88
FIGURE 3-1	New Client From Local Web Service Dialog Box	92
FIGURE 3-2	New Web Service Client Dialog Box	93
FIGURE 3-3	Client SOAP Runtime Property	94
FIGURE 3-4	Client Documents Node and GenClient Node in Explorer	96
FIGURE 3-5	Client HTML Welcome Page	98
FIGURE 3-6	Client Sample JSP Page	99
FIGURE 3-7	Client SOAP Proxy GenClient Node	100
FIGURE 3-8	Client References to Documents, Libraries, and Classes	101

FIGURE 3-9	Client WAR File in Explorer (Deployed Web Modules Node)	104
FIGURE 3-10	Client WAR File in Explorer (Client Node)	104
FIGURE 3-11	Client WAR File Contents	105
FIGURE 3-12	Client Welcome Page	106
FIGURE 3-13	Client Display of Company Information	107
FIGURE 3-14	Client Display of Company Information (SOAP Request/Response)	108
FIGURE 3-15	Client Input Proxy Server Page	109
FIGURE 3-16	kSOAP Client Nodes in the Explorer	110
FIGURE 3-17	kSOAP Client Executing In Cell Phone Emulator	111
FIGURE 3-18	New Web Service Client Wizard	114
FIGURE 3-19	UDDI Registry Selection Dialog Box	115
FIGURE 3-20	UDDI Registry Search Dialog Box	116
FIGURE 3-21	UDDI Registry Search Types	117
FIGURE 3-22	UDDI Registry Search Dialog Box With Matching Businesses	119
FIGURE 3-23	UDDI Registry Filter Business Progress Monitor	120
FIGURE 3-24	UDDI Registry Select Service	121
FIGURE 3-25	UDDI Registry Display Service Details and tModel	122
FIGURE 3-26	Explorer View of Web Service and Client That Process Images and Text	125
FIGURE 3-27	Client Welcome Page For Image and Text Processing	126
FIGURE 3-28	Client JSP Page For the <code>getImage</code> Method	127
FIGURE 3-29	Client Display of Image	127
FIGURE 3-30	Client Class <code>attWSRPC</code> With Use DataHandler Only Property <code>False</code>	128
FIGURE 3-31	Client Class <code>attWSRPC</code> With Use DataHandler Only Property <code>True</code>	129
FIGURE 4-1	SOAP Message Handlers Property	139
FIGURE 4-2	SOAP Message Handlers Dialog Box	140
FIGURE 4-3	SOAP Message Handlers Dialog Box	141
FIGURE 4-4	SOAP Message Handlers Set Properties Dialog Box	142
FIGURE 4-5	SOAP Message Handlers Add Property Dialog Box	142
FIGURE 4-6	SOAP Message Handlers Set Properties Dialog Box	143
FIGURE 4-7	Set SOAP Header Dialog Box	144

FIGURE 4-8	Set SOAP Header QName Dialog Box	145
FIGURE 4-9	Configure SOAP Actor Roles Dialog Box	146
FIGURE 4-10	Add SOAP Actor Role Dialog Box	147
FIGURE 4-11	Configure SOAP Actor Roles Dialog Box	147
FIGURE 5-1	XML Operation Calling Multiple Methods to Fulfill a Client Request	156
FIGURE 5-2	XML Operation Source Editor, Showing Complex Input	157
FIGURE 5-3	XML Operation Source Editor, Showing Complex Output	158
FIGURE 5-4	Input Document Elements Node	159
FIGURE 5-5	New XML Operation Dialog Box	163
FIGURE 5-6	Select Method Dialog Box	164
FIGURE 5-7	Collection of What? Dialog Box	165
FIGURE 5-8	Add Method to XML Operation Dialog Box	166
FIGURE 5-9	Select Method Dialog Box	167
FIGURE 5-10	Add Input Document Element Dialog Box	168
FIGURE 5-11	Input Document Element Properties Dialog Box	169
FIGURE 5-12	Output Document Element Properties Dialog Box	170
FIGURE 5-13	Input Document Element Properties (Default Value)	171
FIGURE 5-14	Method Parameter Source Dialog Box	173
FIGURE 5-15	Source Editor: (Excluding an Output Element)	176
FIGURE 5-16	Method Parameter Source Dialog Box	179
FIGURE 5-17	Resolve Object References Dialog Box	181
FIGURE 5-18	Resolve Object References Dialog Box With XML Operations	182
FIGURE 5-19	Map Parameters Dialog Box	186
FIGURE A-1	HTTP Basic Authentication Login Page	190
FIGURE A-2	Authentication Dialog Box	195
FIGURE A-3	Add Role Dialog Box	195
FIGURE A-4	Sun ONE AS Properties	196
FIGURE A-5	Mapped Security Roles Dialog Box	197
FIGURE A-6	Security Roles Dialog Box	200
FIGURE A-7	Edit Application Roles Dialog Box	201

FIGURE A-8	Application Server Administrative Tool: Manage Users	202
FIGURE A-9	Create Client Trust Store	207
FIGURE A-10	Set SSL Trust Store Option	208
FIGURE A-11	Application Server Administrative Tool: Manage Certificates	209
FIGURE A-12	List Client Trust Store	210
FIGURE A-13	Application Server Administrative Tool: HTTP Listeners	211
FIGURE A-14	Application Server Administrative Tool: HTTP Listener Properties	212
FIGURE A-15	Client Welcome Page: Security Links	213
FIGURE A-16	Client Page: Set Trust Store and Trust Store Password	213
FIGURE B-1	Deployment Descriptor Final Edit Dialog Box	217

Before You Begin

This book explains how to build and deploy web services and web service clients using the Sun™ ONE Studio 5, Standard Edition 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. Although almost all procedures use the interface of the Sun ONE Studio 5 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
```

A UNIX command might look like this:

```
% cd MyWorkDir/MyPackage
```

Before You Read This Book

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

- Java programming language
- Enterprise JavaBeans™ (EJB™) technology concepts
- JavaServer Pages™ technology syntax
- HTML syntax
- XML 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**
<http://java.sun.com/j2ee/blueprints>
- **Java 2 Platform, Enterprise Edition Specification**
<http://java.sun.com/j2ee/download.html#platformspec>
- **The J2EE Tutorial**
<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 Platform, Micro Edition)**
<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 5, Standard Edition 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 SOAP message handlers for your web service and client.

Chapter 5 explains how you create and edit XML operations, which are optional (deprecated) building blocks of a web service. It also provides a description of the tools you use for this job.

Appendix A describes how to secure a web service application, using HTTP basic authentication or HTTPS/SSL authentication and encryption.

Appendix B 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 <code>.cvspass</code> file. Use <code>DIR</code> 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.
<code>AaBbCc123</code>	Command-line variable; replace with a real name or value	To delete a file, type <code>DEL filename</code> .

Related Documentation

Sun ONE Studio 5 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.com`SM web site and from the documentation page of the Sun ONE Studio Developer Resources portal at <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 that is installed with the product on your local system or network.

- Release notes (HTML format)

Available for each Sun ONE Studio 5 edition. Describe last-minute release changes and technical notes.

- *Sun ONE Studio 5, Standard Edition Release Notes* - part no. 817-2337-10
- Getting Started guides (PDF format)

Describe how to install the Sun ONE Studio 5 integrated development environment (IDE) on each supported platform and include other pertinent information, such as system requirements, upgrade instructions, application server information, command-line switches, installed subdirectories, database integration, and information on how to use the Update Center.

 - *Sun ONE Studio 5, Standard Edition Getting Started Guide* - part no. 817-2318-10
 - *Sun ONE Studio 4, Mobile Edition Getting Started Guide* - part no. 817-1145-10
- Sun ONE Studio 5 Programming series (PDF format)

This series provides in-depth information on how to use various Sun ONE Studio 5 features to develop well-formed J2EE applications.

 - *Building Web Components* - part no. 817-2334-10

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. 817-2327-10

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. 817-2330-10

Describes how to build EJB components (session beans, message-driven beans, and entity beans with container-managed persistence or bean-managed persistence) using the Sun ONE Studio 5 EJB Builder wizard and other components of the IDE.
 - *Building Web Services* - part no. 817-2324-10

Describes how to use the Sun ONE Studio 5 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. 817-2332-10

Describes how to use the JDBC productivity enhancement tools of the Sun ONE Studio 5 IDE, including how to use them to create a JDBC application.
- Sun ONE Studio 5 tutorials (PDF format)

These tutorials demonstrate how to use the major features of Sun ONE Studio 5, Standard Edition:

 - *Sun ONE Studio 5 Web Application Tutorial* - part no. 817-2320-10

Provides step-by-step instructions for building a simple J2EE web application.
 - *Sun ONE Studio 5 J2EE Application Tutorial* - part no. 817-2322-10

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 you build is compliant with the Java 2 Platform, Micro Edition (J2ME™ platform) and conforms 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 in the Sun ONE Studio 5 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 5 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 that are used in this document.

Javadoc Documentation

Javadoc documentation is available within the IDE for many Sun ONE Studio 5 modules. Refer to the release notes for instructions on installing this documentation.

Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table.

Type of Documentation	Format and Location of Accessible Version
Books and tutorials	HTML at http://docs.sun.com
Mini-tutorials	HTML at http://forte.sun.com/ffj/tutorialsandexamples.html
Integrated example readmes	HTML in the example subdirectories of <i>sIstudio-install-directory/examples</i>
Release notes	HTML at http://docs.sun.com

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

<http://www.sun.com/service/contacting>

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 (817-2324-10) 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™ ONE Studio 5, Standard Edition 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 implementations of 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 all or part of the functionality of an application to other applications over an open network.

Application functionality exposed through a web service can be used within an enterprise, by related enterprises such as business partners, and by external enterprises that might or might not have an organizational or contractual relationship with the enterprise providing the 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.

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

Enterprises often need to accommodate elaborate processes, regulations, and relationships with external enterprises and government entities. A payroll system might be responsible for tax withholding, direct deposit to banks, and other functions that provide information to systems outside the enterprise. 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. 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 common example of a business-to-business (B2B) application is software 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. This is a more complex requirement than the typical business-to-customer application that might involve an online catalog and ordering process on the business side, and a browser on the customer side.

A Standardized Solution

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

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.

Web services technologies support distributed application development. Clients of different types can be created by developers based on a description of a web service's external interfaces, using a standardized web service definition language. This eliminates the need for client developers to have knowledge of the internal logic of a web service.

The Sun ONE Studio 5, Standard Edition IDE provides a user-friendly environment in which you can create web services and clients that meet these requirements.

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. This section provides a summary description of the three standards, followed by a more substantial description and references to the full specifications.

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. As a result, web service developers can publish their services, enabling a wide community of developers to create clients to access the services and incorporate their functionality into applications.

Implementations of web services generally involve passing Extensible Markup Language (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 data.

Note – The web services generated by the Sun ONE Studio 5, Standard Edition 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. With this information, developers can create SOAP clients 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, such as the location of the WSDL file that describes a web service's external interfaces.

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 provides the technical information to enable a developer to 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 web service and its provider. For example, potential users might want to know how committed the provider is to supporting the service and whether there is a charge for accessing the service.

A registry might have entries for multiple web services that implement the same external interfaces (the same WSDL), possibly differing in runtime characteristics such as performance or availability, or provided by different vendors on different terms. Registry search facilities can help a client developer to choose the best web service to access for a given application.

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.

One of the issues to consider in using a web service that you find through a UDDI registry search is the reputation of the service provider. This issue does not originate with web services. The question of trustworthy sources arises with network security technology in which digitally signed certificates are exchanged. Certain companies are widely recognized as certificate-granting authorities. The same question arises with older technology, as in the credit card industry, or in relying on government institutions such as the National Bureau of Standards.

Web services technologies can enhance the capabilities of widely-recognized industry or governmental organizations to make available complex, standardized tables of technical information to interested parties.

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.

Web Services Interoperability Organization

The Web Services Interoperability Organization (WS-I) is an open industry association whose purpose is to promote web services interoperability across platforms, applications, and programming languages. This is achieved through the specification of *profiles* which define the precise rules to be used by conforming services. A profile includes web services specifications at designated release levels, together with guidelines for how to use the features in the specifications. A WS-I profile guides the work of web services developers in using lower-level specifications such as SOAP, WSDL, and UDDI.

WS-I works with other standards organizations, including the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). WS-I, founded in February 2002, is composed of members from over 150 companies, and is governed by a Board of Directors, of which Sun Microsystems is a member. You can find more information about WS-I, including their published documents, at the WS-I web site: <http://www.ws-i.org>.

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. The numbered steps in FIGURE 1-1 can be described as follows:

The first three steps in the process occur during development. The fourth step occurs at runtime, when the deployed client communicates with the deployed web service.

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 deploying the web service and making it available over a network.

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

Many search criteria are possible. 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. In a large development project there might be several versions of a web service, and the client developers might search for a particular version.

The UDDI registry provides a URL for the WSDL file and a URL for the runtime web service. The WSDL file itself is not stored in the UDDI registry.

3. Company R gets the WSDL file and creates a client component capable of interacting with the web service. Company R incorporates the client into an application and deploys the application.
4. Company R runs the application. At runtime, the client component binds to the web service and executes SOAP requests, passing XML data over HTTP.

Note – The UDDI registry is not used at runtime. It is a development facility.

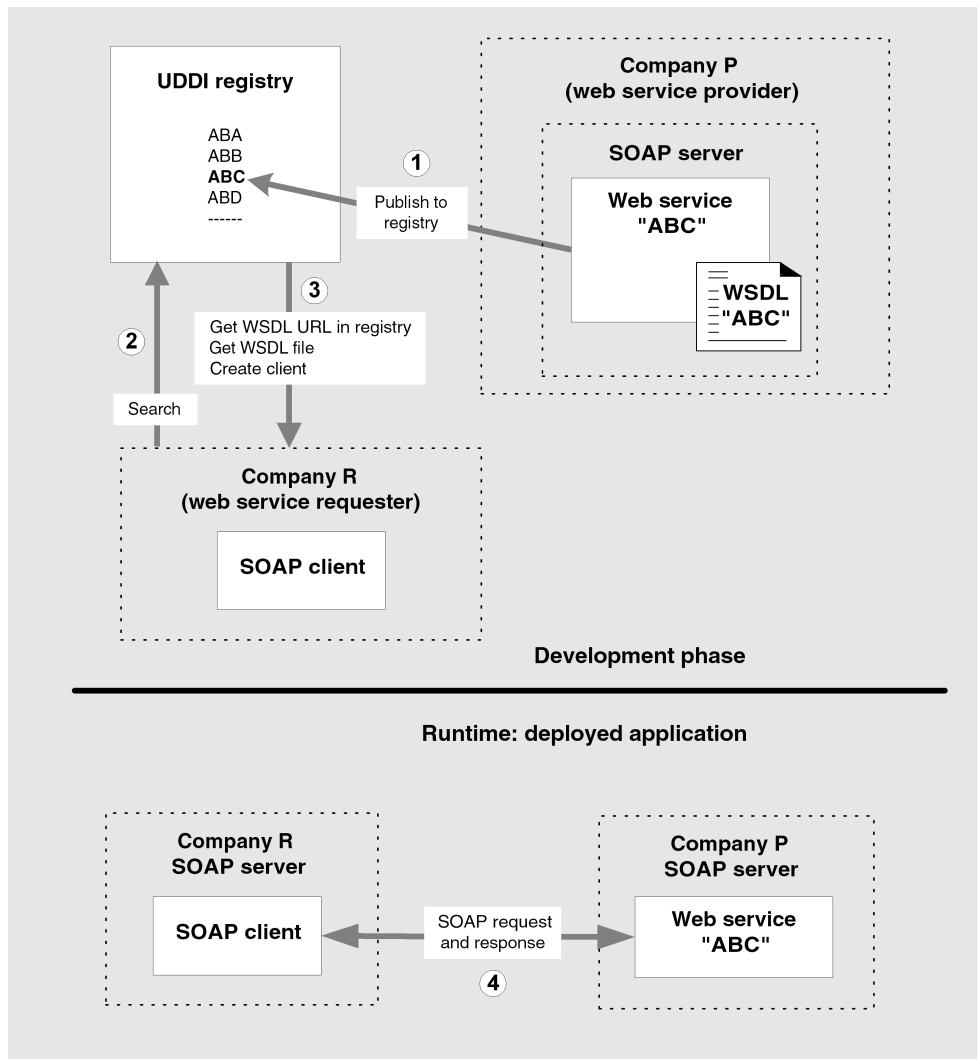


FIGURE 1-1 Publishing and Using a Web Service

The process includes the following technical tasks:

- Company P gets permission from the UDDI registry host to write to the registry. Company P establishes itself in the registry by adding business entries that include name, locator information, and business categorization information.
- 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 *tModel* (technical model).

- 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.
- 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.
- Company R can use this information to create and customize a variety of clients for a given web service.

Sun ONE Studio 5, Standard Edition Web Services

The 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)

The following features are new in this release.

- Stateful web services and clients
- Security: HTTP Basic Authentication
- Security: HTTPS/SSL Authentication and Encryption
- SOAP message handlers

- Attachments
- Client support: WSDL taglib generation, MIDP enhancements

You can create a web service based on the methods of an existing business component (a *bottom-up* development model) or based on a WSDL file (a *top-down* development 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.

When you build a web service or client, the IDE generates a supporting structure of classes and other objects. You can use the IDE to customize the generated objects. For example, you can customize a generated client to better meet the needs of your production application and end-users.

Web Services Architectures

When you create a web service, the IDE gives you a choice of two architectures, called *multitier* and *web-centric*. The default multitier architecture is illustrated in FIGURE 1-2. The web-centric architecture is illustrated in FIGURE 1-3. See “Creating a JAX-RPC Web Service From Java Methods” on page 42 for procedures.

- The multitier architecture is recommended if your business logic is implemented in an EJB component.
- The web-centric architecture is recommended if your business logic is implemented in Java classes and your application does not require an EJB container.

Multitier Architecture

FIGURE 1-2 illustrates the main components of a web service application developed with the IDE using the multitier architecture. The web service is a logical entity that at runtime spans the web and EJB containers of your J2EE application server.

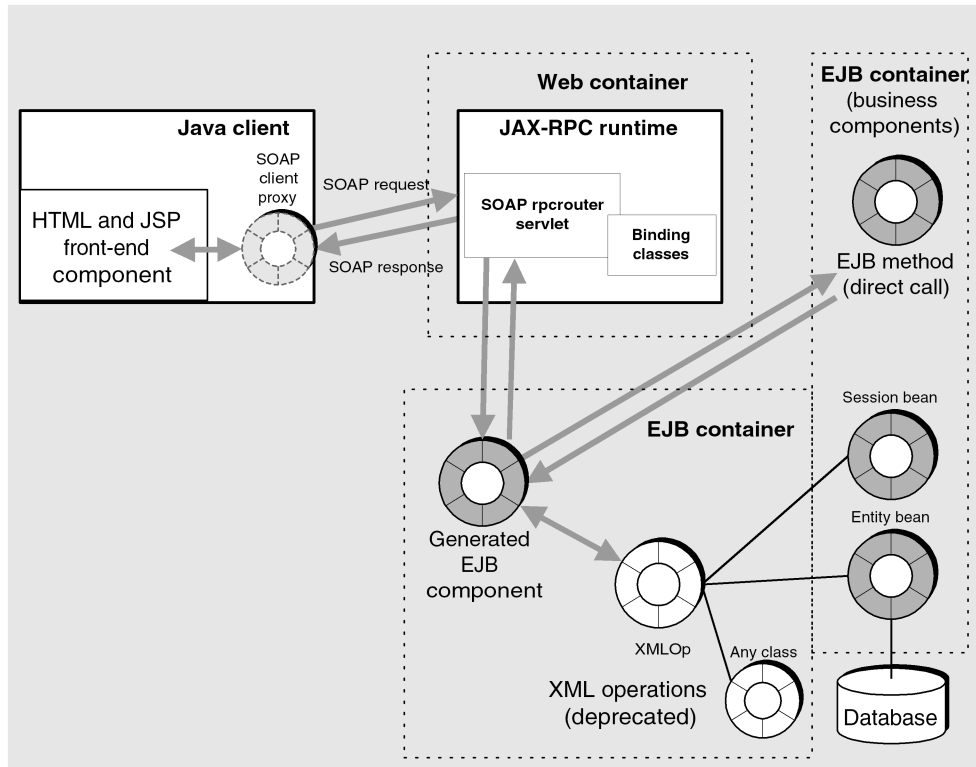


FIGURE 1-2 Web Service and Client (Multitier Model)

On the client side, there is a *SOAP client proxy*. This is a set of Java classes that use 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* component that makes calls to the SOAP client proxy and processes or displays the value returned by the web service. This component might consist of HTML and JSP pages, or it could be a Java Swing component, another kind of client, or another service. The HTML browser, typically used to access the client, is not illustrated.

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 client's 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 (optionally) invokes an XML operation.

A multitier web service cannot run in a web container, because the generated EJB component requires an application server. Moreover, a multitier web service can only call methods on business components running in the EJB container of an application server.

Web-centric Architecture

FIGURE 1-3 illustrates the main components of a web service application developed with the IDE using the web-centric architecture. 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.

A web-centric web service can call methods on web tier business components and EJB business components, thus providing more flexibility than a multitier web service. If your web-centric web service uses only web tier business components, you do not need an application server. A web server is sufficient.

If your application uses only EJB business components, choose the multitier architecture for scalability. A generated support structure that runs in the same container as the business EJB components is more efficient.

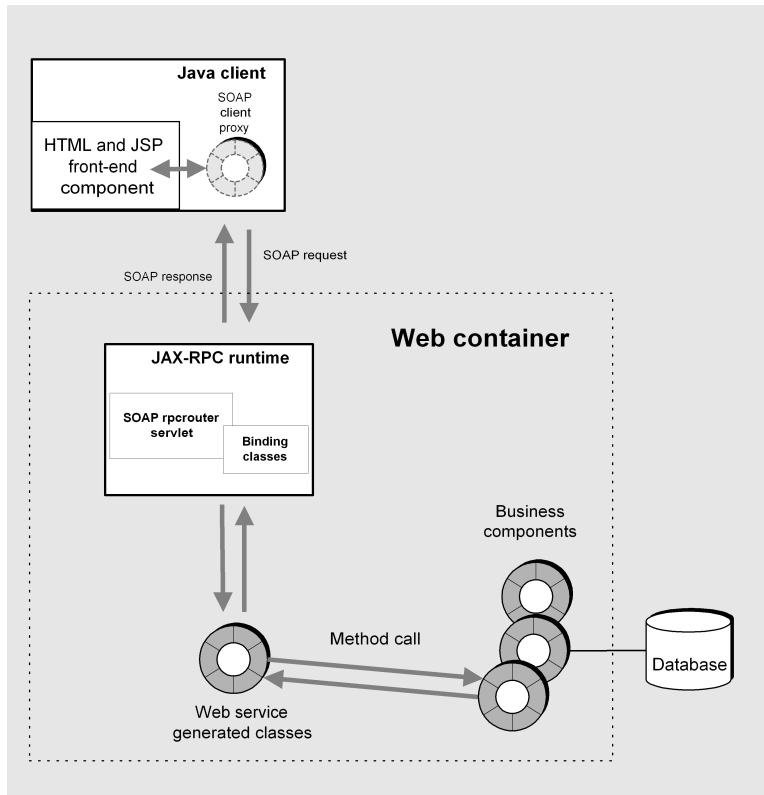


FIGURE 1-3 Web Service and Client (Web-centric Model)

Deployment Architecture

The deployment model depends on the web service architecture and the type of business components:

- For the multitier architecture, the IDE creates a `.ear` file for deployment to an application server.
- For the web-centric architecture with EJB business components, the IDE creates a `.ear` file for deployment to an application server.
- For the web-centric architecture with only web business components, the IDE creates a `.war` file for deployment to a web server.

See “Assembling and Deploying a Web Service” on page 53 for more information about assembly and deployment.

Software Versions

Web standards and technologies are still evolving. The Sun ONE Studio 5, Standard Edition 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/>
- *kSOAP 1.0 (a SOAP API suitable for the J2ME Platform)*
<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, Standard Edition
http://sun.com/software/products/appsrvr/home_appsrvr.html
- Java™ 2 Software Development Kit, Standard Edition (J2SE™ SDK), version 1.4.1_02 or higher

Note – The IDE generates JAX-RPC web services. The IDE enables you to choose either of two client-types: JAX-RPC or kSOAP. If you have an old Apache SOAP service or client, the IDE will automatically convert it to JAX-RPC.

XML Operations (Deprecated)

This feature is deprecated and might not be supported in future releases of the IDE. See Chapter 5 for further information.

Building a Web Service

This chapter describes the tools and procedures that you can use to develop a web service. 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 enterprise bean or Java classes.
- In the *top-down* approach, you use the IDE to generate both the web service and an EJB session bean from WSDL source. Subsequently, you add business logic to the dummy methods of the generated session bean.

Both approaches are covered in this chapter.

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.

This chapter includes the following tasks:

- Creating a JAX-RPC web service from existing Java methods
- Adding operations to a web service
- Resolving references to runtime objects
- Adding environment entries
- Creating a JAX-RPC web service from WSDL source
- Generating runtime classes
- Assembling and deploying a web service
- Creating a test client
- Testing a web service
- Creating a stateful web service and client
- Working with UDDI registries
- Using arrays and collections

Creating a JAX-RPC Web Service From Java Methods

This technique can be thought of as a *bottom-up* model for creating a web service. The starting point is an existing business component, and the web service is built to implement remote calls to business methods.

To create a new web service from Java methods:

1. To open the New Web Service wizard, go to the Explorer, right-click the Java package in which you want to create the web service, and choose New → All Templates → Web Services → Web Service.

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

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

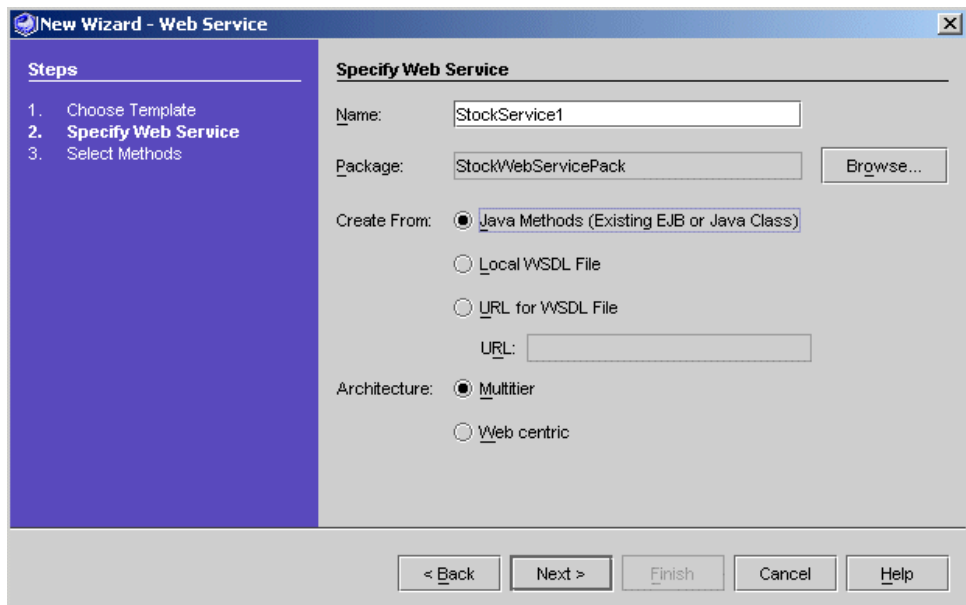


FIGURE 2-1 New Web Service Wizard

Note – The wizard has radio buttons that enable you to select the web service architecture: Multitier (the default) or Web centric. The distinction between these architectures is explained in “Web Services Architectures” on page 35. 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 Source” on page 48.

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.

Note – Since the IDE will generate many files to implement your web service, it is good practice to place the web service in a package of its own.

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 chooser. Select one or more methods and click Finish in the chooser to add method references to your web service. Then click Finish to create the web service.

You can also add method references after creating a web service (see “Adding Operations to a Web Service” on page 45).

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 55).

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 with the SOAP RPC URL highlighted.

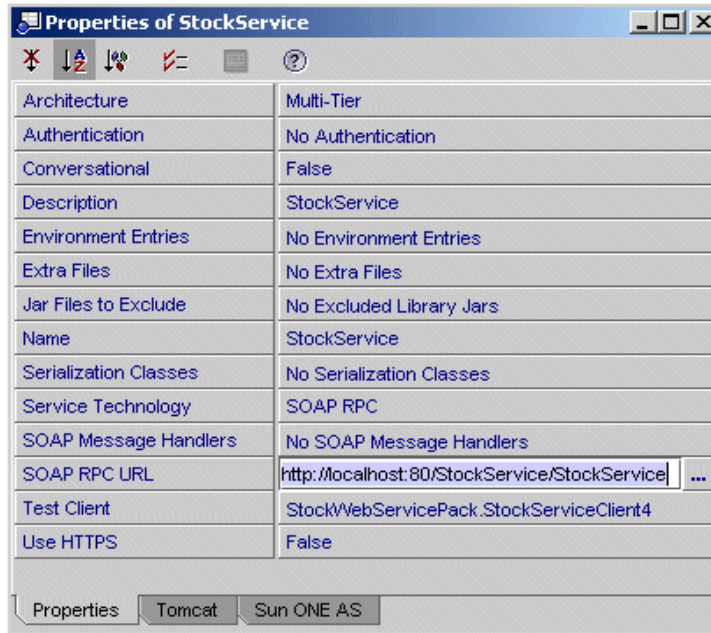


FIGURE 2-2 Web Service Properties With Context Root Set to StockService

Developing XML Operations (Deprecated)

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 how to develop XML operations, see Chapter 5.

Adding Operations to a Web Service

You must have available in the IDE an enterprise bean or other business component whose methods you want to expose in the web service. Adding an *operation* to your web service causes the IDE to generate code that issues a remote method call. The logical term, operation, refers to the generated mechanism that implements the method call.

Note – Web service operations are created from EJB and Java class methods, or from XML operations (deprecated).

To add an operation to a web service:

1. **In the Explorer, right-click the web service and choose New Operation.**

A file chooser is displayed.

2. **Select the desired business method (or XML operation) and click OK.**


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 from 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

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

Operations and XML operations are organized as follows:

- Operations based on EJB or Java methods are under the `Methods` node.
- XML operations are under the `XML Operations (deprecated)` node.

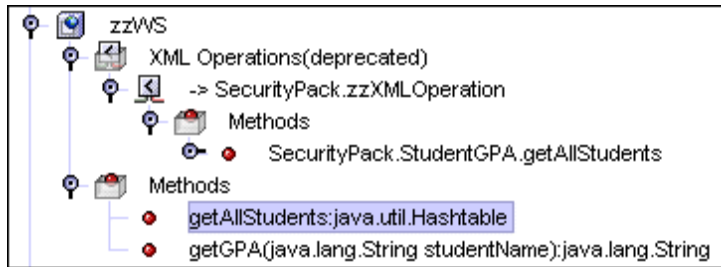


FIGURE 2-4 Web Service Operations

The IDE prefixes references with the names of the packages containing the referenced objects. The IDE adds the arrow symbol (->) before the names of the XML operation references.

Note – You can also add operations 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.

You can direct the IDE to add external directories or files to a web service. This feature supports the use of attachments, property files and other resources, such as .gif, .jpg, xml, .property, and .jar files.

To add external directories or files:

1. **Right click the web service node in the Explorer and choose Properties.**
2. **Click the External Files property value, illustrated in FIGURE 2-2, and click the ellipsis (...) button.**

The IDE displays the External files dialog box.

3. **Click Add.**
A chooser appears.
4. **Select the desired directories or files in the chooser, click OK, and click OK in the External files dialog box.**

Deleting Operations From a Web Service

To delete an operation:

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

The operation is deleted.

Resolving References to Runtime Objects

For information on this task, see “Instantiating Objects and Resolving References” on page 180. This step is necessary only if you are using XML operations and 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, and click the ellipsis (...) button.**

The Environment Entries dialog box is displayed, as illustrated in FIGURE 2-5.

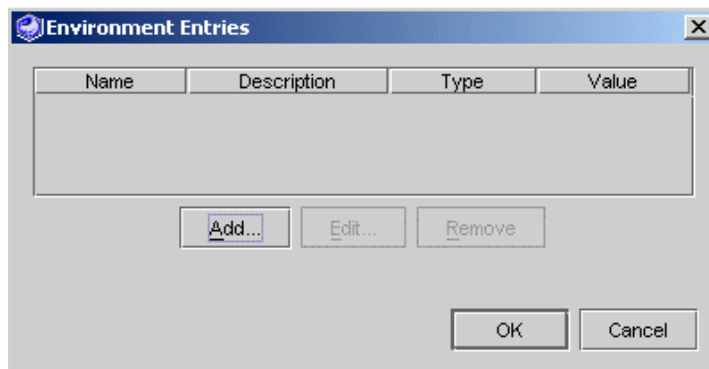


FIGURE 2-5 Environment Entries Dialog Box

3. **Click Add.**

The Add Environment Entry dialog box is displayed, as illustrated in FIGURE 2-6.

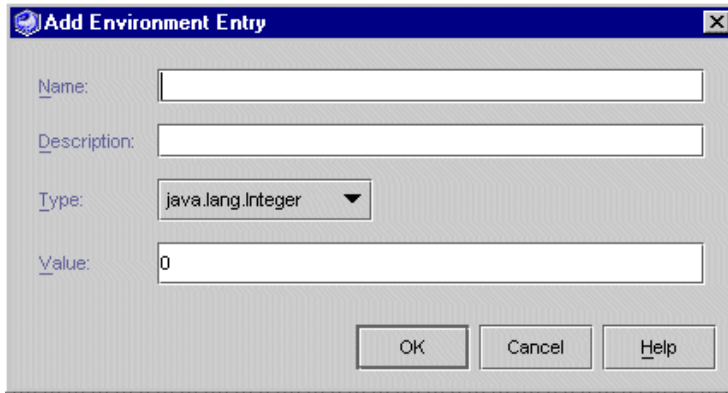


FIGURE 2-6 Add Environment Entry Dialog Box

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.

Creating a JAX-RPC Web Service From WSDL Source

This technique can be thought of as a *top-down* model for creating a web service. The starting point is a WSDL file that describes the external interfaces of the web service. The IDE creates the web service and the stub of an EJB session bean with dummy business methods that match the specifications in the WSDL. You must provide code to implement the business methods in the EJB session bean.

The top-down model enables developers to create web services with different implementations of a common standard. The top-down model can be used by a company that deals with many suppliers, each of which might have its own computerized order and inventory systems. For example, an auto manufacturer might develop an application based on web services, in which each parts supplier has an implementation that works for its business.

To create a new web service from WSDL:

1. **To open the New Web Service wizard, go to the Explorer, right-click the Java package in which you want to create the web service, and choose New → All Templates → Web Services → Web Service.**

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

The New Web Service wizard is illustrated in FIGURE 2-7.

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.

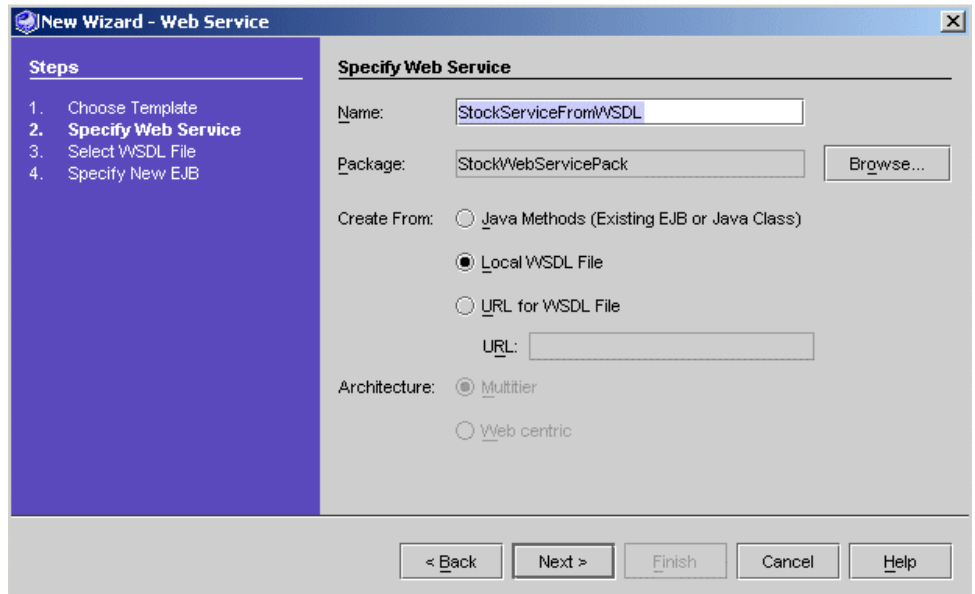


FIGURE 2-7 New Web Service Wizard

2. **Select the radio button for Create From Local WSDL File, click Next, and select the desired WSDL file from the chooser, as illustrated in FIGURE 2-8.**

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

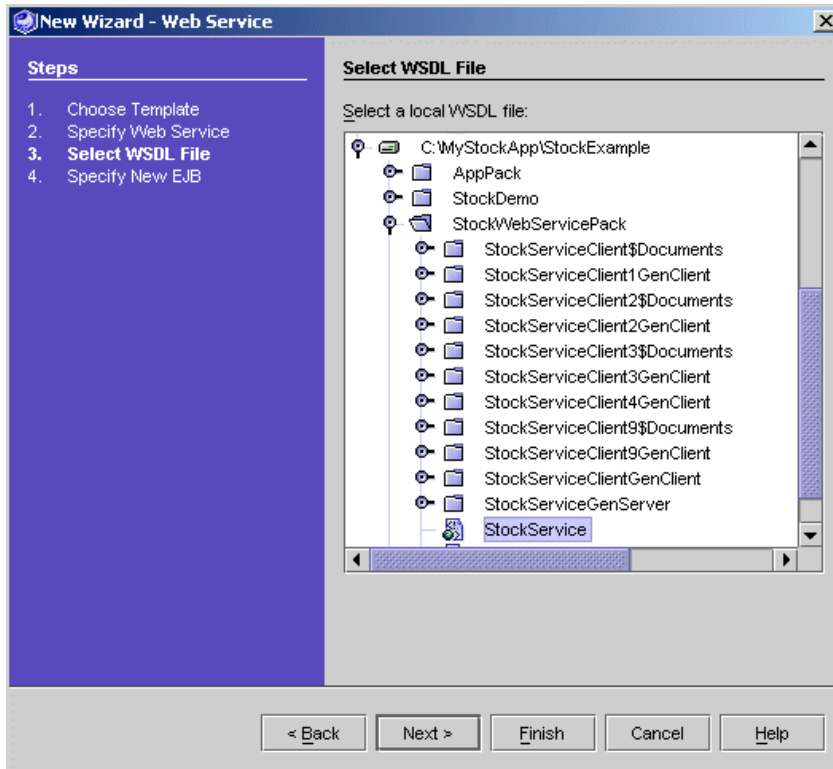


FIGURE 2-8 New Web Service Wizard: Select WSDL File

The IDE displays the Specify New EJB window, as illustrated in FIGURE 2-9.

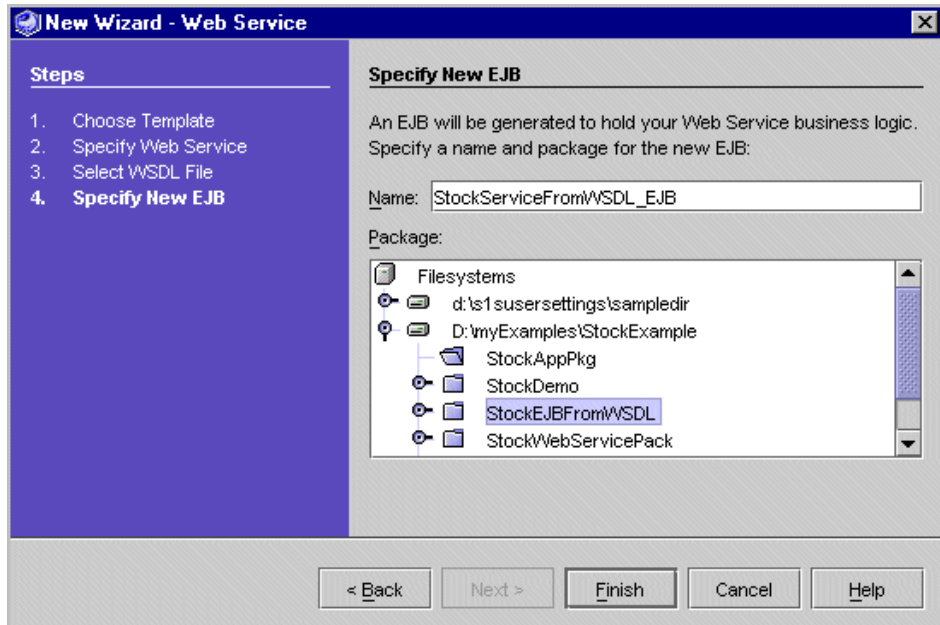


FIGURE 2-9 New Web Service Wizard: Specify New EJB

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

The IDE creates the new web service and the new EJB component, placing each in the package that you designated in the wizard. See FIGURE 2-10 for an Explorer view of a web service and EJB component created from a WSDL file.

Note – Since the IDE will generate many files to implement your web service, it is good practice to place the EJB session bean and web service in different packages.

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 the EJB classes.

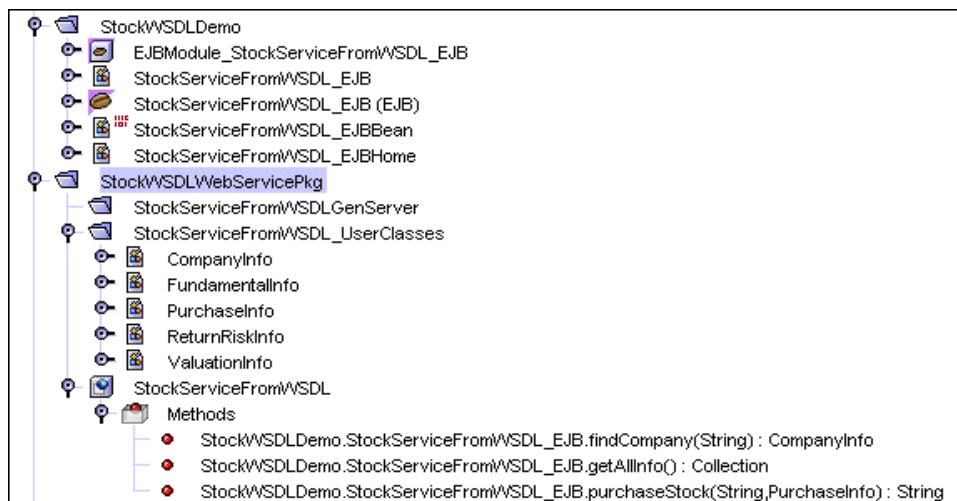


FIGURE 2-10 Web Service and EJB Nodes Generated From WSDL: Explorer View

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 5.

To generate a web service's runtime classes:

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

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

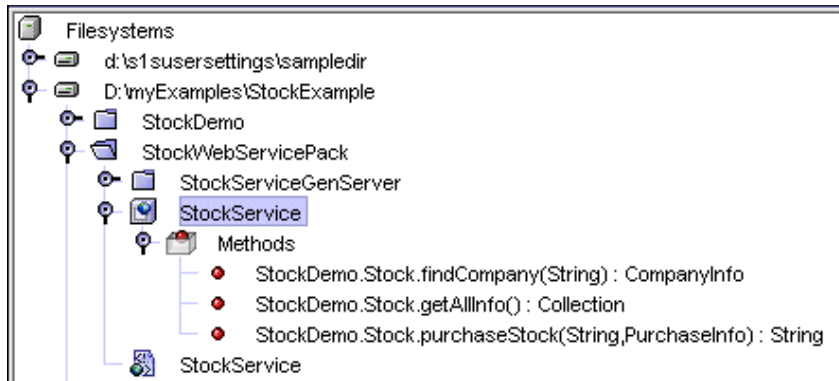


FIGURE 2-11 Web Service Hierarchy of Nodes

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 69.

Assembling and Deploying a Web Service

The explanation in this section assumes that any 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 Sun ONE Application Server 7. 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 and does not reference an EJB component, the web service node has the Deploy and Execute menu items enabled in the Explorer.

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

The IDE assembles and deploys the WAR file to the IDE's default web tier application server. See "Testing a Web Service" on page 58 for information about testing your web service.

You might need to edit the WAR file to override the default configuration, for example, to change the Context Root property to match the SOAP RPC URL property of your web service. You might need to provide the WAR file to someone else to do the deployment, for example, in a controlled production environment. In these situations, you can use the following procedure (modified as necessary to match your situation):

1. **Right-click the web service node and choose Export 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 5 Programming series for instructions on how to do this.

The following sections apply to a web service that is assembled and deployed as a J2EE application.

Assembling the J2EE Application

The procedure in this section applies to web services that use EJB business components.

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 → All Templates → 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 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 Module 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.**

In most cases this step is not necessary because the IDE sets a default Web Context property (the name of the web service) for the web service WAR file and the web service. The value of this property is part of the URL used to access your web service. The WAR file value must match the *context root* value embedded 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 is `StockService`, matching 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 `StockService`

- 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:

- a. Right-click the J2EE application node and choose Properties.**
- b. Click the Application Server property value, and click the ellipsis (...) button.**

The IDE displays a dialog box, as illustrated in FIGURE 2-14.
- c. Select any available application server instance.**

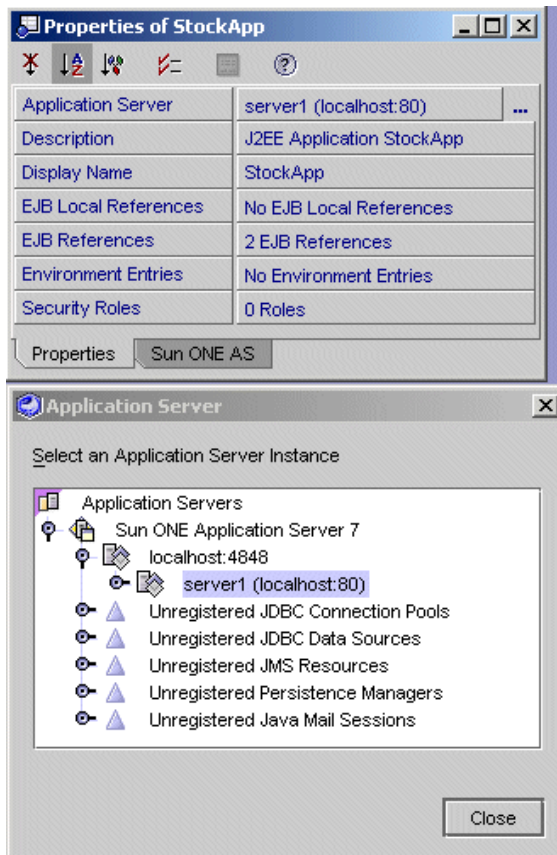


FIGURE 2-14 Application Server Instance for Deployment

Deploying the J2EE Application

To deploy your web service application to the application 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 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.

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

Testing a Web Service

The IDE provides comprehensive support, fully described in Chapter 3, for creating web service clients from any of the following sources:

- A local IDE web service
- A WSDL file
- A web service published in a UDDI registry

See “Creating a Client From a Local Web Service” on page 91 for the procedure. An important special case involves testing your web service with a *default test client*.

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 – The IDE automatically packages the default test 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 it the default test client for your web service.**
2. **Assemble your web service into a J2EE application.**

See “Assembling the J2EE Application” on page 55 for the procedure.

- 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 asks if you want to create a test client.

Alternatively, you can first deploy the web service application, and 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 web container, and executes the client.

Testing a Web-Centric 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 web server.**
 - To view or change the SOAP RPC URL property, right-click the web service node in the Explorer and choose Properties.
 - To find the server port number, expand the Installed Servers node in the Explorer's Runtime tab.

- 3. Right-click the web service node and choose Deploy.**

The IDE assembles a WAR file, deploys it to the default web server, and runs the web service.

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

The IDE starts the default test client and displays its welcome page in a browser.

Iterative Testing

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

Creating a Stateful Web Service

The IDE enables you to create an application in which a stateful web service maintains a session with a stateful client, and within the session a series of *conversations* can take place.

A conversation is a set of web service operations that are executed within a single session. A conversation is begun when a method marked as a *conversation initiator* is executed. A conversation ends when a *conversation terminator* method is executed. Other methods executed within the bounds of the conversation are members of the session and share session data.

Conversational components must be designed so that the initiator method must be executed before any other methods. An initiator method can also be used to create resources for a session. A terminator method should release any resources not previously released in a session, to prevent resource leakage.

Multiple requests from the same client to a stateful web service are routed to the same instance of the business component. A client can make multiple calls to web service methods and data is preserved through the session.

For a web service with the multitier architecture, the business component is a stateful EJB session bean. For a web service with the web-centric architecture, the business component can be a Java object.

The procedure in this section assumes that your business component is a stateful EJB session bean.

Setting Up the Business Component

You can make a session bean stateful in two ways.

When you create the bean:

- **Click the `Stateful` radio button in the wizard, as illustrated in FIGURE 2-15.**

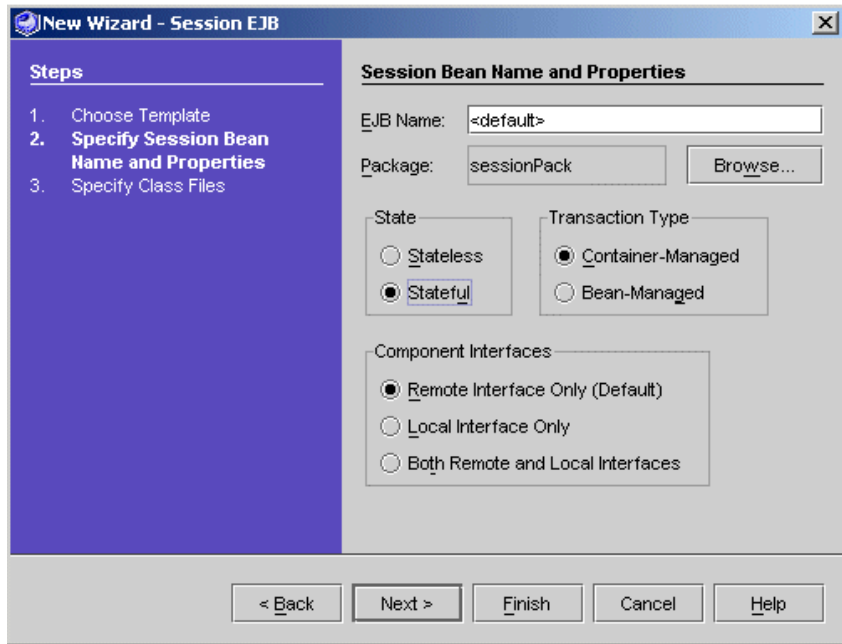


FIGURE 2-15 Stateful Session Bean: Wizard

If the bean already exists:

1. **Right-click the logical EJB node and choose Properties.**
2. **Set the Bean Type property to `Stateful`, as illustrated in FIGURE 2-16.**

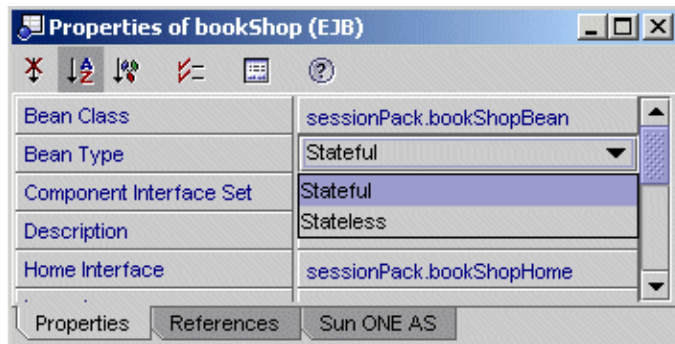


FIGURE 2-16 Stateful Session Bean Bean Type Property

You must designate business methods on the session bean that the web service client can use to start and stop a conversation. It is good design to have a method that creates resources needed for the conversation and starts the conversation, and another method that destroys resources which are no longer needed and stops the conversation.

Note – The method names are arbitrary. The example in this section uses method names `startOrder` and `submitOrder`.

The following procedure creates two methods that start and stop a conversation:

- **Modify the session bean class to add methods named `startOrder` and `submitOrder`.**

To do this in the IDE:

1. **Expand the logical EJB node in the Explorer, as illustrated in FIGURE 2-17.**

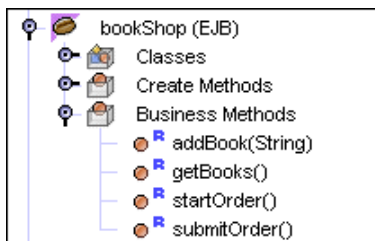


FIGURE 2-17 Explorer View of Business Methods

2. **Right click the Business Methods node and choose `Add Business Method`.**

The Add New Business Method dialog box opens, as illustrated in FIGURE 2-18.

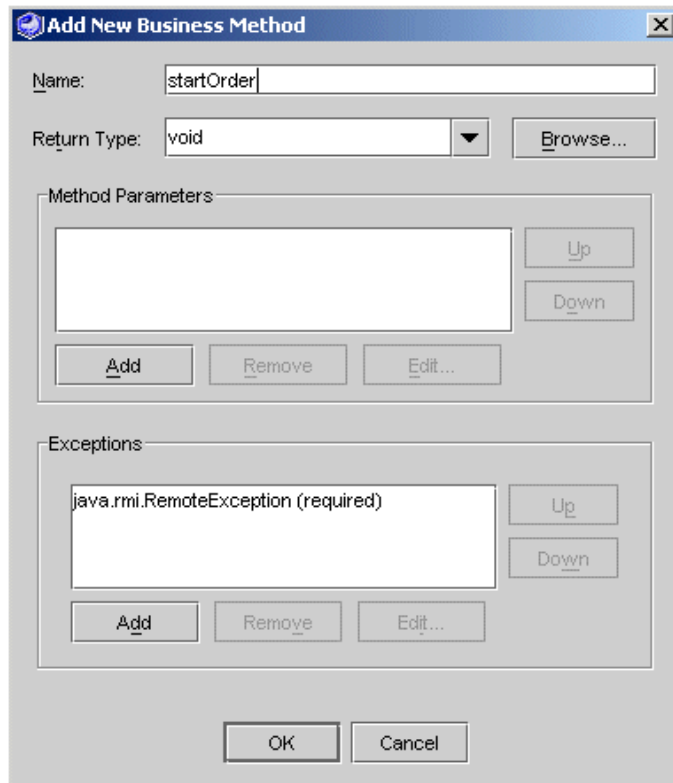


FIGURE 2-18 Add New Business Method Dialog Box

3. Type a value for Name and click OK to add a method with default properties.

The method names in this section are for illustration only. You can use whatever method names you prefer.

When you complete this procedure, the session bean has two new methods:

```
public void startOrder() {}  
public void submitOrder() {}
```

Suppose that the purpose of each conversation is to create and submit an order for one or more books. The `startOrder` method can initialize a shopping cart object with customer information (including credit card) and a `Vector` to contain book objects. The `submitOrder` method can submit a purchase request, passing customer information and the books to be ordered, and release the shopping cart object before terminating the conversation.

Setting Up the Web Service and Test Client

Suppose that you have created your web service and added business methods, including the conversation initiator and terminator methods. If your method names are different, modify the procedure accordingly.

To make the web service conversational, you must set properties for the web service and for the methods that initiate and terminate conversations:

1. **Right-click the web service node and choose Properties.**
2. **Set the value of the Conversational property to `True`, as illustrated in FIGURE 2-19.**

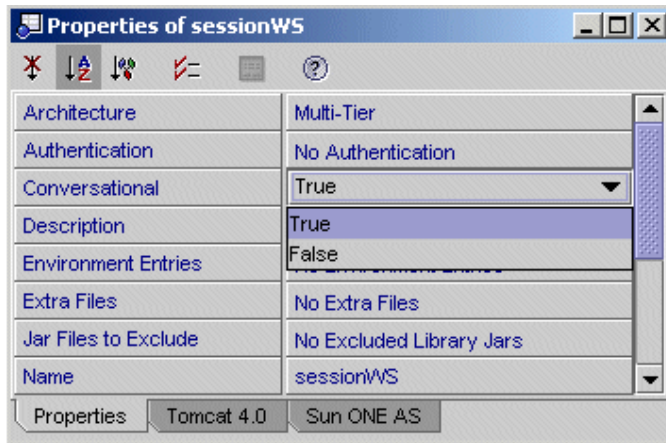


FIGURE 2-19 Web Service Conversational Property

Set properties for the methods that initiate and terminate conversations.

1. **Expand the web services node in the Explorer , expand the Methods node, right-click the `startOrder` method, and choose Properties.**
2. **Set the Conversation Initiator property to `True` (see FIGURE 2-20).**

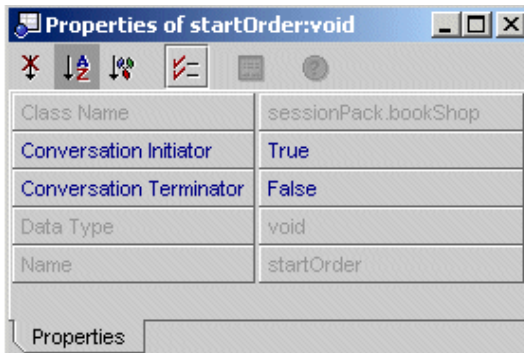


FIGURE 2-20 Web Service Conversation Initiator and Terminator Method Properties

Note that the Conversation Terminator property value should be `False` (the default).

3. **Expand the web services node in the Explorer , expand the Methods node, right-click the `submitOrder` method, and choose Properties.**
4. **Set the Conversation Terminator property to `True`.**

Note that the Conversation Initiator property value should be `False` (the default).

At this point you can generate the web service and create a test client, as described in “Testing a Web Service” on page 58.

The IDE creates the test client with the same Conversational property value as the web service. If you are using an existing test client, make sure its Conversational property value is `True`.

To set the Conversational property:

1. **Right-click the test client node and choose Properties.**
2. **Set the value of the Conversational property to `True`, as illustrated in FIGURE 2-21.**

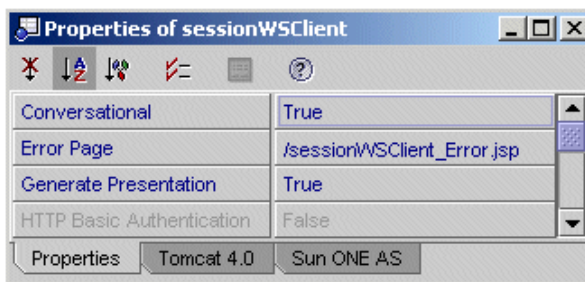


FIGURE 2-21 Web Service Test Client Conversational Property

When you generate the test client, the IDE creates:

- JSP tags to invoke the conversation initiator and terminator methods
- HTML for a client welcome page that has Invoke buttons for the web service's methods, including the conversation initiator and terminator methods

To see the HTML welcome page and JSP nodes:

1. **Expand the test client node in the Explorer, and expand the Generated Documents node.**
2. **Right-click a node and choose Open.**

For more information about the test clients generated by the IDE, see Chapter 3.

Note – The WSDL standard does not provide for statefulness. Therefore, if you generate a client from a UDDI registry or from WSDL, the client's Conversational property will be set to the default value, `False`. If you know that the web service is stateful, you can manually set the Conversational property to `True`.

Testing a Stateful Web Service

Suppose that you have generated your web service and test client, and that you have assembled and deployed your application. (See “Assembling and Deploying a Web Service” on page 53 and “Testing a Web Service” on page 58.)

See “Testing a Web Service” on page 58 for the procedure to execute your application with the test client.

When you execute your application, the welcome screen appears. You should see Invoke buttons for your web service's business methods, including the conversation initiator and terminator methods, as illustrated in FIGURE 2-22.

The image shows a web interface for testing a stateful client. It is divided into four horizontal sections, each representing a different method to be invoked. The first section, titled `void addBook(String string_1)`, includes a text input field labeled `string_1:` and two buttons: `Invoke` and `Reset`. The second section, titled `Vector getBooks()`, has a single `Invoke` button. The third section, titled `void startOrder()`, also has a single `Invoke` button. The fourth section, titled `void submitOrder()`, has a single `Invoke` button.

FIGURE 2-22 Stateful Test Client Welcome Screen

The exact sequence of steps to test the statefulness of your application depends on what the business methods do. For the example illustrated in FIGURE 2-22, you can test statefulness as follows:

- 1. Click `Invoke` for the `startOrder` method.**
- 2. Use the `addBook` method to add several books, and use the `getBooks` method after each add to see if each new book is appended to the `Vector`.**
- 3. Click `Invoke` for the `submitOrder` method.**

Now try the same procedure but without using the `startOrder` method. You should find that the added books are not displayed.

Note – In the simple test clients generated by the IDE, based on HTML and JSP pages, it is impossible to ensure that an end-user will always click the button to invoke the terminator method and release resources at the end of a conversation. However, a more robust client designed for production use can force the cleanup step regardless of how the end-user exits the application.

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. Before you can publish your web service to a UDDI registry, you must complete the following tasks:

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

The IDE publishes the WSDL automatically when you deploy your web service.

- Configure the IDE's registry options.

This establishes default values that greatly reduce the amount of information that you must enter when publishing a web service to a UDDI registry.

This section covers the following topics:


- Generating WSDL
- Managing UDDI registry options
- Gaining access to an external UDDI registry
- Publishing your web service to a UDDI registry
- Using the internal UDDI registry

Note – Client development is described in Chapter 3. See “Creating a Client From a UDDI Registry” on page 113 for information about how to find a web service in a UDDI registry and how to generate a client that can access the web service.

Generating WSDL

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*.

If you use the IDE to generate and deploy your web service, and to publish the web service to a UDDI registry, the IDE manages the WSDL for you automatically.

Note – 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 a WSDL File” on page 112.

Managing UDDI Registry Options

The IDE stores a list of known UDDI registries and information (such as URLs, userid, and password) needed to access each registry. The IDE also stores 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 – Set default values for registry information before you use the IDE wizards to access UDDI registries. By setting defaults, you avoid repetitive entry of values in the wizards.

To manage UDDI registry options:

- Choose Tools → Options → Distributed Application Support → Web Service Settings → UDDI Registries from the IDE’s main window.

The Options dialog box is displayed, as illustrated in FIGURE 2-23.

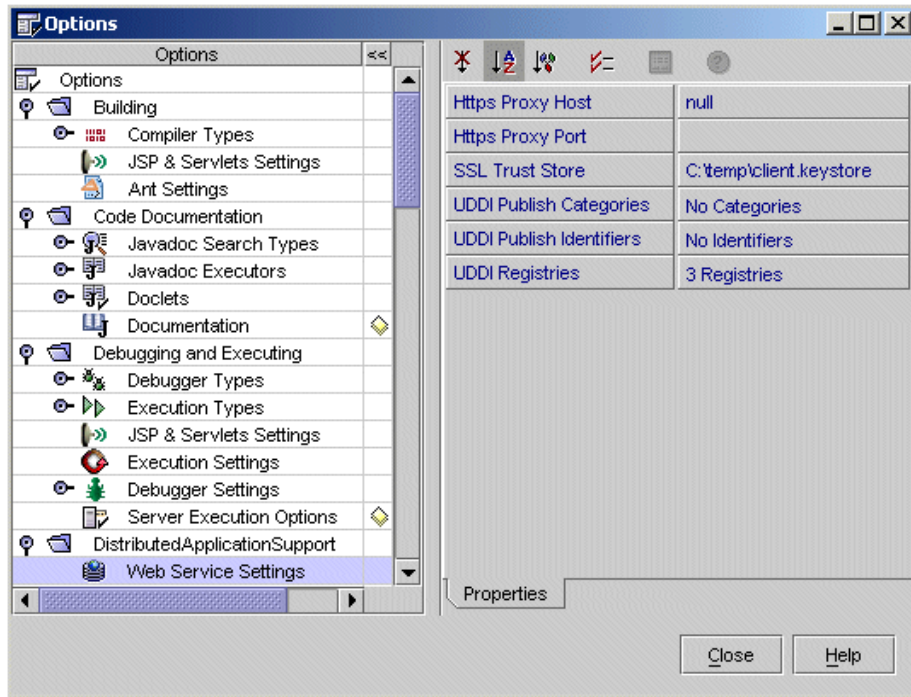


FIGURE 2-23 UDDI Registries Options

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:

1. Click the **Publish Categories** value in the **UDDI registries Options** dialog box (see FIGURE 2-23).
2. Click the **ellipsis (...)** button that appears.

The Publish Categories dialog box appears, as illustrated in FIGURE 2-24.

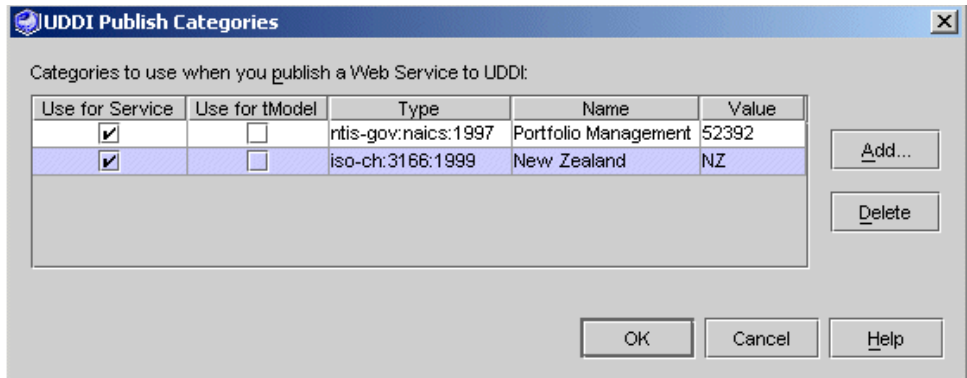


FIGURE 2-24 UDDI Publish Categories Dialog Box

You can edit default publish categories as follows:

- To set a category as a default for services or tModels when you publish them to a registry, select the Use for Service checkbox or the Use for tModel checkbox.
- To delete a category, select the category and click Delete.
- To add a category, click Add.

The UDDI Categories (taxonomies) chooser appears, as illustrated in FIGURE 2-25. The categories are industry standards.

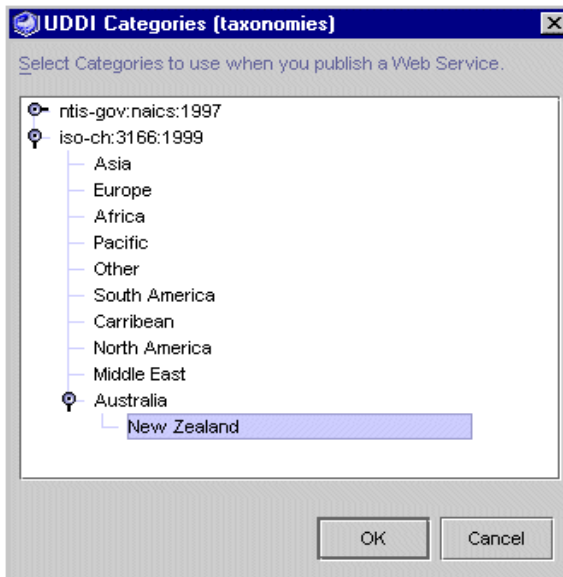


FIGURE 2-25 UDDI Categories (Taxonomies)

- Expand the nodes, select the desired category, and click OK. 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.

The selected categories appear in the Publish Categories dialog box.

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.

To set default publish identifiers:

1. Click the **Publish Identifiers** value in the **UDDI registries Options** dialog box (see FIGURE 2-23).
2. Click the **ellipsis (...)** button that appears.

The Publish Identifiers dialog box appears, as illustrated in FIGURE 2-26.

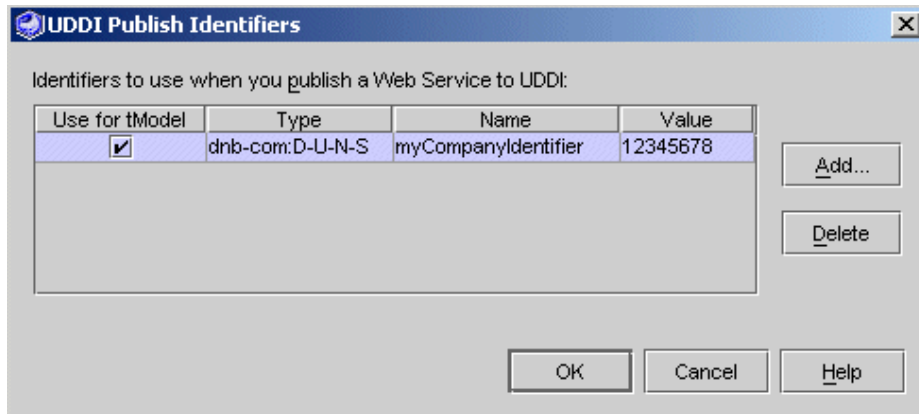


FIGURE 2-26 UDDI Publish Identifiers Dialog Box

You can edit default publish identifiers as follows:

- To set an identifier as a default for tModels when you publish them to a registry, select the Use for tModel checkbox.
- 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 2-27. The identifier types and tModel UUIDs are industry standards.

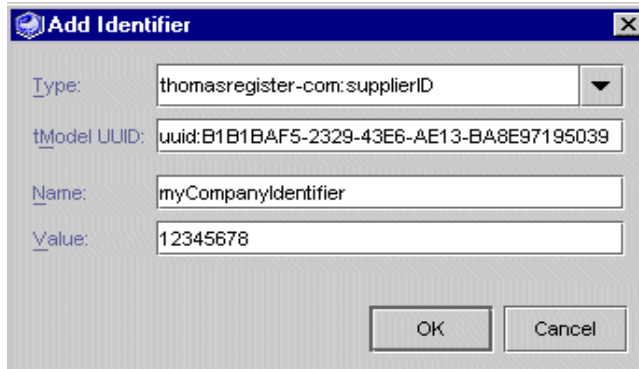


FIGURE 2-27 UDDI Add Identifier Dialog Box

1. Select one of the standard choices for Type.

The IDE displays the tModel UUID for the selected type. Alternatively, you can enter the name and Model UUID (the registry key) of a user-defined type.

2. Enter Name and Value, and click OK.

The IDE returns you to the Publish Identifiers dialog box.

3. 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 2-23). The Property Editor dialog box is displayed, as illustrated in FIGURE 2-28.

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, enter values for Name and Password.

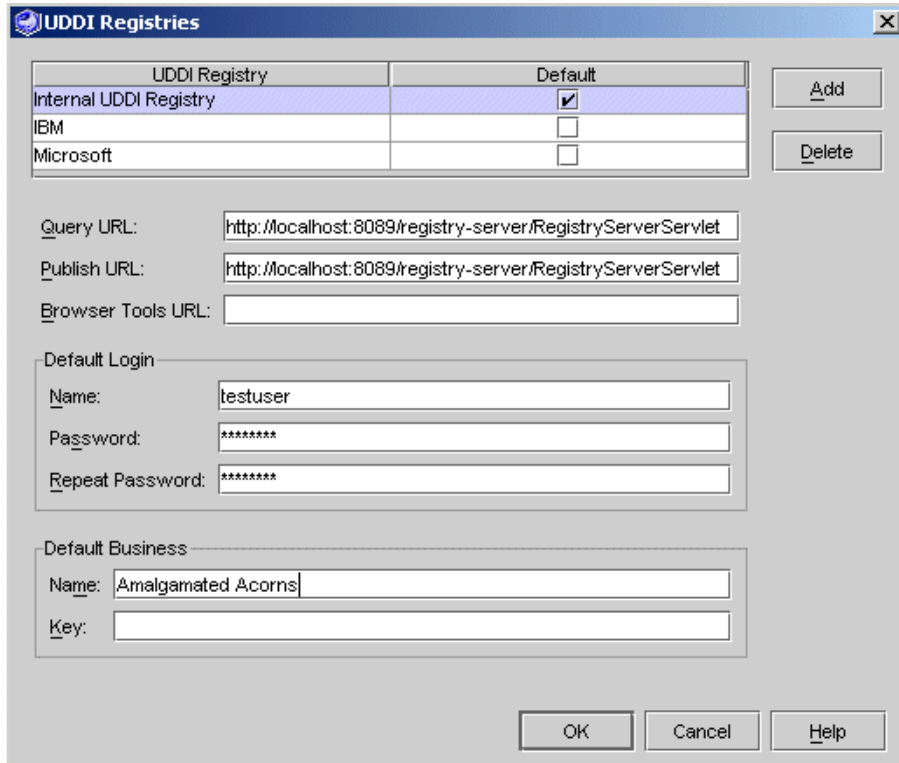


FIGURE 2-28 UDDI Registries Property Editor

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

Note – You can set a default login name and password, but doing so is a security risk in some environments. The password is stored in your IDE filesystem and is not encrypted. Typing the password each time is more secure.

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 114.

Gaining Access to an External UDDI Registry

To access a UDDI registry, you must provide the IDE with certain information (see “Editing Registries Information in the IDE” on page 73).

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*, 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:

- Adding a new business entry or finding an existing business entry, and associating the business entry with the web service entry. (Businesses added by this wizard do not have associated categories and identifiers.)
- Adding a new tModel entry or finding an existing tModel entry, and associating the tModel entry with the web service entry.
- (Optional) Associating one or more standard industry categories and identifiers with the web service entry or tModel entry.

The business is the organizational entity that owns the web service. Each business entry in the registry has a unique key, which is automatically generated when you add the entry to the registry.

The tModel is a registry entry that contains a URL (called the *overviewURL*) that points to your WSDL file. The WSDL file 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 113.

A business can publish many different services to a UDDI registry, where each service is described structurally by WSDL associated with a tModel entry. Many service entries can 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 with 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 choose Publish to UDDI.**

The Publish New Web Service to UDDI dialog box is displayed, as illustrated in FIGURE 2-29.

- a. Enter the Service Name in the UDDI registry.**
- b. Select http (the default) or https for the Network Access Point Type.**
- c. Edit the Network Access Point Address URL, which defaults to the value of the SOAP RPC URL property of your web service.**
- d. (Optional) Enter text for the Service Description.**

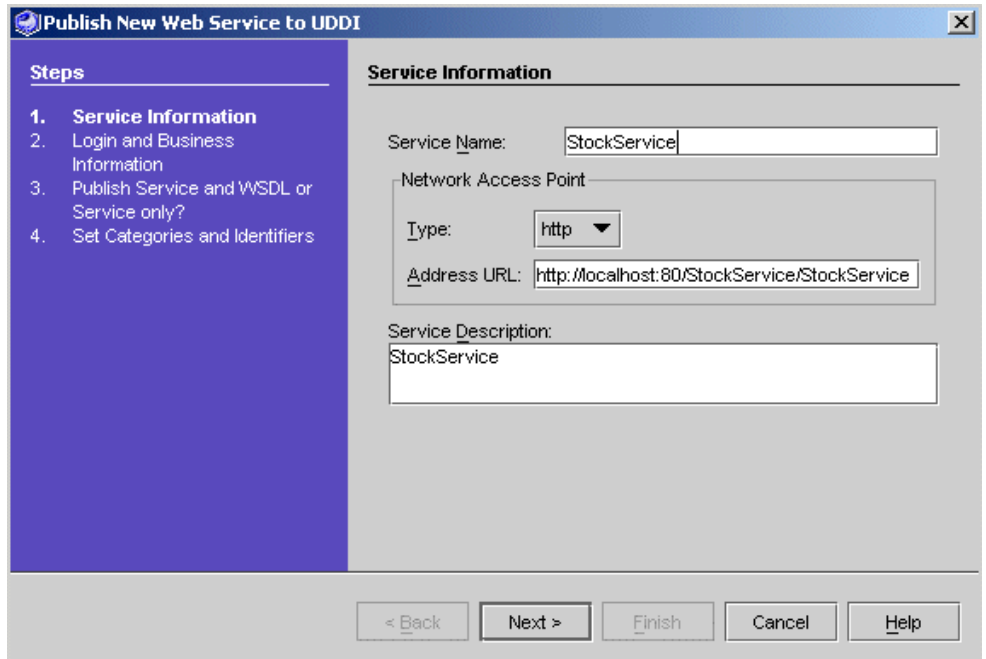


FIGURE 2-29 UDDI Publish New Web Service Dialog Box

Note – The host name in Address URL defaults to `localhost`. Change this to the network machine name of the host and make sure that the port number is correct.

2. **Click Next to display the Login and Business Information dialog box, as illustrated in FIGURE 2-30.**

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 entering a different value in each field.

- a. **Select the target UDDI registry for the publication.**
- b. **Enter a `userid` and password that entitles you to publish to the selected registry.**

c. Enter a value for Business Name.

The Find and Add buttons become active.

You can specify a business that already has an entry in the registry, or you can add a new business to the registry and associate that business with your web service.

To use an existing business, click Find. 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.

To use a new business, click Add. 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.

You can also change registry default values from the Login and Business Information dialog box.

Note – To change registry default values during the publication process, click Edit. The IDE displays the UDDI Registries Property Editor. For further information, see “Editing Registries Information in the IDE” on page 73.

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 2-30 UDDI Login and Business Information Dialog Box

d. Click Next.

The IDE displays the tModel selection dialog box, as illustrated in FIGURE 2-31.

3. Enter information about your web service in the tModel selection dialog box.

Specify what you want to publish and provide details.

You can associate a new or existing tModel with your web service. The dialog box enables you to:

- Create a new tModel in the registry
- Find an existing tModel

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 2-29), 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.

The WSDL File URL is used in creating a client from a UDDI registry entry. The default URL, illustrated in FIGURE 2-31, causes the running web service to deliver its WSDL file for download when you create a client from the UDDI registry. You might prefer to keep the WSDL file in another location. You might want to provide a static URL for the WSDL file—a URL that does not depend on the web service being available when a client is created.

Note – If you are publishing a web service whose Authentication property is HTTP Basic Authentication, you might have to change the WSDL File URL. For more information, see “Basic Authentication and UDDI Publication” on page 204

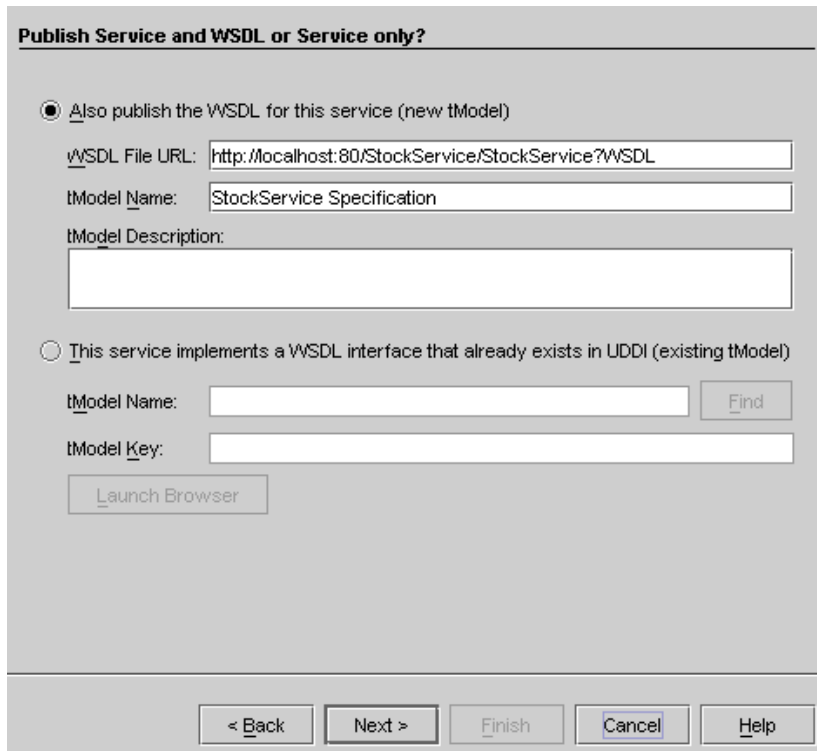


FIGURE 2-31 UDDI tModel Selection Dialog Box

To use an existing tModel in the UDDI, enter 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, enter it in 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 2-28).

4. Click Next.

The IDE displays the UDDI Set Categories and Identifiers dialog box, as illustrated in FIGURE 2-32. The dialog box shows your default categories and identifiers.

Click Edit to add or delete categories or identifiers. For a detailed description of the edit procedure, see “Setting Default Publish Categories and Identifiers” on page 70.

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 2-32 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.

Using 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. The internal registry is provided as a convenience for end-to-end testing of your development process. The UDDI registry consists of a servlet that runs in a dedicated web server (different from the server instance used for web application development) and uses an XML database for persistent storage. The IDE automatically starts and stops the dedicated web server and the database server when you start and stop the registry server.

Note – The internal UDDI registry is preconfigured with a single user. The name is `testuser` and the password is `testuser`.

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 see the Internal UDDI Registry node.
2. **Right-click the Internal UDDI Registry node and choose Start Server, as shown in FIGURE 2-33.**

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

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

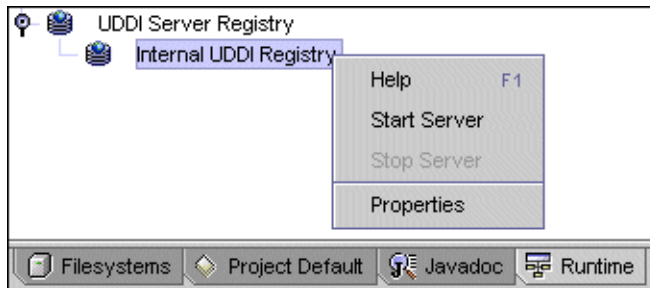


FIGURE 2-33 Start Internal UDDI Registry Server

To stop the internal UDDI registry server:

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

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 can be used to perform several UDDI maintenance tasks that are not provided by the IDE's UDDI wizards.

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 5, Standard Edition IDE. However, its main use is to access the internal registry server. An external registry typically has 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 82.

To start the sample registry browser:

1. **Open a command window and change your current directory to `s1studio-install-directory/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 Registry Browser window is displayed, as illustrated in FIGURE 2-34.

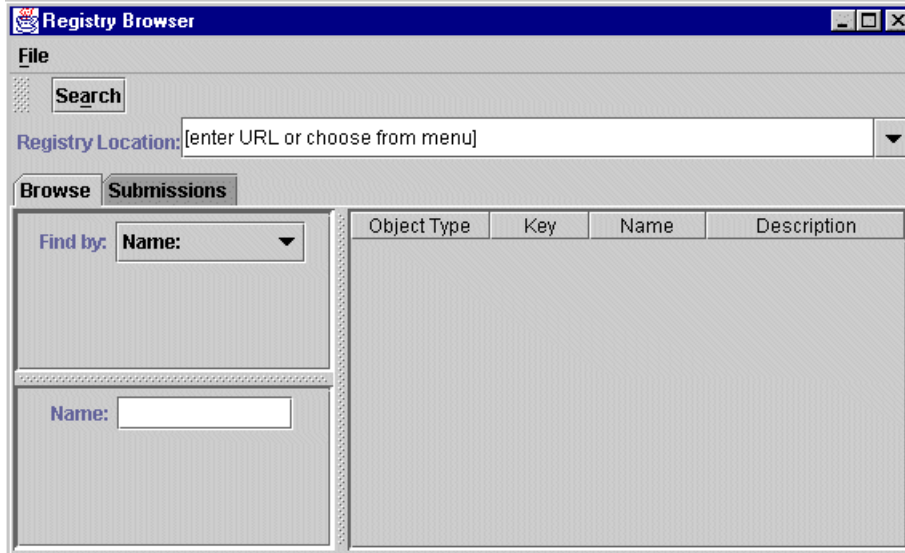


FIGURE 2-34 Java WSDP Sample Registry Browser

To configure the sample registry browser for your internal registry:

1. **Select the last URL in the Registry Location list, as illustrated in** FIGURE 2-35.



FIGURE 2-35 Java WSDP Sample Registry Browser URL Selection

2. **Change the port number in the URL from 8080 to the correct port number for the dedicated registry web server, as illustrated in** FIGURE 2-36.

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

To verify the port number:

- a. **Expand UDDI Server Registry Node in the Explorer's runtime tab.**

The Internal UDDI Registry Node is displayed.

b. Right-click the Internal UDDI Registry Node and choose Properties.

The Server URL property value contains the port number.

The port number is also displayed in the UDDI Registries Property Editor, in the Query URL and Publish URL, as illustrated in FIGURE 2-28.

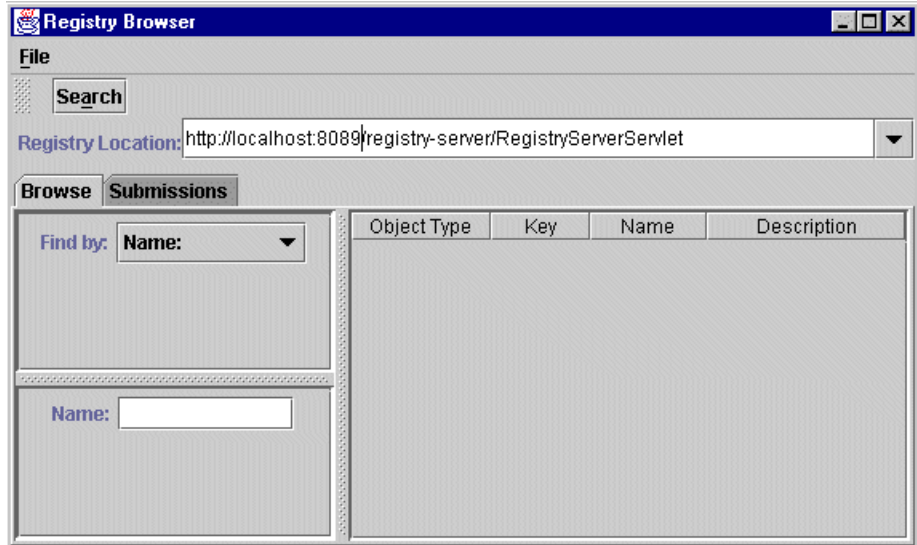


FIGURE 2-36 Java WSDP Sample Registry Browser With Internal Registry URL

An example showing the result of a search by company name is illustrated in FIGURE 2-37.

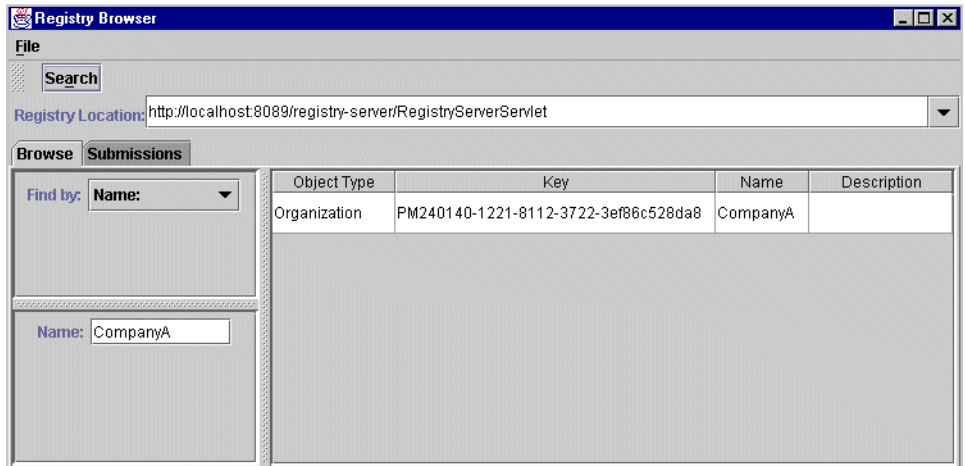


FIGURE 2-37 Java WSDP Sample Registry Browser Displaying Selected Business

An example using the Submissions tabbed pane is illustrated in FIGURE 2-38.

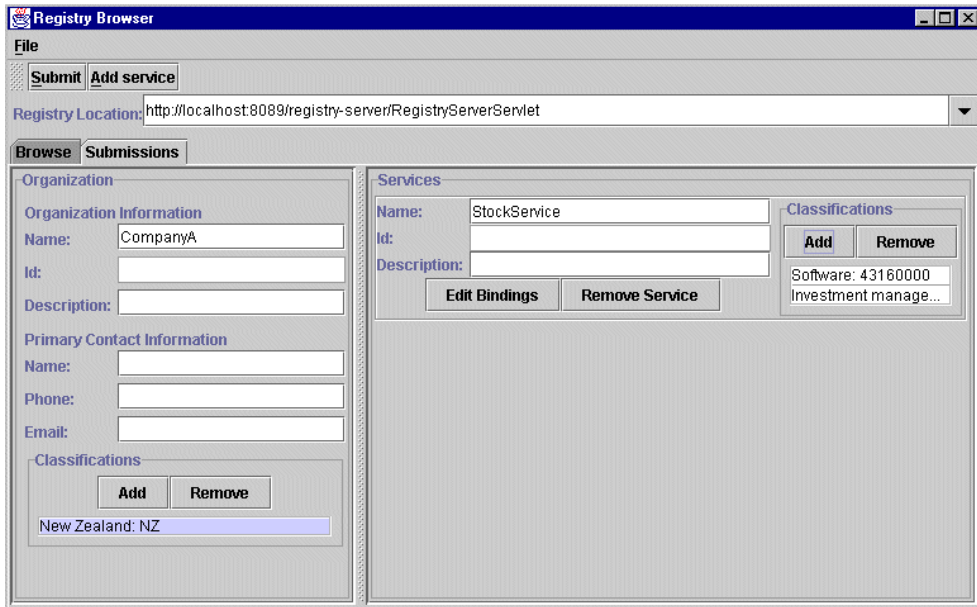


FIGURE 2-38 Java WSDP Sample Registry Browser Displaying Submissions Tabbed Pane

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>.

Using 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 5.

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 user-defined objects, you must provide the serialization classes and 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, and 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 2-39.

- 2. Click Add.**

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

Alternatively, you can click Add Single. The IDE opens the Add Single Class dialog box in which you can enter a serialization class (such as `Java.lang.String`) by name.

- 3. Select the desired classes, using Ctrl-click to select more than one, and click OK.**

The IDE displays the serialization classes.

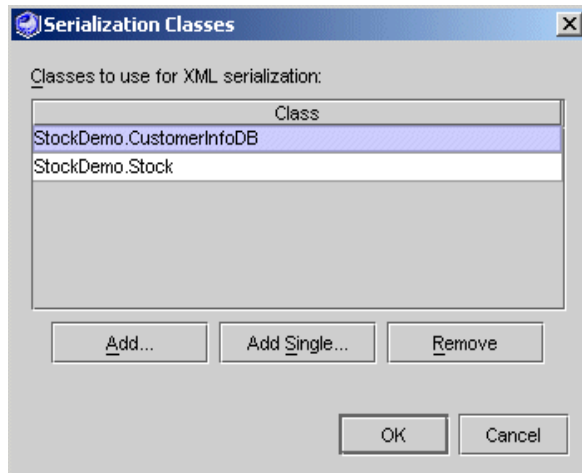


FIGURE 2-39 Serialization Classes Property Editor

Attachments

For information about using attachments with web services and clients, see “Attachments” on page 123.

SOAP Message Handlers

For information about using SOAP message handlers with web services and clients, see Chapter 4.

Security

For information about implementing security for web services and clients, see Appendix A.

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 a web service's deployment descriptors in the Source Editor at any time during development. You can also edit the deployment descriptors as a final step at the end of the development process. For information about how to view or edit deployment descriptors, see Appendix B.

Creating a Web Service Client

The Sun ONE Studio 5, Standard Edition 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:

- Creating a client from a local IDE web service
- Creating a client from WSDL
- Creating a client from a UDDI registry entry

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 to test your web service during development. (See “Setting a Default Test Client for a Web Service” on page 58.)

You can customize the generated 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 web service node, a package node, or the IDE’s main window menu.

To start at a web service node:

1. To open the New Web Service Client wizard from the Explorer, right-click the web service node (the one with the blue sphere icon ) and choose Create New Web Service Test Client.

A dialog box is displayed, as illustrated in FIGURE 3-1.

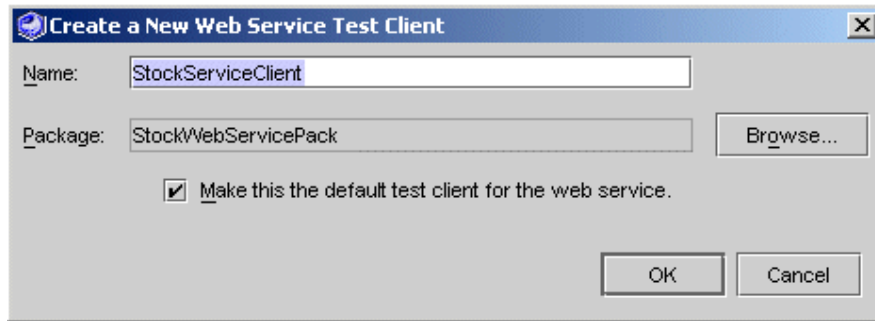



FIGURE 3-1 New Client From Local Web Service Dialog Box

2. Enter the desired location of your web service client in the Package field, or click Browse to find and select a package.
3. In the Name field, type a name for the new web service client and click OK.

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

If you enter the name of a web service client that already exists in the package designated in the wizard, the IDE displays an informative error message.

To start at a Java package node:

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

Alternatively, Choose File → New → 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.

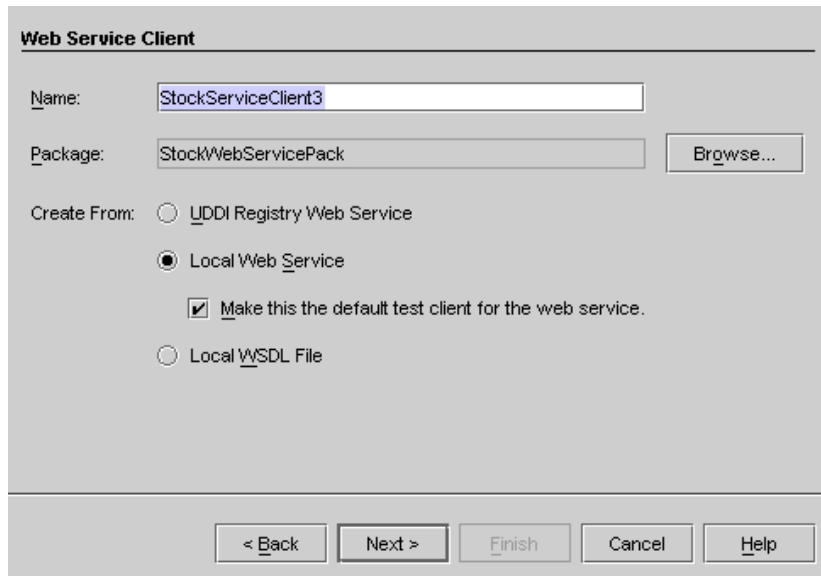


FIGURE 3-2 New Web Service Client Dialog Box

2. Set the desired name and package.
3. Select **Create From: Local Web Service**.
4. Click **Next** to open a chooser, select the desired web service, and click **Finish**.

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

Setting the Client Type: JAXRPC or kSoap

The IDE can generate a client that uses the JAX-RPC runtime (the default) or the kSOAP runtime. A JAX-RPC client is implemented as a web application with JSP pages as the presentation layer and a web browser as the front end. kSOAP is a SOAP API suitable for the Java 2 Microedition (J2ME) and wireless applications.

Note – If you have an old Apache SOAP client, the IDE automatically converts it to a JAX-RPC client during the generation.

To set the client type:

1. **Create the client.**
2. **Right-click the client node and choose Properties. The Soap Runtime property has the default value JAXRPC.**
3. **Click the property value.**

The choices JAXRPC and kSoap are displayed, as shown in FIGURE 3-3.

4. **Choose kSoap or leave the default value, JAXRPC.**

When you generate the client, a kSOAP or JAX-RPC client is created.

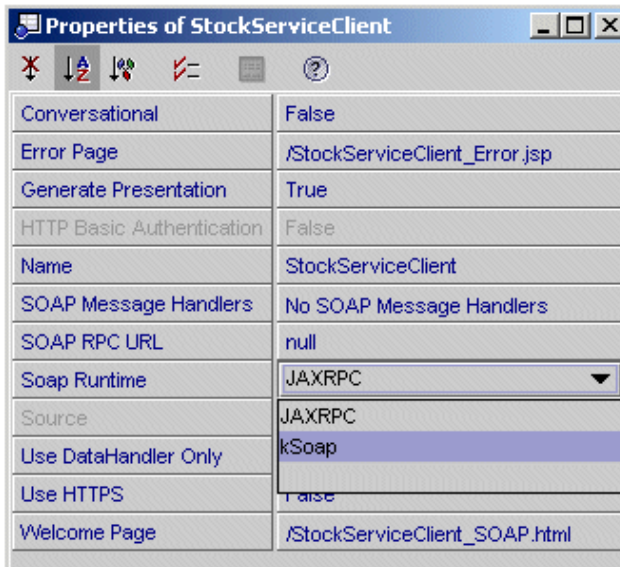


FIGURE 3-3 Client SOAP Runtime Property

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.

Note – In order to generate a kSOAP client in the IDE, you must have the `kjava` module installed. In order to test a kSOAP client in the IDE, you must have the `kjava-emulator` module installed. These two modules are not automatically installed with the IDE, but you can download them from the Update Center. To do this, choose Tools → Update Center and use the Update Center Wizard. For more information about the Update Center, see *Sun ONE Studio 5, Standard Edition Getting Started Guide*, or the IDE’s online help, or the Release Notes.

For information about wireless applications, see the following resources:

- J2ME site: <http://wireless.java.sun.com>
- Wireless developer site: <http://java.sun.com/j2me>
- *Mobile Edition* documents in “Documentation Available Online” on page 20.

The following sections describe JAX-RPC clients and kSOAP clients.

Generating a JAX-RPC Client

To generate a JAX-RPC client:

1. **Set the Soap Runtime property value to `JAXRPC`.**
2. **Right-click the client node in the Explorer, and choose `Generate Client Files`.**

The IDE generates:

- A set of classes called the *client proxy* or *client stub* that exchange SOAP messages with the web service at runtime
- A JSP tag library with custom tags, one tag for invoking each web service operation
- An HTML page and a set of JSP pages (one for each web service operation) that use the custom tags. The HTML page, also called the *welcome page*, is the application’s starting point for an end-user.

Each of these elements is described in the following sections.

The Generated Files

The IDE displays the generated client files under two nodes in the same package as the client:

- A node named `xxxGenClient` (where `xxx` is the client name) has the client proxy classes.
- A node named `Generated Documents` (under the client node) has the generated HTML welcome page, an HTML error page, and JSP pages.

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

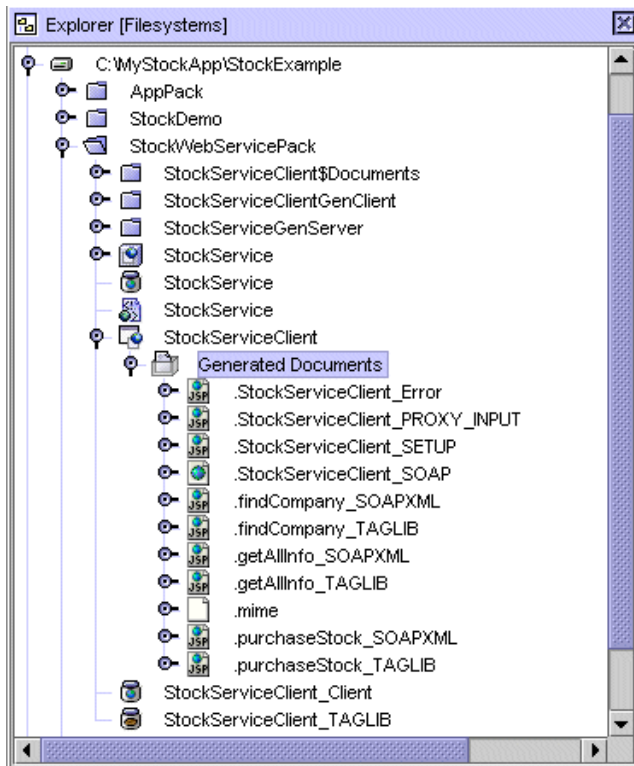


FIGURE 3-4 Client Documents Node and GenClient Node in Explorer

The JSP Custom Tags

This book assumes general knowledge of JSP technology. There are many books available on this subject. You can find the JSP specification at:

<http://java.sun.com/products/jsp/download.html#specs>

The IDE generates a JSP tag library with custom tags that make the client's JSP pages easier to read. Web designers can customize the client's presentation without expert knowledge of the Java programming language, using web design tools such as Macromedia's Dreamweaver.

Here is an example of a custom tag:

```
<ws:sayHello string_1=":?" var="resp">
</ws:sayHello>
```


This tag could be used to invoke a remote web service operation exposing the following method:

```
String resp = sayHello(String string_1);
```

Each web service's operation name and operation parameters are mapped to the corresponding tag name and tag attributes.

The tag attribute value ":" instructs the tag handler to automatically bind the attribute value from HTML form request variables.

The result of the web service operation call is returned to the JSP page as a JSP scripting variable named in the attribute `var`. The JSP page renders the content of the returned scripting variable, using JSTL (Java Standard Tag Library) tags to produce the default result presentation.

The Client HTML and JSP Pages

The generated client documents include:

- **One JSP page for each web service operation.** Each JSP page calls the client proxy to send a SOAP message to the web service to execute a business method (or XML operation) and return a result object. Web page designers can customize the generated JSP pages and use them as the basis for more sophisticated, user-friendly clients.
- **An HTML welcome page.** The welcome page displays an Invoke button, a Reset button and input fields (where input parameters are required) for each generated JSP page, as illustrated in FIGURE 3-12.
- **A JSP error page.**

The names of the welcome page and JSP error page appear in the IDE as properties of the client node, as illustrated in FIGURE 3-3.

To view the code of an HTML page or JSP page:

- **Right-click the node of the desired document and choose Open.**

Sample code for a generated welcome page is illustrated in FIGURE 3-5.

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>
StockService
</title>
</head>
<center>
<h2>StockService</h2>
<h4>http://localhost:80/StockService/StockService</h4>
<hr> </hr>
<h4><a href="StockServiceClient_PROXY_INPUT.jsp">Input proxy server</a>
<hr> </hr>
<h3>CompanyInfo findCompany(String string_1)</h3>
<form method="post" action="findCompany_TAGLIB.jsp">
<table width="75%">
<tr>
<td>
<table width="100%" border="1">
<tr>
<th>string_1:</th>
<td>
<input type="text" name="string_1"/>
</td>
</tr>
</table>
</td>
</tr>
</table>
<br>
<input type="submit" value="Invoke"/>
<input type="reset" value="Reset"/>
</form>

```

FIGURE 3-5 Client HTML Welcome Page

Sample code for a generated JSP page is illustrated in FIGURE 3-6.

```

<%@ taglib uri="/WEB-INF/StockServiceServantInterfacePort.tld" prefix="ws"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<%@ page contentType="text/html; charset=UTF-8" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>
StockService
</title>
</head>
<center>
<h2>StockService</h2>
<h4>http://localhost:80/StockService/StockService</h4>
<hr> </hr>
<h3>CompanyInfo findCompany(String string_1)</h3>
<a href="findCompany_SOAPXML.jsp"><h4>View SOAP Request/Response</h4></a>

<ws:findCompany string_1=":"?" var="resp">
</ws:findCompany>

<table width="75%" border="1">
<tr>
<th>Return Value</th>
<td>
<table width="100%" border="1">
<tr>
<th>totalRevenue</th>
<td><pre><c:out value="{resp.totalRevenue}"/></pre></td>
</tr>
<tr>
<th>valuationData</th>
<td>

```

FIGURE 3-6 Client Sample JSP Page

The Client Proxy Classes

FIGURE 3-7 shows an Explorer view of some of the classes generated for a JAX-RPC client.

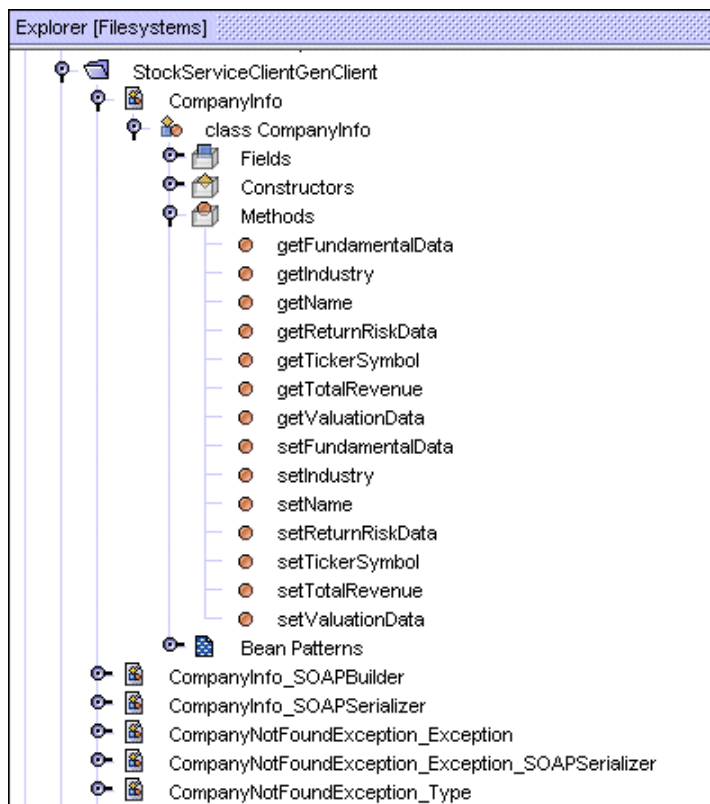


FIGURE 3-7 Client SOAP Proxy GenClient Node

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.

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

Using Your Own Front-End Client Components

You might already have a front end for your business client, such as a Swing component or a more sophisticated set of HTML pages and JSP pages than those generated by the IDE. Under those circumstances you might want the IDE to generate *only* the client proxy classes.

To generate only the client proxy classes:

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 proxy classes are created. Typically you might do this if you are using your own front-end presentation. In this case, you might need to add your own documents, libraries, or classes to the client's WAR file.

To add documents, libraries, or classes to your client's WAR file:

1. If the files were created outside the IDE, such as HTML and JSP pages created in a web design tool, mount the directory containing the files so they are visible in the IDE's Explorer.
2. Right-click the client node and choose Add to WAR File > *objectType*, where *object-type* is Documents, Libraries, or Classes.

The IDE displays a chooser.

3. Navigate to a folder that contains your documents, libraries, or classes, select the desired objects, and click OK.

The IDE creates a Documents, Libraries, or Classes folder under the client node, with references to the added objects, as illustrated in FIGURE 3-8.

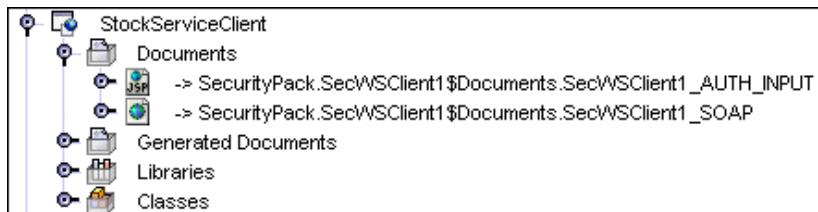


FIGURE 3-8 Client References to Documents, Libraries, and Classes

The IDE adds the documents, libraries, or classes to your client's WAR file and you can use them to build a custom client.



Caution – Deleting a reference to a file or folder deletes only the reference, leaving the referenced file or folder intact. However, nodes *inside* a referenced folder are links that represent *actual* files or folders. 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.

Validation During Client Generation

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 JavaBeans component). Validation is especially 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.

Refreshing a Client From Its Web Service

The *Refresh From Web Service* menuitem is an alternative to *Generate Client Files*. When you choose *Generate Client Files*, the IDE generates client proxy classes and documents based on a combination of the web service properties and the client properties.

For example, if a local web service is stateful, the IDE creates a stateful client by default. The web service and client both have their *Conversational* property set to `True`. If you manually change the client's *Conversational* property to `False` and then use the *Generate Client Files*, the IDE keeps your change and generates a stateless client.

An alternative to *Generate Client Files* is *Refresh from Web Service*. When you choose this menuitem, the IDE generates the client proxy classes and documents based on the properties of the web service. In this case, if your local web service has the *Conversational* property `True`, the IDE makes the *Conversational* property of the client `True` to match the web service.

If a web service is enhanced after a client was created, methods might be added and method signatures might change. The *Refresh* uses the new WSDL and generates client files appropriate to the enhanced web service.

If the client is generated from a WSDL file or from the WSDL reference in a UDDI registry, only the web service properties that are carried in the WSDL are used by *Refresh From Web Service*. In particular, security and conversational properties are not specified in WSDL, so the IDE does not change the client values.

To regenerate a client based on the web service:

- **Right-click the client in the Explorer and choose *Refresh From Web Service*.**

This causes the client proxy and documents to be regenerated.

Note – If the client was generated from a local web service, the menuitem is *Refresh From Web Service*. If the client was generated from a WSDL file, the menuitem is *Refresh From WSDL*. If the client was generated from a UDDI registry, the menuitem is *Refresh From UDDI*. In all cases, the IDE generates the client on the basis of the WSDL file at the original location. The IDE keeps the URL of the WSDL file as the value of the client's Source property.

The Service Endpoint URL

For your client to access a service instance at runtime, the client proxy must use 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 instead 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.

You might want the client to override the WSDL value of the endpoint URL if the web service is redeployed to a production system, or for other reasons.

To make your client pass the endpoint URL at runtime:

1. **Right-click the client in the Explorer and choose Properties.**
2. **Set the SOAP RPC URL property to a URL of your choice.**

This URL overrides any default URL that might have been derived from the WSDL file.

Deploying a JAX-RPC Client

The IDE deploys the client as a web application in your application server's web container.

To deploy the client:

- **Right-click its node in the Explorer and choose Deploy.**

The client's deployed WAR file is visible in the Explorer's runtime tabbed pane, as illustrated in FIGURE 3-9.

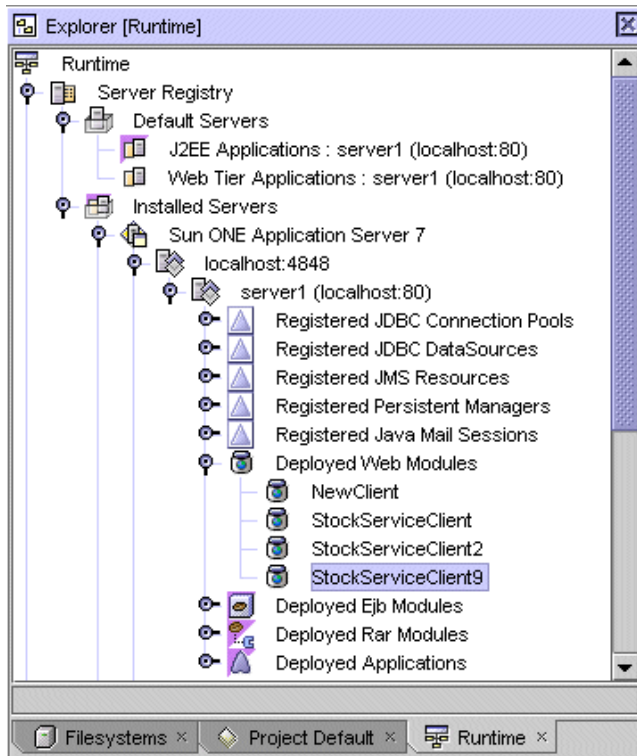


FIGURE 3-9 Client WAR File in Explorer (Deployed Web Modules Node)

The client's WAR file node is visible in the Explorer's Filesystems tabbed pane, as illustrated in FIGURE 3-10.

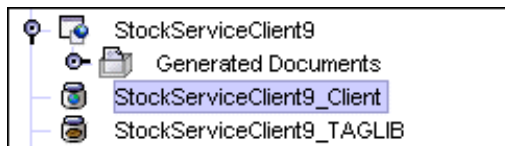


FIGURE 3-10 Client WAR File in Explorer (Client Node)

To view the contents of the WAR file:

- **Right-click the client's WAR node in the Explorer's Filesystems tabbed pane, and choose View WAR Content.**

The IDE displays the WAR file contents, as illustrated in FIGURE 3-11.

Path Name	File Name	Ext
	purchaseStock_SOAPXML	jsp
	findCompany_SOAPXML	jsp
	getAllInfo_SOAPXML	jsp
	getAllInfo_TAGLIB	jsp
	StockServiceClient9_SETUP	jsp
	purchaseStock_TAGLIB	jsp
	StockServiceClient9_SOAP	html
	StockServiceClient9_PROXY_INPUT	jsp
	findCompany_TAGLIB	jsp
	mime	types
	StockServiceClient9_Error	jsp
META-INF	MANIFEST	MF
WEB-INF	StockServiceServantInterfacePort	tid
WEB-INF	web	xml
WEB-INF	sun-web	xml
WEB-INF/lib	saaj-ri	jar
WEB-INF/lib	jaxrpc-ri	jar
WEB-INF/lib	saaj-api	jar
WEB-INF/lib	StockServiceClient9_TAGLIB	jar
WEB-INF/lib	commons-fileupload-1.0-dev	jar
WEB-INF/lib	jaxrpc-api	jar
WEB-INF/lib	schema2beans	jar
WEB-INF/lib	jstl	jar
WEB-INF/lib	commons-beanutils-1.3	jar
WEB-INF/lib	standard	jar

Close

FIGURE 3-11 Client WAR File Contents

Executing a JAX-RPC Client

This section assumes that the web service referenced by your client has been deployed, that your client has been deployed, and that the servers hosting the service and client are running.

To execute the client:

- **Right-click the client node in the Explorer and choose Execute.**

The IDE executes the client and displays the welcome page in your default web browser, as illustrated in FIGURE 3-12.

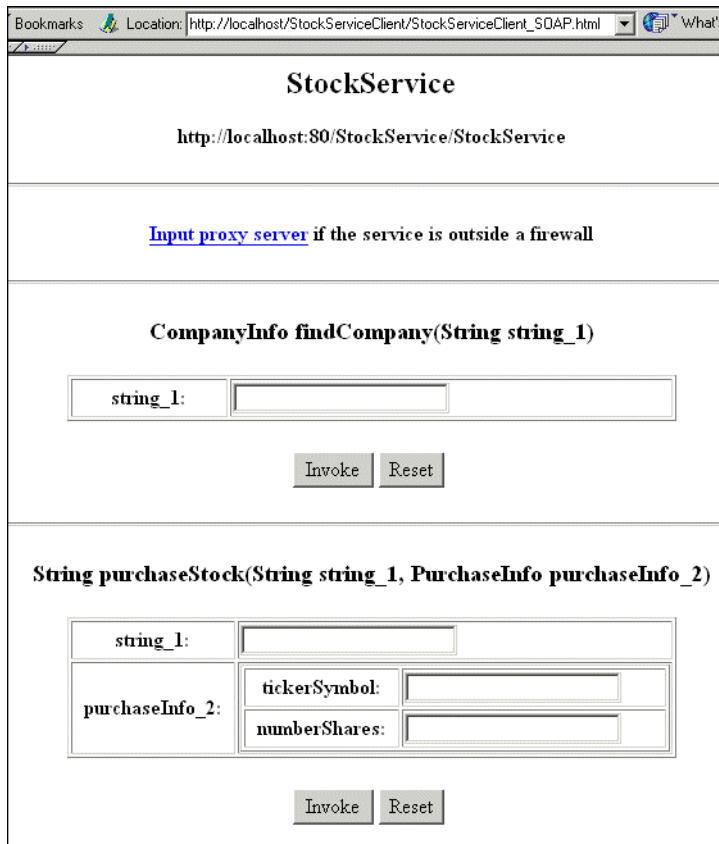


FIGURE 3-12 Client Welcome Page

Suppose that the web service is deployed in a J2EE application, and that you make the client the default test client, as explained in “Setting a Default Test Client for a Web Service” on page 58. In this case the IDE deploys the client with the web service, and you can execute the client from the J2EE application node.

To execute the default test client from the J2EE application node:

- **Right-click the J2EE application node in the Explorer and choose Execute.**

The client generated by the IDE gives you two ways of viewing the result of each web service method call. You can view the results in a user-friendly format, or you can view the SOAP request and response messages in XML format.

Consider the example in FIGURE 3-12 and the `findCompany` method. Suppose you enter a valid company ticker symbol for `string_1` and click Invoke. The client displays company information in the browser, as illustrated in FIGURE 3-13. This is a user-readable display that can be a starting point for a web designer to customize the presentation.

CompanyInfo findCompany(String string_1)			
View SOAP Request/Response			
Return Value	totalRevenue	18300000	
	valuationData	PEGRatio	1.8
		trailingPE	33.4
		priceToSales	1.7
	industry	Comp. Hard.	
	tickerSymbol	MCE	
	returnRiskData	dividendYield	1.0
		assetClass	Large Cap
		fiveYearPrjEarning	18.0
	fundamentalData	returnOnEquity	11.0
salesPerEmployee		12.0	
profitMargin		5.4	
returnOnAssets		5.4	
debtToEquity		0.2	
name	Moon Company		

FIGURE 3-13 Client Display of Company Information

If you click the View SOAP Request/Response link, the client displays the SOAP request message (which is sent from the client to the web service) and the SOAP response message (which is sent from the web service to the client). This display, illustrated in FIGURE 3-14, includes SOAP envelopes and data, and might be useful for debugging a client or web service.

```

CompanyInfo findCompany(String string_1)

SOAP Request Response
*****
Request
Content-Type: text/xml; charset="utf-8"
Content-Length: 563
SOAPAction: ""
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns0="urn:StockService/wsd1" xmlns:ns2="http://java.sun.com/jax-rpc/ti/internal"
  xmlns:ns3="urn:StockService/types"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap:Body>
    <ns0:findCompany>
      <string_1
        xsi:type="xsd:string">BCE</string_1>
      </ns0:findCompany>
    </soap:Body>
  </soap:Envelope>

Response
Content-Type: text/xml; charset="utf-8"
Content-Length: 563
SOAPAction: ""
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns2="urn:StockService/types" xmlns:ns3="http://java.sun.com/jax-rpc/ti/internal"
  xmlns:ns4="urn:StockService/wsd1" xmlns:ns5="http://schemas.xmlsoap.org/soap/encoding/">
  <soap:Body>
    <ns1:findCompanyResponse
      xmlns:ns1="urn:StockService/wsd1">
      <result href="#I11"/>
    </ns1:findCompanyResponse>
    <ns0:CompanyInfo id="I11"
      xsi:type="ns0:CompanyInfo">
      <totalRevenue
        xsi:type="xsd:long">1800000</totalRevenue>
      <valuationData
        href="#I12"/>
      <industry
        xsi:type="xsd:string">Comp. Hard.</industry>
      <tickerSymbol
        xsi:type="xsd:string">BCE</tickerSymbol>
      <returnRiskData href="#I13"/>
      <fundamentalData href="#I14"/>
    </ns0:CompanyInfo>
    <ns0:ValuationInfo id="I12"
      xsi:type="ns0:ValuationInfo">
      <PERatio
        xsi:type="xsd:double">1.8</PERatio>
      <trailingPE
        xsi:type="xsd:double">33.4</trailingPE>
      <priceToSales
        xsi:type="xsd:double">1.7</priceToSales>
    </ns0:ValuationInfo>
    <ns0:ReturnRiskInfo id="I13"
      xsi:type="ns0:ReturnRiskInfo">
      <dividendField
        xsi:type="xsd:double">1.0</dividendField>
      <assetClass
        xsi:type="xsd:string">Large Cap</assetClass>
      <fiveYearPEarning
        xsi:type="xsd:double">18.0</fiveYearPEarning>
    </ns0:ReturnRiskInfo>
    <ns0:FundamentalInfo id="I14"
      xsi:type="ns0:FundamentalInfo">
      <returnOnEquity
        xsi:type="xsd:double">11.0</returnOnEquity>
      <salesPerEmployee
        xsi:type="xsd:double">11.0</salesPerEmployee>
      <profitMargin
        xsi:type="xsd:double">5.4</profitMargin>
      <returnOnAssets
        xsi:type="xsd:double">5.4</returnOnAssets>
      <debtToEquity
  
```

FIGURE 3-14 Client Display of Company Information (SOAP Request/Response)

Note – The IDE generates JSP pages with names that match the corresponding operations of the web service. If you are testing a client and you want to be sure which JSP page is implementing a given part of the display, see the URL in your browser.

Firewalls

If the client is executing inside a firewall and the web service is outside the firewall, you must specify the proxy host and proxy port.

To specify the proxy host and proxy port at runtime:

- **Click the Input proxy server link on the welcome page.**

The client displays a page, illustrated in FIGURE 3-15, in which you can enter values for the proxy host and proxy port.

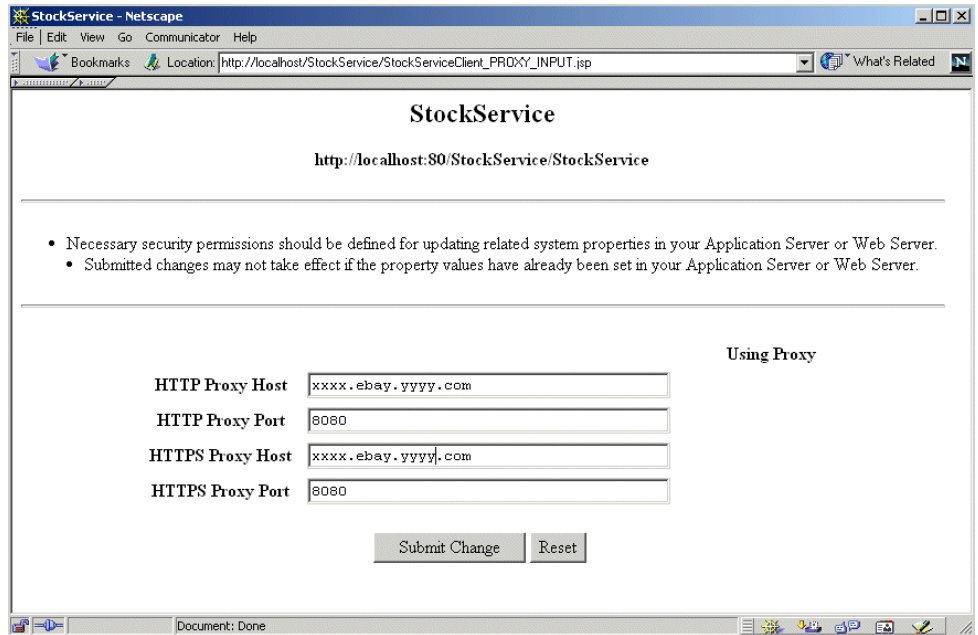


FIGURE 3-15 Client Input Proxy Server Page

To specify default values for the proxy host and proxy port in the IDE:

- **Choose Tools → Options → IDE Configuration → System → System Settings → System from the IDE's main window, and set the values of Proxy Host and Proxy Port.**

These values become the default values for clients generated by the IDE.

Note – If a client's user enters runtime values for proxy host and proxy port, the runtime values override any default values set in the IDE.

Distributed Applications

A variety of problems can arise when the client and web service are running in different servers. In a development environment the servers are likely to be the same and under your control. In a production environment the servers are likely to be

different and under the control of system administrators. You might encounter runtime messages, such as a `java.security.AccessControlException` message, that require permissions to be set on the server.

For further information about problems of this kind, see *Sun ONE Studio 5, Standard Edition Release Notes*.

Generating a kSOAP Client

In order to generate a kSOAP client in the IDE, you must have the `kjava` module installed. For information about how to install this module, see “Setting the Client Type: JAXRPC or kSoap” on page 93

To generate a kSOAP client:

1. **Set the client’s Soap Runtime property value to `kSoap`, as described in “Setting the Client Type: JAXRPC or kSoap” on page 93.**
2. **Right-click the client node in the Explorer, and choose Generate Client Files.**

The IDE generates:

- A client proxy class named `xxxkSOAPProxy`, where `xxx` is the client name. The client proxy class exchanges SOAP messages with the web service at runtime
- A MIDlet named `xxxMIDlet`, where `xxx` is the client name. The MIDlet provides a graphical user interface for requests and responses

The generated client nodes are illustrated in FIGURE 3-16.



FIGURE 3-16 kSOAP Client Nodes in the Explorer

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.

Note – For a kSOAP client, the IDE does not create a Generated Documents node and does not generate HTML or JSP pages.

Executing a kSOAP Client

In order to execute a kSOAP client in the IDE, you must have the `kjava-emulator` module installed. For information about how to install this module, see “Setting the Client Type: JAXRPC or kSoap” on page 93.

To execute a kSOAP client in the IDE:

- **Right-click the client node in the Explorer, and choose Execute.**

The IDE opens the emulator and displays a client menu, as illustrated in FIGURE 3-17.

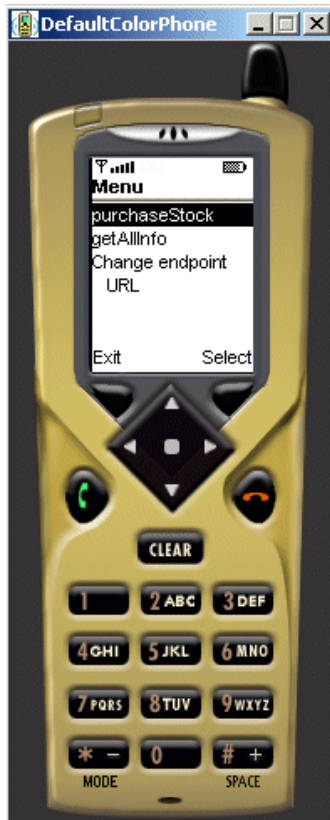



FIGURE 3-17 kSOAP Client Executing In Cell Phone Emulator

Creating a Client From a WSDL File

WSDL provides a portable representation of the external interfaces of a web service. See “Generating WSDL” on page 69 for the procedure to create a WSDL description of your web service. See “WSDL” on page 29 and “Working With UDDI Registries” on page 68 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 a WSDL file:

1. **From the IDE's main window, choose File → New → Web Services → Web Service Client, and click Next.**

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

2. **Set the desired name and package, and select Create From: Local WSDL File.**
3. **Click Next to open a chooser window.**
4. **Select the desired WSDL node in the chooser window, and click Finish.**

The IDE creates a client node with default Soap Runtime property `JAXRPC`. You can change this property to `kSoap` if you want a wireless client.

5. **Right-click the client node in the Explorer and choose Generate Client Files.**

The IDE creates the client files.

Note – As an alternative to Generate Client Files, you can choose Refresh From WSDL. See “Refreshing a Client From Its Web Service” on page 102 for an explanation of this context-sensitive menuitem.

6. **The remaining steps, including deployment and execution, are the same as in “Creating a Client From a Local Web Service” on page 91.**

To generate only the client's supporting classes, without the Generated Documents (for a JAX-RPC client) or the MIDlet (for a kSOAP client):

1. **Right-click the client node in the Explorer, choose Properties, set the value of the Generate Presentation property to `False`.**

2. **Right-click the client node in the Explorer and choose Generate Client Files.**

The IDE creates only the client proxy classes.

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

This might be the best approach if you already have fully developed clients that you can easily customize to use the client 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. This section gives an overview of the planning and work flow, followed by the procedures for searching a UDDI registry, selecting a web service, and creating a client.

Creating a Client—Planning and Work Flow

The work flow consists of these tasks:

- Determining your criteria for selecting a web service
- Selecting a UDDI registry to search
- Searching the UDDI registry for a web service
- Downloading the WSDL for the web service
- Generating the client
- Customizing the client

Determining your criteria for finding a web service in a UDDI registry requires advance planning and is part of the application design process. It is not enough to consider only 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 is likely to 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 → All Templates → Web Services → Web Service Client.**

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

The IDE displays the Web Service Client dialog box, as illustrated in FIGURE 3-18.

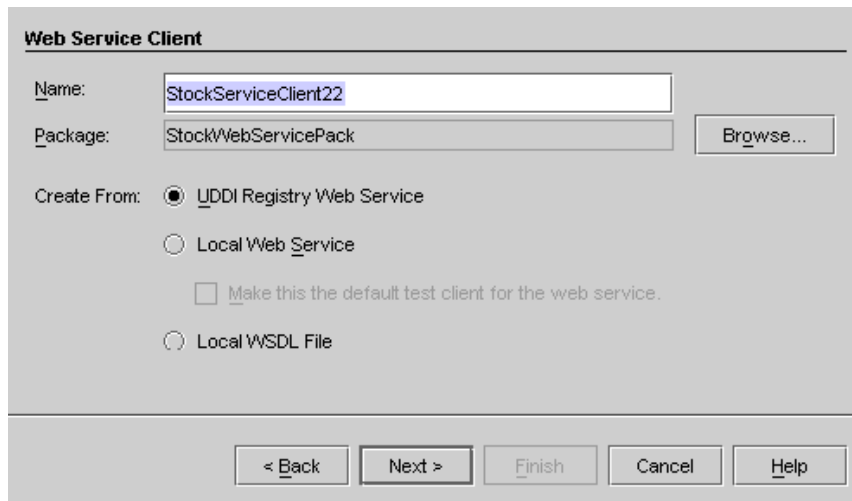


FIGURE 3-18 New Web Service Client Wizard

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 3-19.**

The dialog box displays a list of UDDI registries.

At this point you can select the desired registry.

You can also click Edit to change the default registry or edit registry information (see “Editing Registries Information in the IDE” on page 73).

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 “Using the Internal UDDI Registry” on page 81.

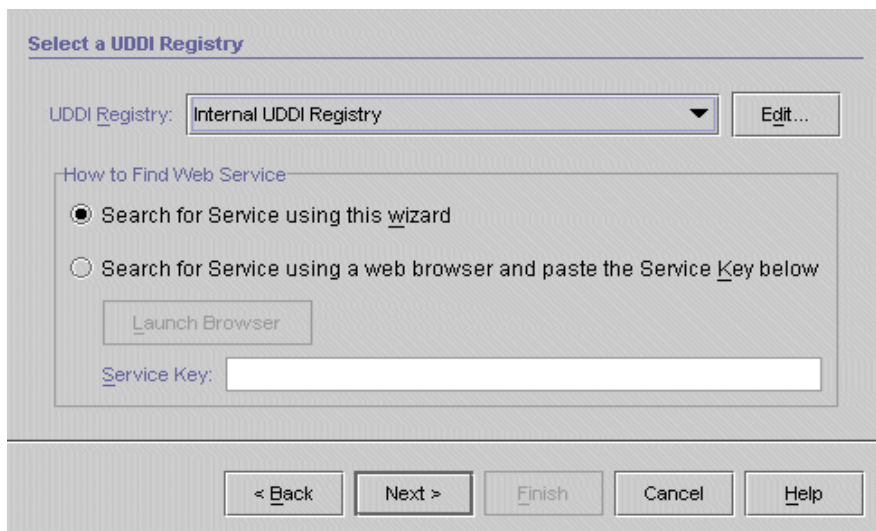


FIGURE 3-19 UDDI Registry Selection Dialog Box

The IDE provides two ways to search for a web service. You choose the search method by clicking one of two radio buttons. The default choice, illustrated in FIGURE 3-19, uses the IDE wizard to conduct the search. The other choice uses a web browser and browser-based software provided by the registry owners.

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 73).

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.

Note – 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 Search UDDI Registry dialog box, as illustrated in FIGURE 3-20.**

Search UDDI Registry and Select a Service

Search For: Business Name Containing: []

Search

Search Results

Matching Businesses (0):

Name	Description
------	-------------

Filter Businesses

Services with WSDL for Selected Business (0):

Name	Description
------	-------------

Enter query text and press Search

< Back Next > Finish Cancel

FIGURE 3-20 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 3-21.

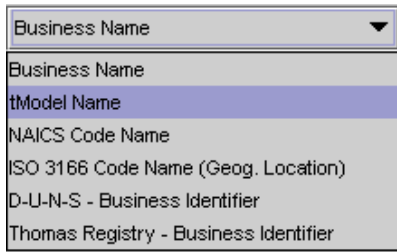


FIGURE 3-21 UDDI Registry Search Types

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 3-20) 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 enter a search string of `United Kingdom` or `GB`.
- Select NAICS and enter 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. Enter 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 3-22. 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 know what businesses they are interested in. The main problem when seeking a business in the registry might be spelling the name correctly 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.

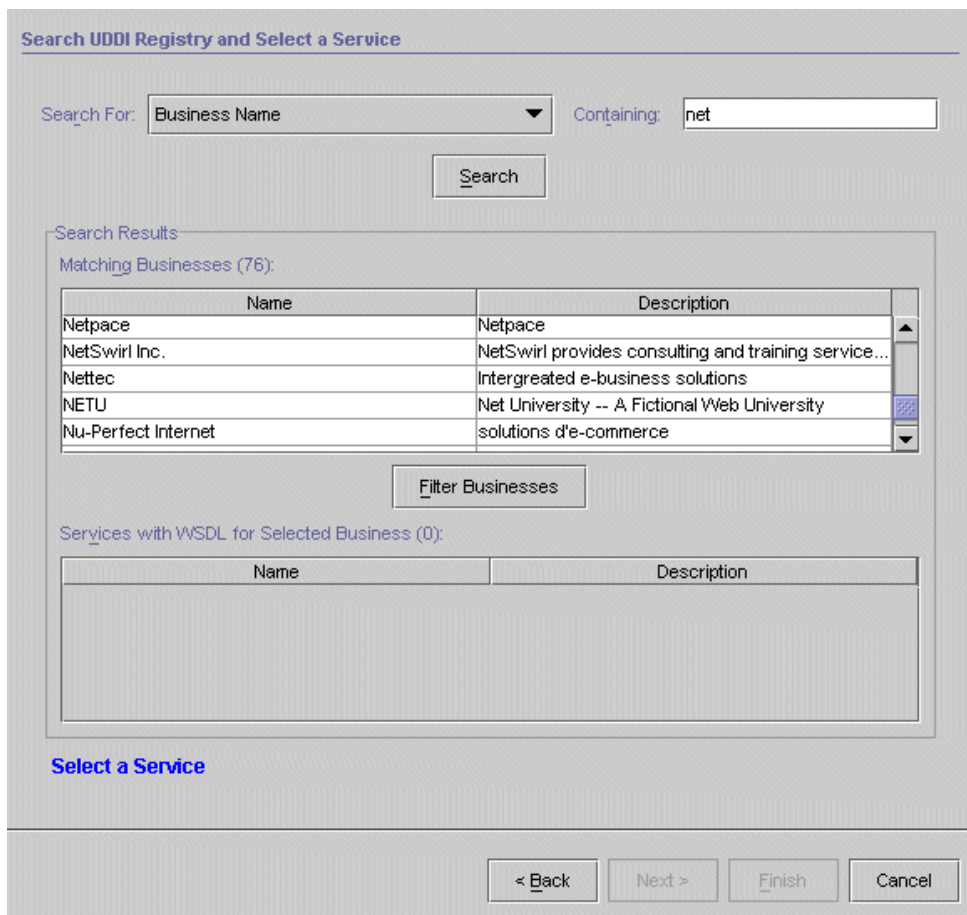


FIGURE 3-22 UDDI Registry Search Dialog Box With Matching Businesses

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 3-23. 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.

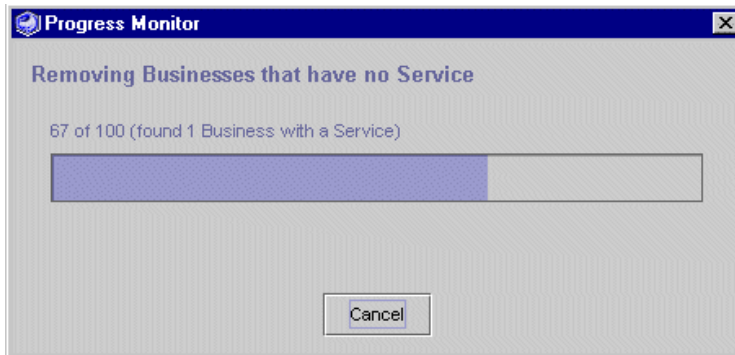


FIGURE 3-23 UDDI Registry Filter Business Progress Monitor

7. Select a business from the Search Results list.

The IDE displays services associated with the selected business, as illustrated in FIGURE 3-24, 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.

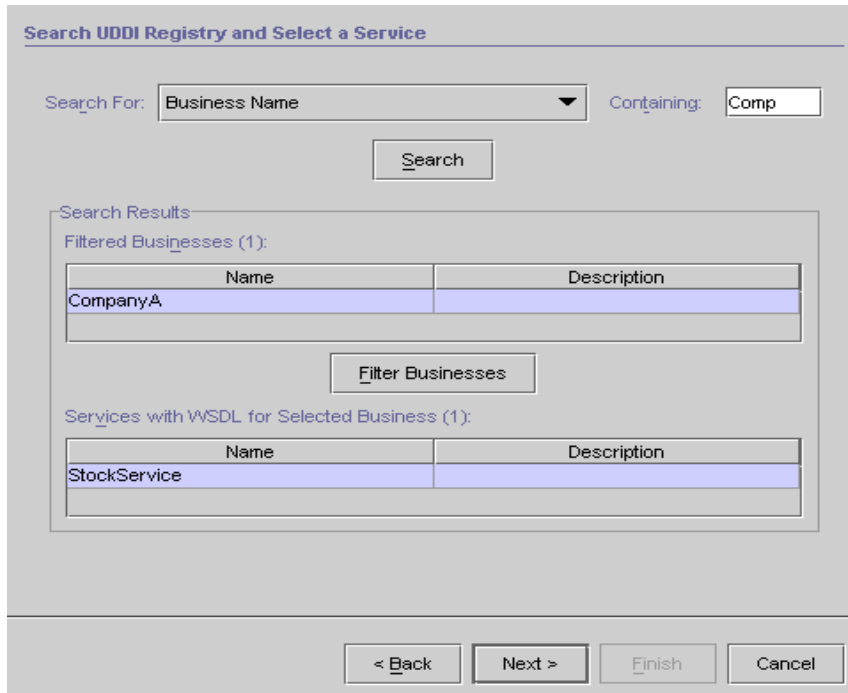


FIGURE 3-24 UDDI Registry Select Service

8. Select a service and click Next to display the Service Interface dialog box, as illustrated in FIGURE 3-25.

The dialog box shows detailed 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 a 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.

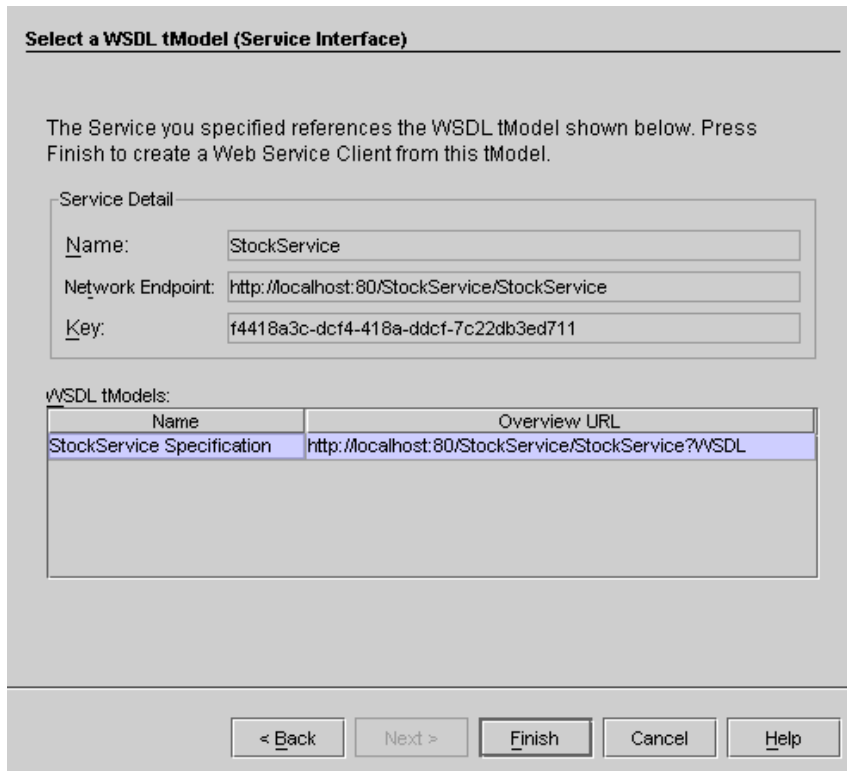


FIGURE 3-25 UDDI Registry Display Service Details and tModel

9. Select a WSDL tModel and click Finish.

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

10. Right-click the client node in the Explorer and choose Generate Client Files.

The IDE creates the client files.

Note – As an alternative to Generate Client Files, you can choose Refresh From UDDI. For an explanation of this context-sensitive “Refreshing a Client From Its Web Service” on page 102.

11. The remaining steps, including deployment and execution, are the same as in “Creating a Client From a Local Web Service” on page 91.

Attachments

A web service operation's input parameters and return values are transmitted in one of two ways: encoded as XML tags or attributes within the message's SOAP body, or appended to the message as an attachment. Attachments are generally used for objects that are not easily or efficiently serialized into an XML representation, such as images. An attachment is packaged as a well-defined MIME-encoded region of the message. The IDE supports the World Wide Web Consortium (W3C) standard for SOAP attachments.

The most common use of JAX-RPC attachments is for the return of images, such as gif and jpeg data. The IDE can generate "image-aware" JAX-RPC test clients for *web-centric* web services that return images (objects of the `Java.awt.Image` class).

Attachments Example

The following example is based on a Java class named `attTest` that has several methods for processing images and text. A web-centric web service named `attWS` exposes the business methods of the Java class. A JAX-RPC test client named `attWSClient` communicates with the web service.

The example assumes that the web service has been deployed and that the server is running.

CODE EXAMPLE 3-1 shows the source code of the Java class, `attTest`. The business methods are:

- `echoImage`
- `echoMessage`
- `echoXML`
- `getImage`.

CODE EXAMPLE 3-1 Java Class Using Images

```
/* attTest.java */  
  
package attPack;  
  
import javax.xml.transform.Source;  
import javax.xml.parsers.*;  
import org.w3c.dom.*;  
import javax.xml.transform.dom.*;  
import javax.xml.transform.stream.*;
```

CODE EXAMPLE 3-1 Java Class Using Images (*Continued*)

```
import org.xml.sax.*;
import java.io.*;
import javax.activation.*;
import javax.mail.internet.*;

public class attTest {

    /* Creates a new instance of attTest */
    public attTest() {
    }

    // input and output java.awt.Image
    public java.awt.Image echoImage(java.awt.Image image){
        java.awt.Image im = image;
        return im;
    }

    // mixed input
    public String echoMessage(java.awt.Image image, String
msg) {
        return msg;
    }

    // input and output xml
    public javax.xml.transform.Source
echoXML(javax.xml.transform.Source src) {
        return src;
    }

    // output java.awt.Image
    public java.awt.Image getImage(){
        //Point to a gif file in your own directory.
        String filename = "C:\\test\\microscope.gif";
        java.awt.Image image =
java.awt.Toolkit.getDefaultToolkit().getImage(filename);
        return image;
    }

}
```

FIGURE 3-26 shows an Explorer view of a web-centric web service named attWS and a test client named attWSClient, which call the methods of the attTest Java class.

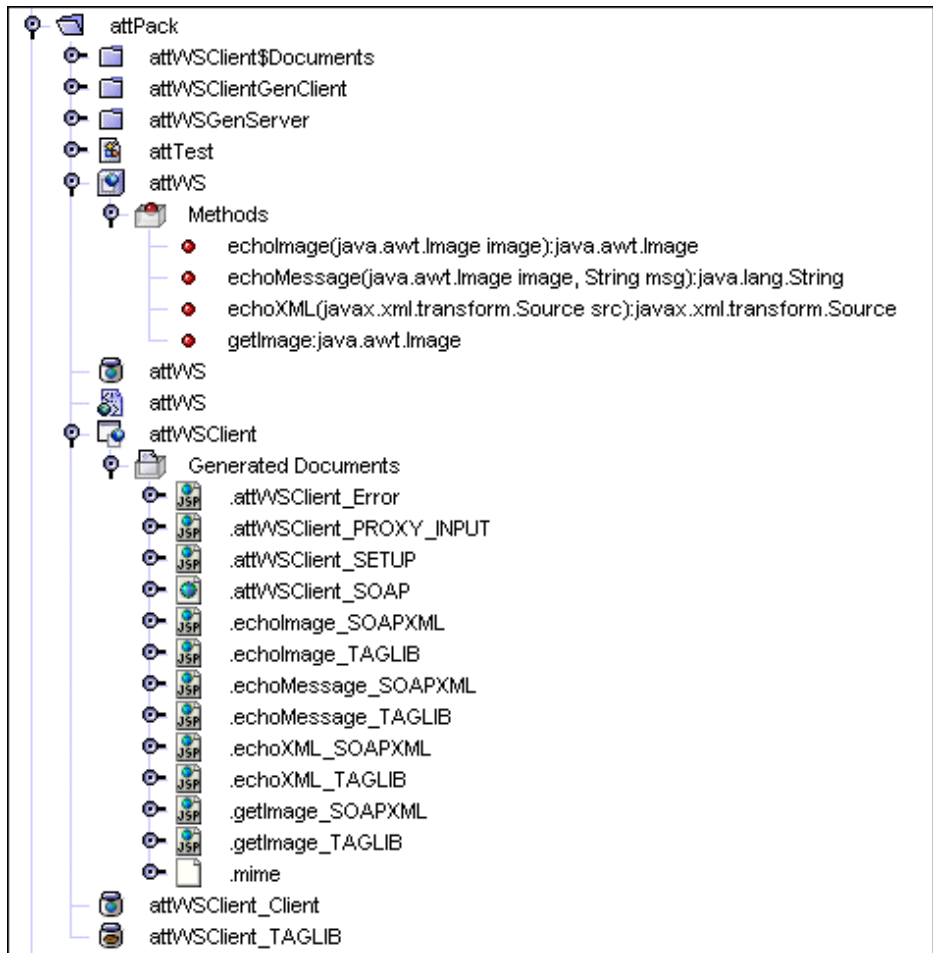


FIGURE 3-26 Explorer View of Web Service and Client That Process Images and Text

When you right-click the web service and choose Execute, the IDE opens a browser and displays the client's welcome page, as illustrated in FIGURE 3-27.

Image getImage()	
<input type="button" value="Invoke"/>	
Image echoImage(Image image_1)	
image_1:	<input style="width: 90%;" type="text"/> <input style="width: 10%; text-align: center;" type="button" value="Browse..."/>
<input type="button" value="Invoke"/> <input type="button" value="Reset"/>	
String echoMessage(Image image_1, String string_2)	
image_1:	<input style="width: 90%;" type="text"/> <input style="width: 10%; text-align: center;" type="button" value="Browse..."/>
string_2:	<input style="width: 90%;" type="text"/>
<input type="button" value="Invoke"/> <input type="button" value="Reset"/>	
Source echoXML(Source source_1)	
source_1:	<input style="width: 90%;" type="text"/> <input style="width: 10%; text-align: center;" type="button" value="Browse..."/>
<input type="button" value="Invoke"/> <input type="button" value="Reset"/>	

FIGURE 3-27 Client Welcome Page For Image and Text Processing

When you click Invoke for the `getImage` method, the client displays the page illustrated in FIGURE 3-28.

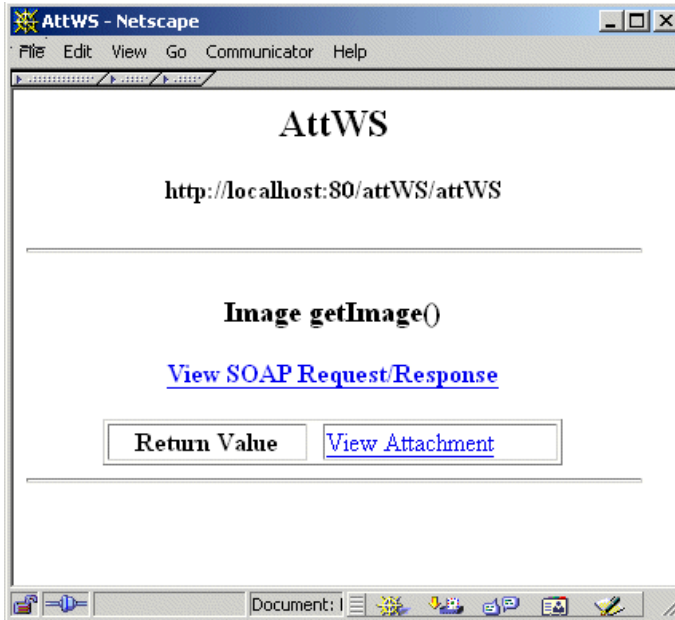


FIGURE 3-28 Client JSP Page For the `getImage` Method

When you click the View Attachment link, the client displays opens a new window and displays the image, as illustrated in FIGURE 3-29.

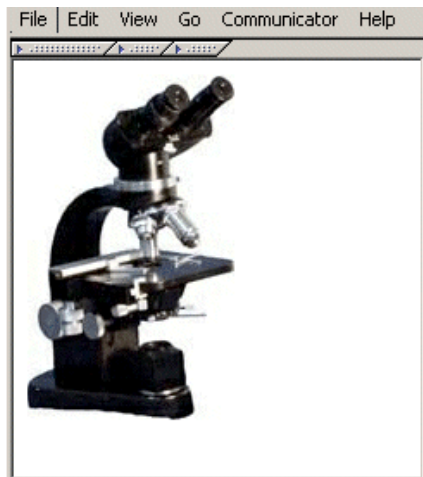


FIGURE 3-29 Client Display of Image

The DataHandler Property

The JAX-RPC specification, Section 7.2, states:

A remote method in a Java service endpoint interface may use the following Java types to represent MIME encoded content:

- Java classes based on the standard Java mapping of MIME types. Refer to section 7.5, “Mapping between MIME type and Java types” for more details.
- Java class `javax.activation.DataHandler` for content with any MIME type.

The JAX-RPC specification uses the JavaBeans Activation Framework to support various MIME content types. The `DataHandler` class provides a consistent interface the data represented in various MIME types....

(See “Software Versions” on page 39 for the URL of the JAX-RPC specification.)

The IDE provides a client property that you can use to cause a generated client to represent MIME encoded content with the `DataHandler` class. To do this:

1. **Right-click the client node, choose Properties, and change the Use DataHandler Only property from `False` (the default) to `True`.**
2. **Right-click the client node and choose Generate Client Files.**

FIGURE 3-30 shows the client class `attWSRPC` that the IDE generates when the `Use DataHandler Only` property is `False`.

```
// Helper class generated by xrpcc, do not edit.
// Contents subject to change without notice.

package attPack.attWSClientGenClient;

public interface AttWSRPC extends java.rmi.Remote {
    public java.awt.Image getImage() throws
        java.rmi.RemoteException;
    public java.awt.Image echoImage(java.awt.Image image_1) throws
        java.rmi.RemoteException;
    public java.lang.String echoMessage(java.awt.Image image_1, java.lang.String string_2) throws
        java.rmi.RemoteException;
    public javax.xml.transform.Source echoXML(javax.xml.transform.Source source_1) throws
        java.rmi.RemoteException;
}
```

FIGURE 3-30 Client Class `attWSRPC` With `Use DataHandler Only` Property `False`

FIGURE 3-31 shows the client class `attWSRPC` that the IDE generates when the `Use DataHandler Only` property is `True`.


```
// Helper class generated by xrpcc, do not edit.
// Contents subject to change without notice.

package attPack.attWSClientGenClient;

public interface AttWSRPC extends java.rmi.Remote {
    public javax.activation.DataHandler getImage() throws
        java.rmi.RemoteException;
    public javax.activation.DataHandler echoImage(javax.activation.DataHandler image_1) throws
        java.rmi.RemoteException;
    public java.lang.String echoMessage(javax.activation.DataHandler image_1, java.lang.String string_2) throws
        java.rmi.RemoteException;
    public javax.activation.DataHandler echoXML(javax.activation.DataHandler source_1) throws
        java.rmi.RemoteException;
}
```

FIGURE 3-31 Client Class attWSRPC With Use DataHandler Only Property True

For information about how to use the `DataHandler` class, see

http://java.sun.com/j2ee/sdk_1.2.1/techdocs/api/javax/activation/DataHandler.html

Stateful Web Services and Clients

For information about creating stateful web services and clients, see “Creating a Stateful Web Service” on page 59.

SOAP Message Handlers

For information about using SOAP message handlers with web services and clients, see Chapter 4.

Security

For information about implementing security for web services and clients, see Appendix A.

Using Message Handlers

This chapter provides an introduction to handlers and describes how the IDE supports the use of handlers with web services and clients.

A *JAX-RPC message handler* is a user-written class that can modify a SOAP message representing an RPC request or response. Handlers can be associated with a web service or a web service client. A handler has access to the body and header blocks of a message, but a handler does not have access to client or server application code. Handlers are normally designed to work with the header blocks and to supplement the processing done by web service or client application code.

A *handler chain* organizes handler classes so they execute in a designated sequence and can share data. When you add the first handler to a web service or client, the IDE automatically creates a handler chain. Any additional handlers become part of the same handler chain. There is at most one handler chain for a given web service or client.

Typical uses of handlers include:

- Logging and auditing
- Encryption and decryption
- Caching of data

For detailed information about how to code handlers, see Java API for XML-Based RPC (JAX-RPC) Specification 1.0: <http://java.sun.com/xml/jaxrpc>.

For detailed information about SOAP messages and headers, see the SOAP 1.1 specification: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>.

Note – This chapter should be read as an advanced topic. Most web services and clients do not require the use of handlers.

SOAP Message Handlers

The purpose of this section is to provide basic information about handlers so you can understand the handler features of the IDE, and to provide a lead-in to more detailed sources of information about handlers. If you are writing handler classes, you should also read the handler material in the JAX-RPC and SOAP specification documents.

SOAP Message Headers and Header Blocks

SOAP headers are an extension to a SOAP message. Headers are typically used to support higher level functions such as security and auditing.

In a SOAP message XML document, the `Header` element, if present, is the first immediate child element of the `SOAP Envelope` element. The immediate child elements of the `Header`, if present, are called *header entries* or *header blocks*.

CODE EXAMPLE 4-1 SOAP Header With One Header Block

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="some-URI"
    SOAP-ENV:mustUnderstand="1"
    SOAP:actor="www.xyz.com/actor1">
    5
  </t:Transaction>
</SOAP-ENV:Header>
```

CODE EXAMPLE 4-1 shows a SOAP header with one header block. The `actor` and `mustUnderstand` attributes are explained later in this chapter.

Request Handlers and Response Handlers

A handler class can process SOAP messages representing RPC requests and RPC responses. The main processing is done by two methods:

```
public boolean handleRequest (MessageContext context) {...}
public boolean handleResponse (MessageContext context) {...}
```

A `handleRequest` or `handleResponse` method gets the parts of the SOAP message from the `MessageContext` parameter. This includes the envelope, header, header elements, body, and body elements. See CODE EXAMPLE 4-2 for sample code.

CODE EXAMPLE 4-2 Sample Handler Code To Extract Parts of a SOAP Message

```
public boolean handleRequest (MessageContext context) {  
  
    try {  
        SOAPMessageContext smc = (SOAPMessageContext) context;  
        SOAPMessage msg = smc.getMessage();  
        SOAPPart sp = msg.getSOAPPart();  
        SOAPEnvelope se = sp.getEnvelope();  
        SOAPHeader shd = se.getHeader();  
  
        SOAPBody sb = se.getBody();  
        java.util.Iterator childElems = sb.getChildElements();  
        SOAPElement child;  
        StringBuffer message = new StringBuffer();  
        while (childElems.hasNext()) {  
            child = (SOAPElement) childElems.next();  
            message.append(new Date().toString() + "--");  
            formLogMessage(child, message);  
        } ...  
    }  
}
```

A `handleRequest` or `handleResponse` method can modify the headers or body of a message. The method can *consume* a header block (remove it from the message) or *add* a header block.

For example, a `handleRequest` or `handleResponse` method can write to an external log or audit file, or it can call routines to encrypt or decrypt the body of a message. A handler is not normally used to analyze or modify elements within the body of a message.

The action taken by a `handleRequest` or `handleResponse` method can depend on the content of the header or data in the body of the message, such as the name of the web service method in an RPC request or response.

There are four places in a client-service interaction where handlers can execute.

Client-side handler:

- A request handler is invoked before the client's RPC request is sent to a service.
- A response handler is invoked before the service's RPC response is delivered to the client.

Service-side handler:

- A request handler is invoked before a client's RPC request is delivered to the service.
- A response handler is invoked before the service's RPC response is sent to the client.

A single handler class can have both `handleRequest` and `handleResponse` methods.

See “Sample Handler Code” on page 148 for an example of a handler with a `handleRequest` method.

Note – A JAX-RPC handler class is required to implement the interface `javax.xml.rpc.handler.Handler` interface.

Handler chains

A handler chain for a given web service or client is illustrated schematically in CODE EXAMPLE 4-3:

CODE EXAMPLE 4-3 Schematic Description of a Handler Chain

```
handler chain
  roles
    "http://acme.org/auditing"
    "http://acme.org/morphing"
  handler class "acme.MyHandler1"
  headers
    "ns1:foo ns1:bar"
    "ns2:foo ns2:bar"
  properties
    name="property1" value="xyz1"
    name="property2" value="xyz2"
    name="property3" value="xyz3"
  handler class "acme.MyHandler2"
```

The handler chain in the example has two handler classes, `acme.MyHandler1` and `acme.MyHandler2`. The class `acme.MyHandler1` is configured with three properties and two header blocks. Two SOAP actor roles are associated with the handler chain.

The uses of handler properties, header blocks, and SOAP actor roles are described in this chapter. The IDE manages handler chains for you. No coding is necessary.

SOAP Actor Roles

You can use SOAP actor roles to ensure that designated header blocks are processed by handlers or by the ultimate destination (web service or client) of a message.

A SOAP `actor` is an attribute of a SOAP header block, typically used in conjunction with the `mustUnderstand` attribute, as illustrated in CODE EXAMPLE 4-1.

Suppose that a header block has the following attributes:

```
SOAP-ENV:mustUnderstand="1"  
SOAP:actor="www.xyz.com/actor1"
```

If a SOAP message with this header block is processed by a handler chain that has the actor role `www.xyz.com/actor1`, the header block must be consumed by one of the handlers in the handler chain or by the destination web service or client. If the header block is not consumed, the SOAP runtime throws an exception before any business method is called by the destination web service or client.

If the `mustUnderstand` attribute is omitted or has the value `"0"`, the SOAP runtime does not enforce the consumption or use of the header block, regardless of any `actor` attribute in the header block.

The SOAP 1.1 specification explains that tagging elements with the attribute `mustUnderstand="1"` is intended to ensure that developers who might not understand the purpose of those elements will not erroneously ignore them. (Handlers might not be written by the same developers who design the SOAP message and its header blocks.)

Handler Properties.

A handler property can be used to configure a handler. The property and its value are passed to a handler instance at initialization time, through the `HandleInfo` parameter. CODE EXAMPLE 4-6 shows how a handler property might be used. The IDE enables you to add properties to a handler during development. (See “Adding a Handler Property” on page 141.)

Two different web services or clients might use the same handler, but with different property values, such as different log files or different encryption and decryption routines.

Getting Header Blocks at Handler Initialization

The action taken by a handler can depend on the header blocks passed in RPC requests and responses, including header blocks added to a message by other handlers executed earlier in the same handler chain. A handler might use some headers blocks and ignore others.

Note – Header blocks are the immediate child elements of the Header element (if present) in a SOAP message. See “SOAP Message Headers and Header Blocks” on page 132.

When you add a handler to a web service or client, you can configure the handler to process certain header blocks. The IDE enables you to provide a list of header block names during the development of a web service or client. (See “Adding a SOAP Header Block To a Handler” on page 144.)

The header block names that you set in the IDE are passed to the handler in a parameter of type `HandlerInfo`. The code fragment in CODE EXAMPLE 4-4 shows how the handler can obtain the array of headers to process. The `requestHandler` or `responseHandler` method can iterate through the `QName` array to get the header block names.

CODE EXAMPLE 4-4 Sample Handler Code To Initialize List of Header Blocks

```
public class StockServerMailHandler
    implements javax.xml.rpc.handler.Handler
{
    private QName[] headers;

    public StockServerMailHandler () {
    }

    public void init(HandlerInfo config) {
        headers = config.getHeaders();
    }
}
```

A runtime SOAP message in an RPC request or response might have many header blocks. They are all available to the handler through a parameter of type `MessageContext` in the `handleRequest` and `handleResponse` methods, as illustrated in CODE EXAMPLE 4-7. However, you can code the handler so that it only processes the header blocks whose names are gathered from the `HandlerInfo` parameter during initialization.

Adding a Header Block to a SOAP Message

Handlers are not intended to be a major source of new headers, but there are some scenarios in which you must add a header block to a SOAP message. Suppose, for example, that you want a given header block to be processed by more than one handler or by a handler and the target web service or client. The problem is that if a handler consumes a header block, it is no longer part of the SOAP message. You can solve this problem by having your handler copy the header block before consuming it, add the copy to the SOAP message, and then pass the message up the chain.

The code fragment in CODE EXAMPLE 4-5 shows how to add a header block to a message. The header block must be a child element of a header element, so if the message does not already have a header element, the code adds one.

CODE EXAMPLE 4-5 Sample Handler Code To Add a Header To a SOAP Message

```
package stock;

import java.util.Hashtable;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import java.util.*;

public class AddHeaderHandler
    implements javax.xml.rpc.handler.Handler {

    private Map props;

    public AddHeaderHandler () {
    }
    public void init(HandlerInfo config) {
        props= config.getHandlerConfig();
    }
    public boolean handleFault(MessageContext context) {
        return true;
    }

    public boolean handleRequest (MessageContext context) {

        try {
            SOAPMessageContext smc = (SOAPMessageContext) context;
            SOAPMessage msg = smc.getMessage();
            SOAPPart sp = msg.getSOAPPart();
            SOAPEnvelope se = sp.getEnvelope();
            SOAPHeader sh = se.getHeader();
            if(sh == null){
```

CODE EXAMPLE 4-5 Sample Handler Code To Add a Header To a SOAP Message (*Continued*)

```
        sh = se.addHeader();
    }
    //add header elements
        Name headerName =
se.createName("myNewHeader");
        SOAPHeaderElement she =
sh.addHeaderElement(headerName);
    }
    catch (Exception ex) {
        //ex.printStackTrace();
    }
    return true;
}
public boolean handleResponse (MessageContext context) {
    return true;
}

public javax.xml.namespace.QName[] getHeaders() {
    return null;
}

public void destroy() {
}
}
```

You can also add header elements to a SOAP message through WSDL code. See the WSDL 1.1 specification: http://www.w3.org/TR/wsd1#_soap:header.

Using Handlers In the IDE

The procedures in this section assumes that you have created a web service, a client, and handler classes in the IDE.

Adding Handlers To a Web Service or Client

To add handler classes to a web service or client:

1. Right-click the web service or client node and choose Properties.

The SOAP Message Handlers properties shows the number of handlers currently associated with the web service, as illustrated in FIGURE 4-1.

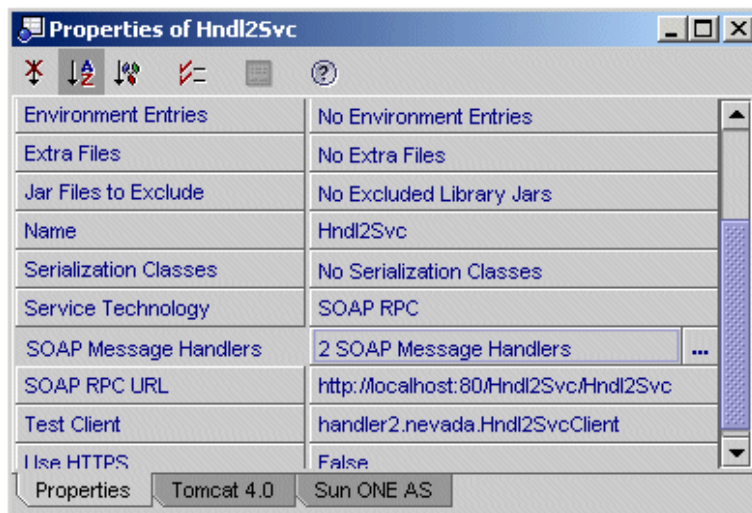


FIGURE 4-1 SOAP Message Handlers Property

2. Click the value of the Soap Message Handlers property, and click the ellipsis (...) button.

The IDE displays the SOAP Message Handlers dialog box, as illustrated in FIGURE 4-2. In this example two handler classes are already associated with the web service.

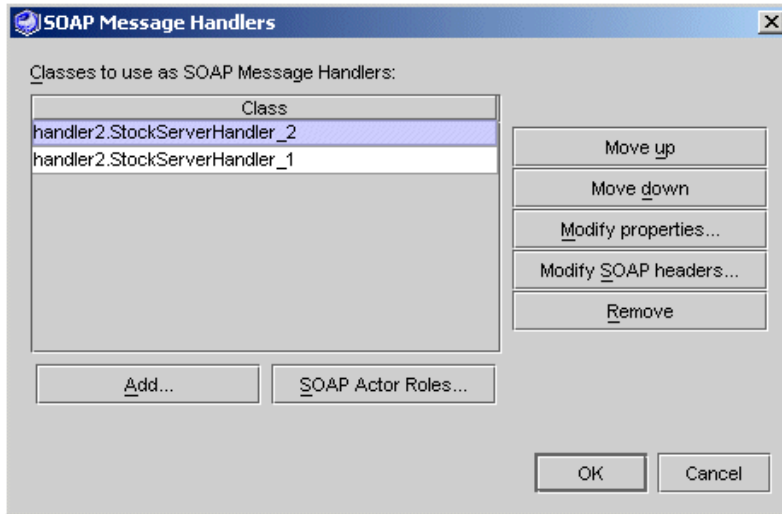


FIGURE 4-2 SOAP Message Handlers Dialog Box

The SOAP Message Handlers dialog box provides several features to help you manage handlers and handler chains:

- The Add and Remove buttons enable you to add or remove handler classes from the handler chain associated with a web service or client.
- The Move up and Move down buttons enable you to control the execution sequence of handler classes within the handler chain. Handlers at the top of the displayed list execute first.
- The Modify properties button enables you to add a property name and value to a handler.
- The Modify SOAP headers button enables you to specify which SOAP header blocks are processed by a handler.
- The SOAP Actor Roles button enables you to associate one or more SOAP actor roles with the handler chain.

Each of these features is described in this chapter.

3. Click Add, select one or more desired handlers from the chooser, and click OK.

The selected handlers appear in the SOAP Message Handlers dialog box and become part of the handler chain for the web service or client.

Note – A JAX-RPC handler class is required to implement the interface `javax.xml.rpc.handler.Handler` interface. If you try to add a handler that does not implement this interface, the IDE will display an error message.

Adding a Handler Property

To add a property and value to a handler:

- 1. Right-click the web service or client node and choose Properties.**

The SOAP Message Handlers properties shows the number of handlers currently associated with the web service, as illustrated in FIGURE 4-1.

- 2. Click the value of the Soap Message Handlers property, and click the ellipsis (...) button.**

The IDE displays the SOAP Message Handlers dialog box, as illustrated in FIGURE 4-3. In this example the web service has one handler class.

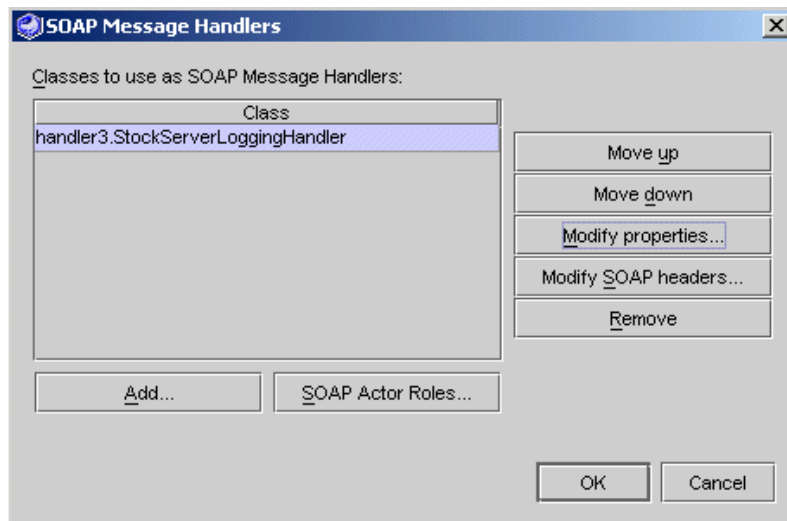


FIGURE 4-3 SOAP Message Handlers Dialog Box

- 3. Highlight a handler and click Modify properties.**

The Set Properties dialog box appears, as illustrated in FIGURE 4-4. In this example, the handler has a property named `logFile`, whose value is the fully-qualified name of the file to which the handler writes log information.

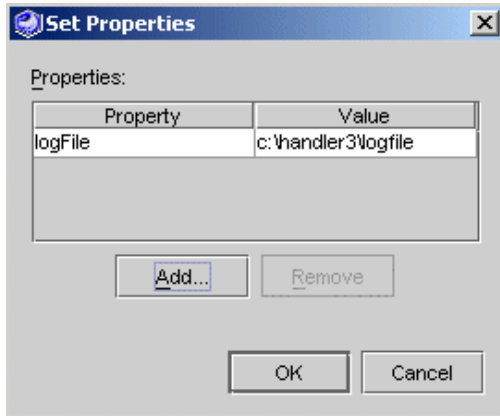


FIGURE 4-4 SOAP Message Handlers Set Properties Dialog Box

The dialog box displays properties and values for the selected handler. You can remove a property or add a property.

4. Click Add.

The Add Properties dialog box appears, as illustrated in FIGURE 4-5. In this example, a new property is added named `auditFile`, whose value is the fully-qualified name of the file to which the handler writes audit information.

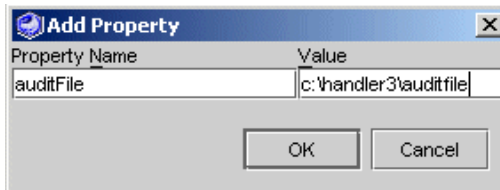


FIGURE 4-5 SOAP Message Handlers Add Property Dialog Box

5. Enter a property name and value, and click OK.

The Set Properties dialog box appears showing the new property and value, as illustrated in FIGURE 4-6.

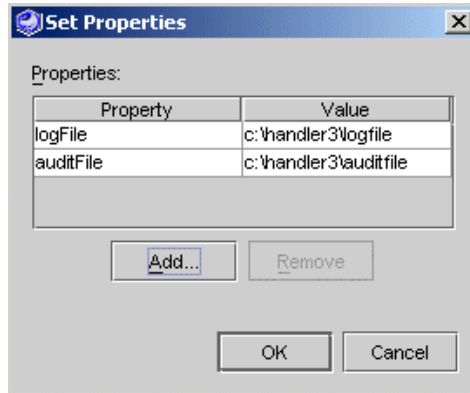


FIGURE 4-6 SOAP Message Handlers Set Properties Dialog Box

See **CODE EXAMPLE 4-6** for sample handler code that initializes a handler with the value of the `logFile` property.

CODE EXAMPLE 4-6 Sample Handler Code to Use the `logFile` Property Value

```

public class StockServerLoggingHandler
    implements javax.xml.rpc.handler.Handler
{
    private PrintStream pout;
    private QName[] headers;

    public StockServerLoggingHandler () {
    }
    public void init(HandlerInfo config) {
        Map props = config.getHandlerConfig();
        headers = config.getHeaders();
        //Get the value of the property "logFile"
        String logFileName = null;
        if(props != null)
            logFileName = (String)props.get("logFile");

        File logFile = new File(logFileName);
        if(!logFile.exists()){
            try{
                logFile.createNewFile();
            }
            catch(Exception e){
                e.printStackTrace();
            }
        }
    }
    try{

```

CODE EXAMPLE 4-6 Sample Handler Code to Use the `logFile` Property Value (*Continued*)

```
        FileOutputStream out = new FileOutputStream(logFile,
true);
        pout = new PrintStream(out);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Adding a SOAP Header Block To a Handler

To add a SOAP header block to a handler:

1. Right-click the web service or client node and choose Properties.

The SOAP Message Handlers properties shows the number of handlers currently associated with the web service, as illustrated in FIGURE 4-1.

2. Click the value of the Soap Message Handlers property, and click the ellipsis (...) button.

The IDE displays the SOAP Message Handlers dialog box, as illustrated in FIGURE 4-3.

3. Highlight a handler and click Modify SOAP headers.

The Set SOAP Header dialog box appears, as illustrated in FIGURE 4-7.

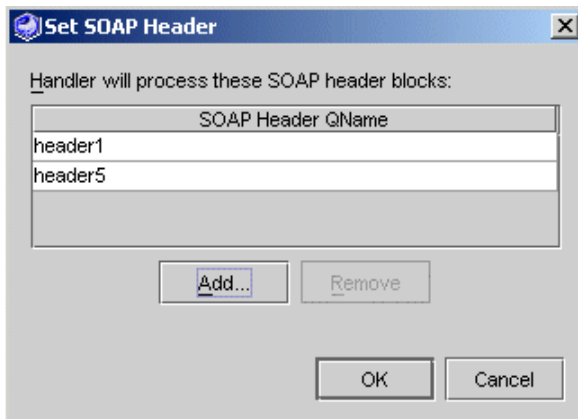


FIGURE 4-7 Set SOAP Header Dialog Box

The dialog box displays SOAP header blocks (SOAP Header QNames) for the selected handler. You can remove a QName or add a QName.

4. Click Add.

A dialog box appears, as illustrated in FIGURE 4-8.

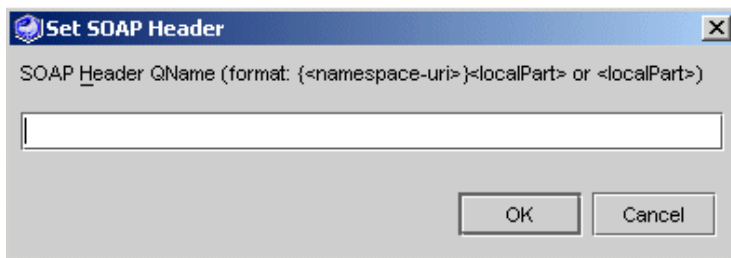


FIGURE 4-8 Set SOAP Header QName Dialog Box

5. Enter the SOAP Header QName and click OK.

The new SOAP header block name appears in the Set SOAP Header dialog box, as illustrated in FIGURE 4-7.

Note – If the header block has a namespace qualifier, it should be entered in the following syntax: `{namespace-uri}localpart`. The syntax is displayed in the Set SOAP Header dialog box in FIGURE 4-8. For example, `{foo/myNS}header1`.

These header names are available to a handler instance when it is configured at runtime. See “Getting Header Blocks at Handler Initialization” on page 136 for information and a code sample, showing how this feature is used.

Note – This list of SOAP header block names is static information that is set up at development time for a given web service or client, and is passed to a handler when it is initialized at runtime. For information about how a handler can add a header block to a SOAP message dynamically at runtime, see “Adding a Header Block to a SOAP Message” on page 137.

Setting a SOAP Actor Role For a Handler Chain

The IDE enables you to associate one or more SOAP actor roles with a handler chain. See “Handler chains” on page 134, “SOAP Actor Roles” on page 135, and “Enforcing the Processing of Header Blocks” on page 147 for an explanation of how SOAP actor roles are used.

1. Right-click the web service or client node and choose Properties.

The SOAP Message Handlers properties shows the number of handlers currently associated with the web service, as illustrated in FIGURE 4-1.

2. Click the value of the Soap Message Handlers property, and click the ellipsis (...) button.

The IDE displays the SOAP Message Handlers dialog box, as illustrated in FIGURE 4-3.

You can set SOAP actor roles for the handler chain, not for individual handlers.

3. Click SOAP Actor Roles.

The IDE displays the Configure SOAP Actor Roles dialog box, as illustrated in FIGURE 4-9.

The dialog box displays SOAP actor roles for the handler chain. You can remove an actor role or add an actor role.

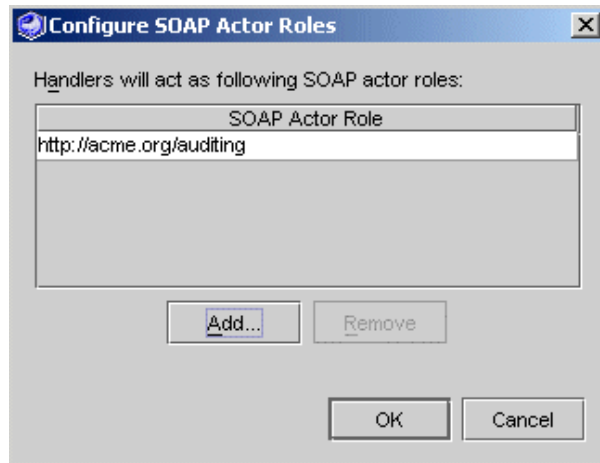


FIGURE 4-9 Configure SOAP Actor Roles Dialog Box

4. Click Add.

A dialog box appears, as illustrated in FIGURE 4-10.



FIGURE 4-10 Add SOAP Actor Role Dialog Box

5. Enter the desired SOAP actor role and click OK.

The SOAP actor role must be expressed as a valid URI.

The new SOAP actor role appears in the Set SOAP Header dialog box, as illustrated in FIGURE 4-11.

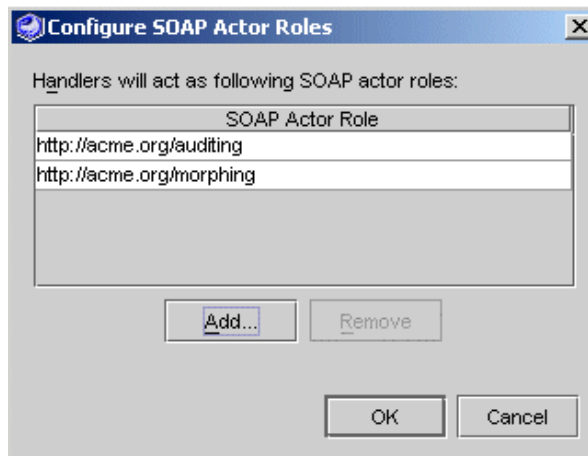


FIGURE 4-11 Configure SOAP Actor Roles Dialog Box

Enforcing the Processing of Header Blocks

This section explains how you can ensure that a handler chain processes mandatory headers that are intended for certain SOAP actors.

A handler chain can be configured to act in the role of one or more SOAP actors, expressed as a valid URI. A handler in the handler chain can be configured to process header blocks with specified qualified names (QNames) and return the header blocks in the handler's `getHeaders()` method.

When a SOAP message is processed, the handler chain does the following:

1. Determines what actor roles (from the configured actors) the handler chain is to play. It also keeps track of all header blocks that each handler in the chain intends to process.
2. Examines the SOAP message header blocks (if any) and determines which ones must be processed by the handler chain, by finding all header blocks that are mandatory for its configured actors. It does this by looking at the values of the "actor" and "mustUnderstand" attributes of the header blocks.
3. Checks if each mandatory header block is in the list that the handlers intend to process, that is, it is "understood" by the handler chain. If a mandatory header block is not "understood" by the chain, a `SOAPFaultException` is thrown.
4. Processes the header blocks that are intended for the chain by invoking the individual handlers.

To enforce the processing of mandatory header blocks by a handler chain:

- 1. Add the appropriate actor roles to the handler chain.**

The procedure is described in "Setting a SOAP Actor Role For a Handler Chain" on page 145.

- 2. Specify the handlers' intent to process certain headers by configuring each handler with the designated header blocks.**

The procedure is described in "Adding a SOAP Header Block To a Handler" on page 144.

- 3. In the handler's code, return the designated header blocks in the `getHeaders()` method.**

You can get the header blocks from the `HandlerInfo` that is passed to the `init()` method of the handler.

- 4. Process the header blocks in the handler code.**

Sample Handler Code

The web service handler in CODE EXAMPLE 4-7 processes an RPC request message as follows:

1. During handler initialization, the `init` method gets a property that specifies the name of a file to be the target for logging messages.
2. The `handleRequest` method builds a log message consisting of the date and time followed by the data in the request

3. The `handleRequest` method writes the log message to the target log file.

CODE EXAMPLE 4-7 Sample Handler with `handleRequest` Method

```
package stock;

import java.util.Hashtable;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
import javax.xml.namespace.*;

public class StockServerLoggingHandler
    implements javax.xml.rpc.handler.Handler
{
    private PrintStream pout;
    private QName[] headers;

    public StockServerLoggingHandler () {
    }

    public void init(HandlerInfo config) {
        Map props = config.getHandlerConfig();
        headers = config.getHeaders();

        //Get the value of the property "logFile"

        String logFileName = null;
        if(props != null)
            logFileName = (String)props.get("logFile");

        //if logFile property is not defined, create default directory and log file name

        if(props == null || !props.containsKey("logFile") || logFileName == null
            || logFileName.length() == 0){
            logFileName = System.getProperty("user.home") + File.separator
                + "stockhandler.log";
        }

        // create log file if it does not already exist

        File logFile = new File(logFileName);
        if(!logFile.exists()){
            try{
```

CODE EXAMPLE 4-7 Sample Handler with `handleRequest` Method (Continued)

```
        logFile.createNewFile();
    }
    catch(Exception e){
        e.printStackTrace();
    }
}

// associate output print stream with log file

try{
    FileOutputStream out = new FileOutputStream(logFile, true);
    pout = new PrintStream(out);
} catch(IOException e){
    e.printStackTrace();
}

}

public boolean handleFault(MessageContext context) {
    return true;
}

public boolean handleRequest (MessageContext context) {

// get SOAP message and extract the envelope, header, body, and body elements

    try {
        SOAPMessageContext smc = (SOAPMessageContext) context;
        SOAPMessage msg = smc.getMessage();
        SOAPPart sp = msg.getSOAPPart();
        SOAPEnvelope se = sp.getEnvelope();
        SOAPHeader shd = se.getHeader();

        SOAPBody sb = se.getBody();
        java.util.Iterator childElems = sb.getChildElements();
        SOAPElement child;
        StringBuffer message = new StringBuffer();

// iterate through body elements, formatting a log message for each element

        while (childElems.hasNext()) {
            child = (SOAPElement) childElems.next();
            message.append(new Date().toString() + "--");
            formLogMessage(child, message);
        }

// extract headers
```

CODE EXAMPLE 4-7 Sample Handler with `handleRequest` Method (Continued)

```
        if(shd != null){
            java.util.Iterator iter =
                shd.extractHeaderElements(SOAPConstants.URI_SOAP_ACTOR_NEXT);
            while(iter.hasNext()){
                SOAPHeaderElement el = (SOAPHeaderElement)iter.next();
                Name n = el.getElementName();

                for(int i = 0; i < headers.length; i++)
                {
                    QName header = headers[i];
                    String qNameSpace = header.getNamespaceURI();
                    String nNameSpace = n.getURI();
                    //System.out.println("Name.getURI(): " + nNameSpace);
                    String qLocalPart = header.getLocalPart();
                    String nLocalPart = n.getLocalName();

// print headers

                    if(qNameSpace != null &&
                        nNameSpace != null &&
                        qNameSpace.equals(nNameSpace) &&
                        qLocalPart.equals(nLocalPart) )
                    {
                        message.append(" " + n.getQualifiedName() + ":");
                        Iterator iter2 = el.getChildElements();
                        while(iter2.hasNext())
                        {
                            Object sel = iter2.next();
                            if(sel instanceof SOAPElement)
                            {
                                message.append(" " + ((SOAPElement)sel).
                                    getElementName().getLocalName()+ ": ");
                            }
                            else if(sel instanceof Text)
                            {
                                message.append(((Text)sel).getValue());
                            }
                        }
                    }
                }
            }
            pout.println(message.toString());
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

CODE EXAMPLE 4-7 Sample Handler with `handleRequest` Method (*Continued*)

```
    return true;
}

// format a log message

private void formLogMessage (SOAPElement child, StringBuffer message) {
    message.append(child.getElementName().getLocalName());
    message.append(child.getValue() != null ? ":" + child.getValue() + " "
        : " ");

    try{
        java.util.Iterator childElems = child.getChildElements();
        while (childElems.hasNext()) {
            Object c = childElems.next();
            if(c instanceof SOAPElement)
                formLogMessage((SOAPElement)c, message);
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}

public boolean handleResponse (MessageContext context) {
    return true;
}

public javax.xml.namespace.QName[] getHeaders() {

// return headers configured on the node

    return headers;
}

public void destroy() {
    pout.close();
}
}
```


Developing XML Operations (Deprecated)

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.

Note – XML operations are deprecated in this release of the IDE, and might not be supported in future releases. You can build an application without XML operations, using direct method calls and stateful web services and clients if necessary. The need for XML operations has been eliminated by the evolution of web services features.

This chapter provides a conceptual overview, a description of the tools, and an explanation of how to create and edit XML operations.

Overview of XML Operations

The IDE 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. You define an XML operation using a codeless editor, described later in this chapter.

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.

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 action 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.

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 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 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 5-1 shows an XML operation named `productName` that calls methods on three different objects.

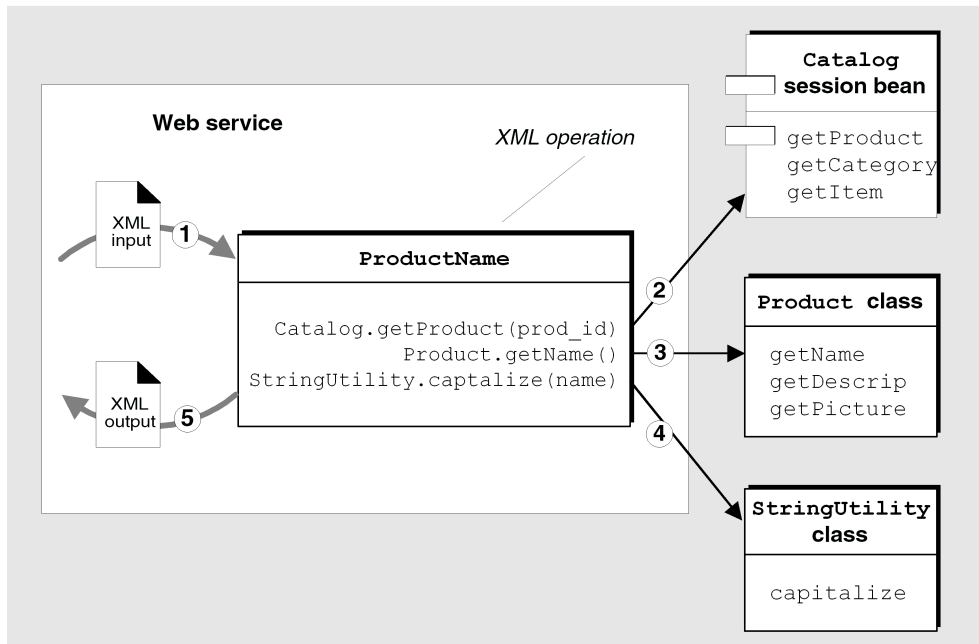


FIGURE 5-1 XML Operation Calling Multiple Methods to Fulfill a Client Request

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.

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. You right-click the node and choose 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 163 for explanation of how to do this.)

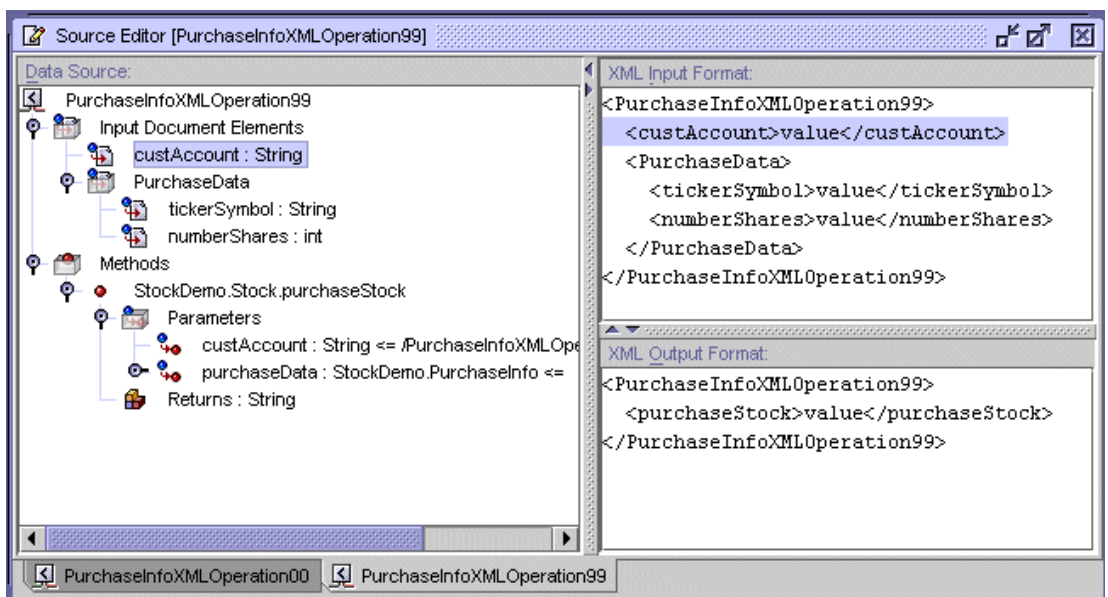


FIGURE 5-2 XML Operation Source Editor, Showing Complex Input

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 5-2 and FIGURE 5-3 show XML operations displayed in the Source Editor.

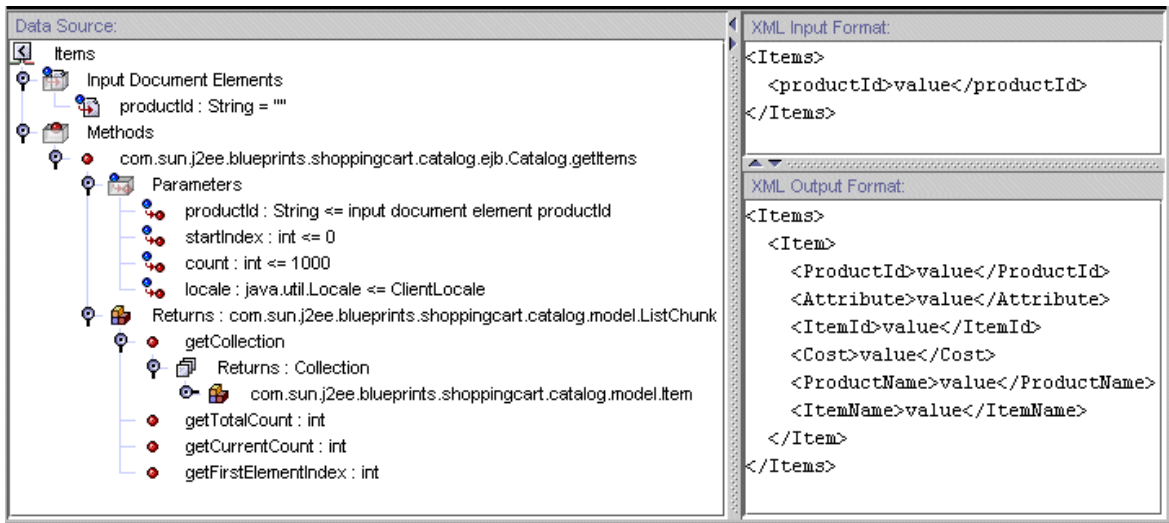


FIGURE 5-3 XML Operation Source Editor, Showing Complex Output

The Source Editor displays an XML operation in three panes:

- **Data Source pane.** This pane enables you 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. This document 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 5-4 shows a cropped view of the Source Editor with the Input Document Elements node expanded.

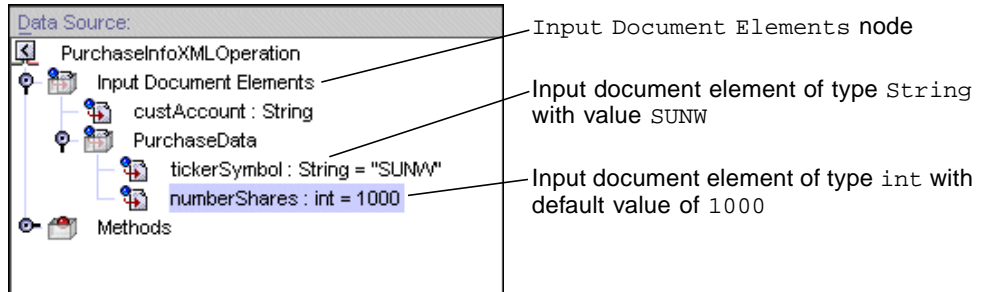


FIGURE 5-4 Input Document Elements Node

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 168)
- Delete elements (see “Deleting a Method or Input Document Element” on page 172)
- Rename elements (see “Renaming an Input Document Element” on page 169)
- Rearrange elements (see “Reordering a Method or Input Document Element” on page 172)
- Provide a default value for an element (see “Giving an Input Document Element a Default Value” on page 170)
- Make an element permanent (see “Making an Input Document Element Permanent” on page 171)

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 method 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 166, “Reordering a Method or Input Document Element” on page 172, and “Deleting a Method or Input Document Element” on page 172.

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 172. For information on target objects, see “Instantiating Objects and Resolving References” on page 180. For information on system shared objects, see “System Shared Objects” on page 177.

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 177.

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 175.

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 42 and “Adding Operations to a Web Service” on page 45.

4. Test the XML operation in the web service.

For more information on this topic, see “Testing a Web Service” on page 58.

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 5-5.

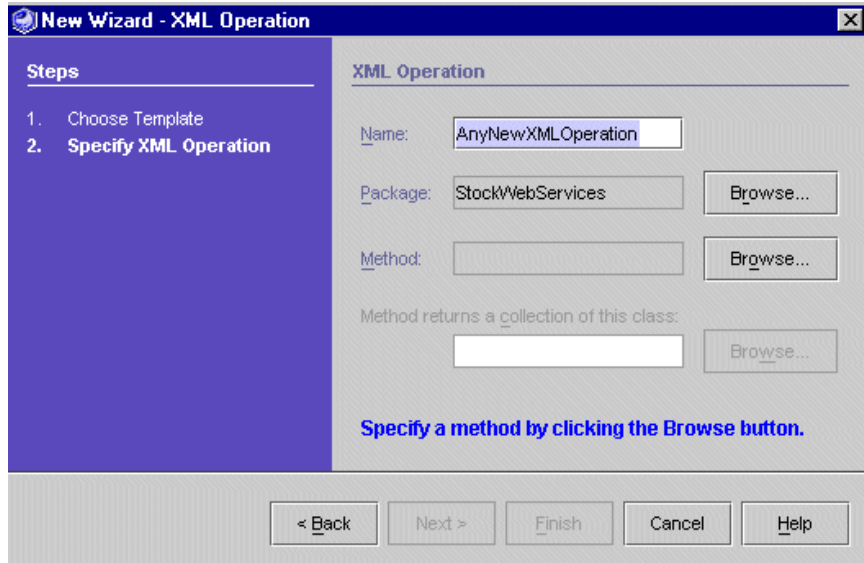


FIGURE 5-5 New XML Operation Dialog Box

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 5-6.

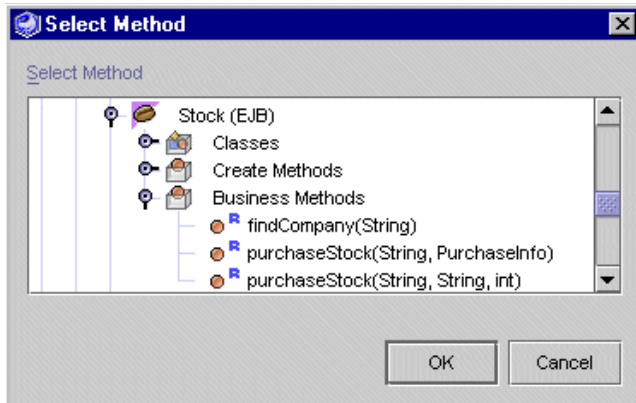



FIGURE 5-6 Select Method Dialog Box

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 166.

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.

a. 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 chooser opens.

b. Use the file chooser 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 5-3). For information on editing your XML operation, see "Editing an XML Operation" on page 166.

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 interface and remote interface 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 42.

To generate XML operations from an enterprise bean:

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

A file chooser 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 5-7.

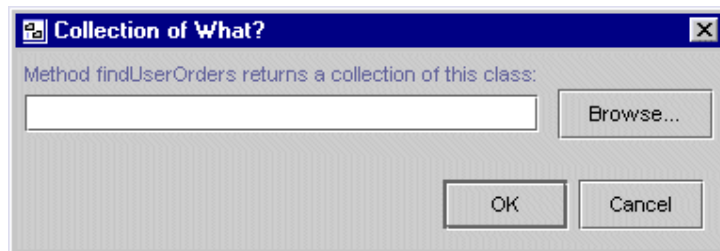


FIGURE 5-7 Collection of What? Dialog Box

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 chooser 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 5-8.

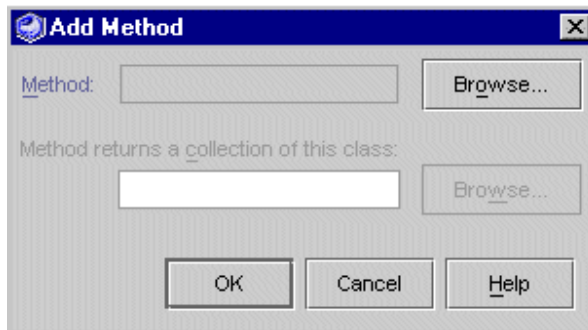


FIGURE 5-8 Add Method to XML Operation Dialog Box

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

The Select Method dialog box is displayed, as illustrated in FIGURE 5-9.

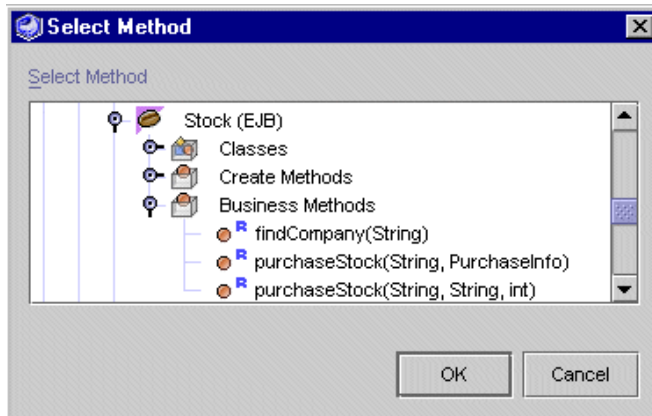



FIGURE 5-9 Select Method Dialog Box

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.

a. Click the Browse button in the Add Method dialog box, next to the field labeled Method Returns a Collection of This Class.

A file chooser opens.

b. Use the file chooser 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 5-10.

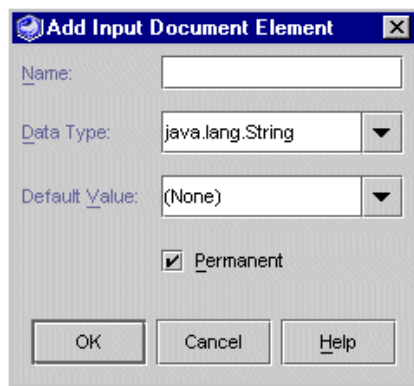


FIGURE 5-10 Add Input Document Element Dialog Box

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`) do 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 5-11.**
- 2. Click the Name property.**

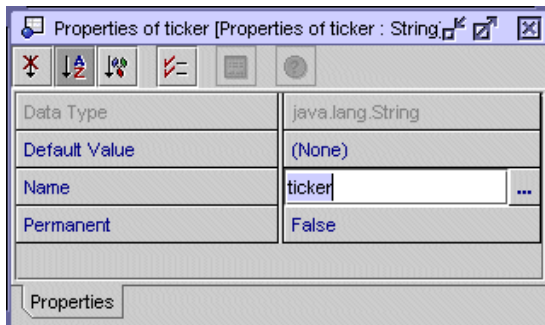


FIGURE 5-11 Input Document Element Properties Dialog Box

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 5-12.**
2. **Click the Tag Name property.**

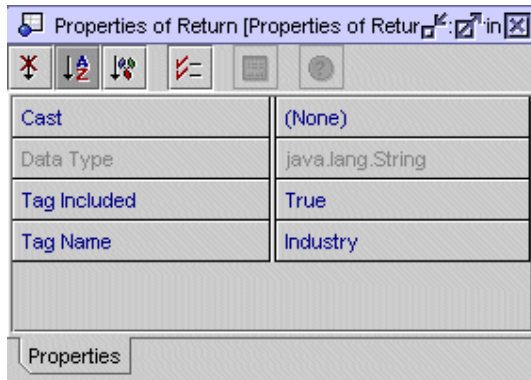


FIGURE 5-12 Output Document Element Properties Dialog Box

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 5-13.**
2. **Click the Default Value property.**

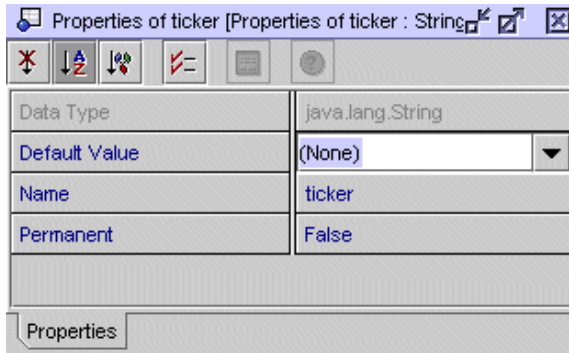


FIGURE 5-13 Input Document Element Properties (Default Value)

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 5-13.

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 172 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 5-14.

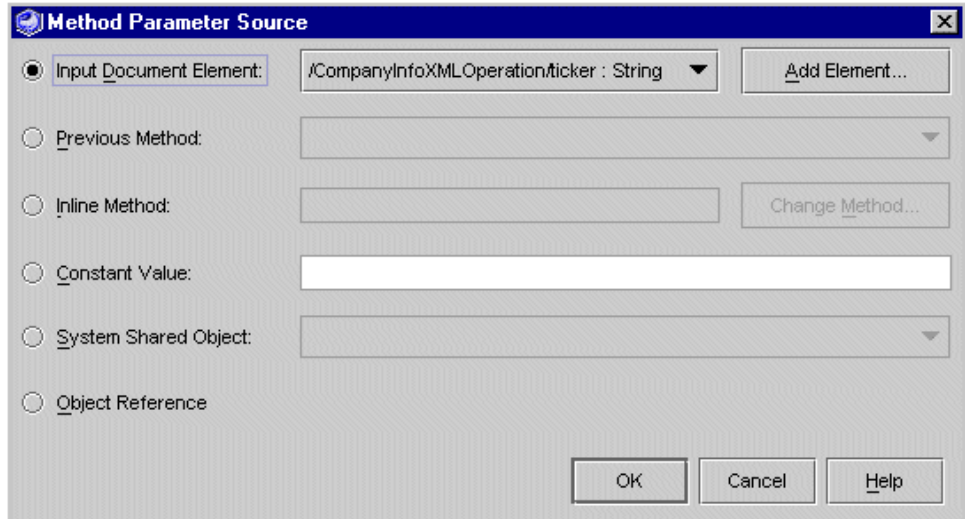


FIGURE 5-14 Method Parameter Source Dialog Box

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>Select this radio button if you want to map the parameter to one of these data types.</p> <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>
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 180.</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	<p>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.</p>
Previous Method or Inline Method	<p>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.</p>
Constant Value	<p>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.</p>
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 5-15 shows a class node selected in the Data Source pane and its corresponding element highlighted in the XML Output Format pane.

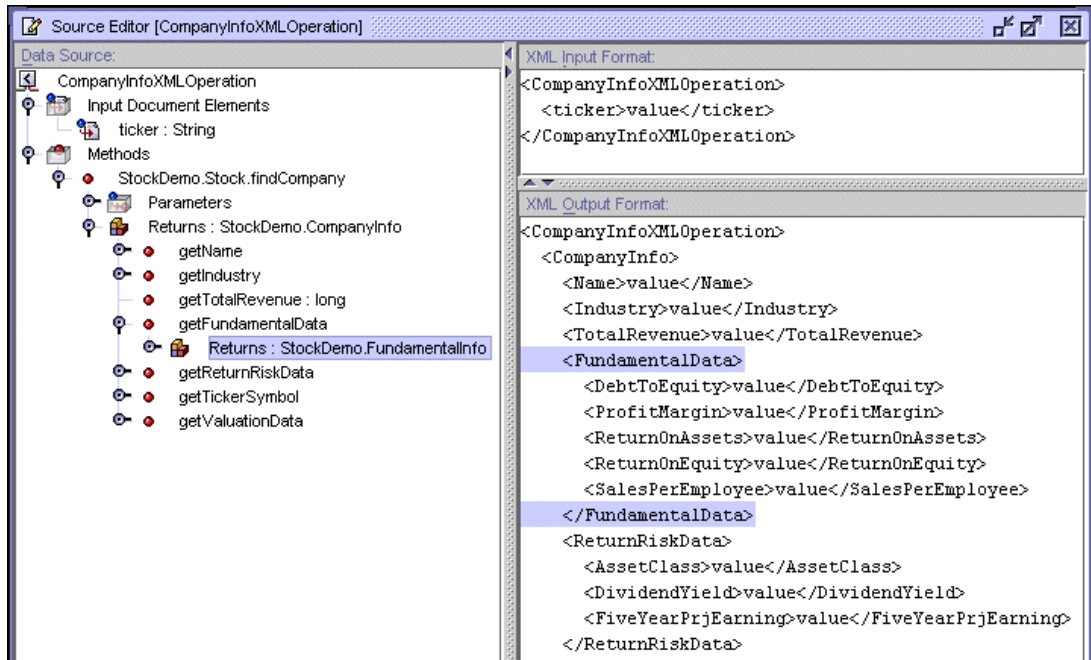


FIGURE 5-15 Source Editor: (Excluding an Output Element)

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 172 and FIGURE 5-14.

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, 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 166 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 5-16.

c. Select Previous Method.

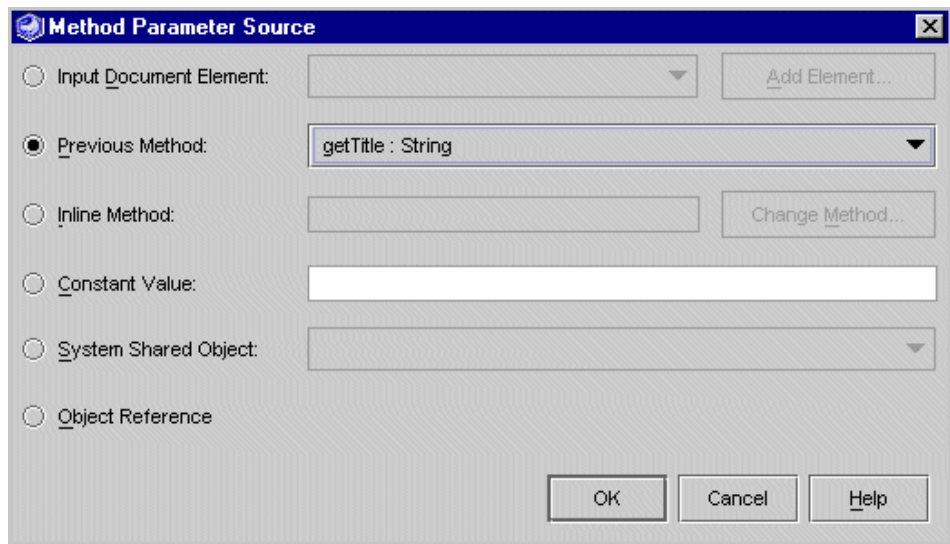


FIGURE 5-16 Method Parameter Source Dialog Box

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

See FIGURE 5-14 and “Mapping a Method Parameter to a Source” on page 172 for more information on mapping parameters.

5. To exclude the return value of the data-component method call from the XML output document, right-click the method that calls the data component and choose 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 170 for information on how to do this.

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 5-17) is displayed.

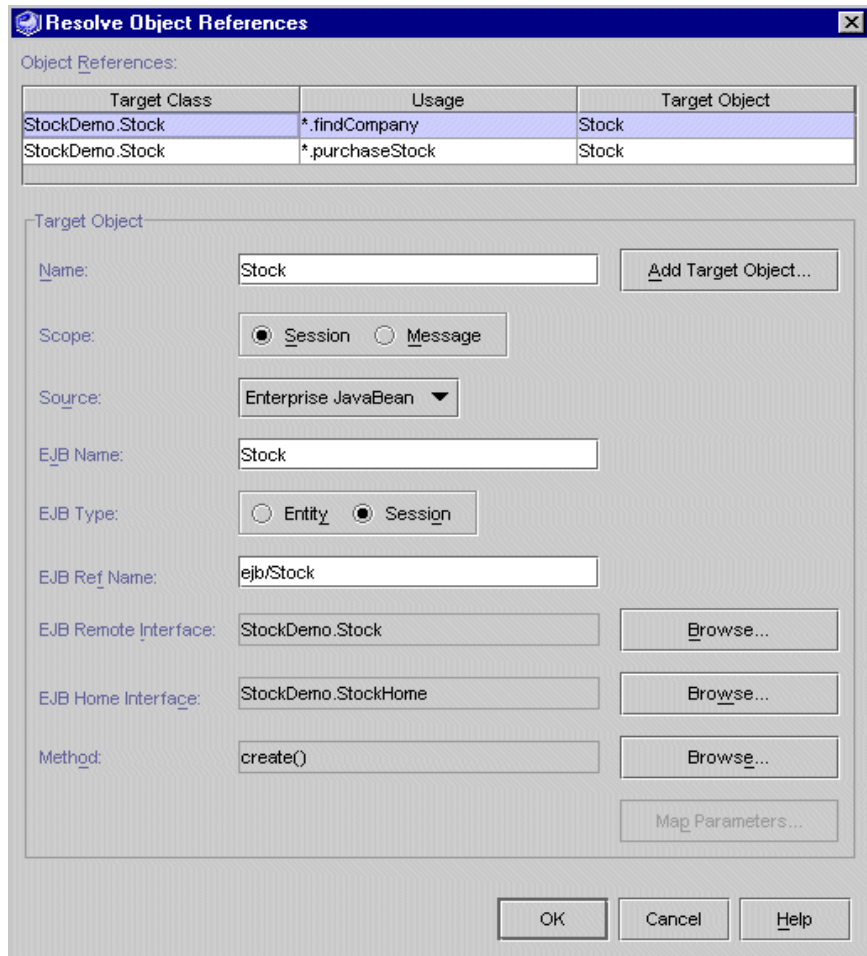


FIGURE 5-17 Resolve Object References Dialog Box

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 5-18.

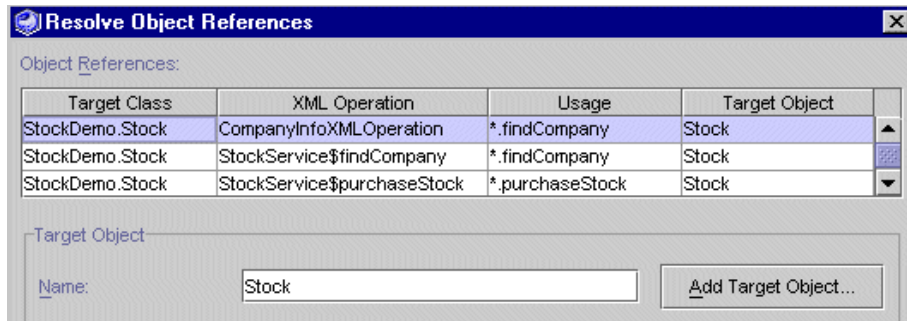


FIGURE 5-18 Resolve Object References Dialog Box With XML Operations

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	How the XML operation uses the target object. There are two possible uses. The XML operation can: <ul style="list-style-type: none"> • Call a method on the target object • Pass the target object as a parameter to a method call 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. 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> .
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 5-17).

- 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 183.

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 183.

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 5-17).

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 5, Standard Edition 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 5-1.

TABLE 5-1 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	Findmethod 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 source is a constructor, refer to TABLE 5-2.

TABLE 5-2 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 5-3.

TABLE 5-3 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.

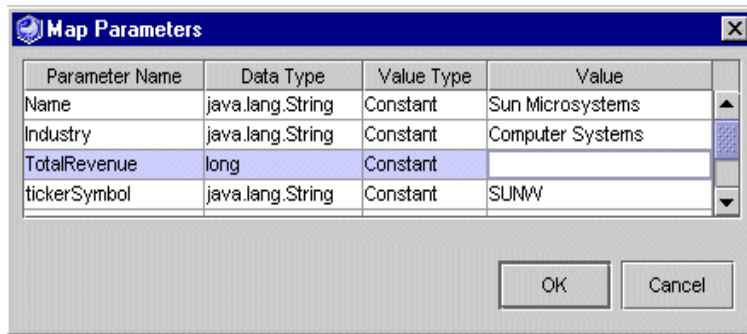


FIGURE 5-19 Map Parameters Dialog Box

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 47.
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, <code>String</code>); it cannot be a primitive (for example, it cannot be <code>int</code>). For information on declaring the data type of an input document element, see “Adding an Input Document Element” on page 168.

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 47.
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 159.

Securing a Web Service

This appendix explains how to secure a web service. The explanation assumes that you have general familiarity with the 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.

The procedures in this appendix assume that you have correctly installed and started the Sun™ ONE Application Server 7 server, and made it the default application server for your IDE. If you are using a different supported application server, modify the instructions accordingly.

Security Overview

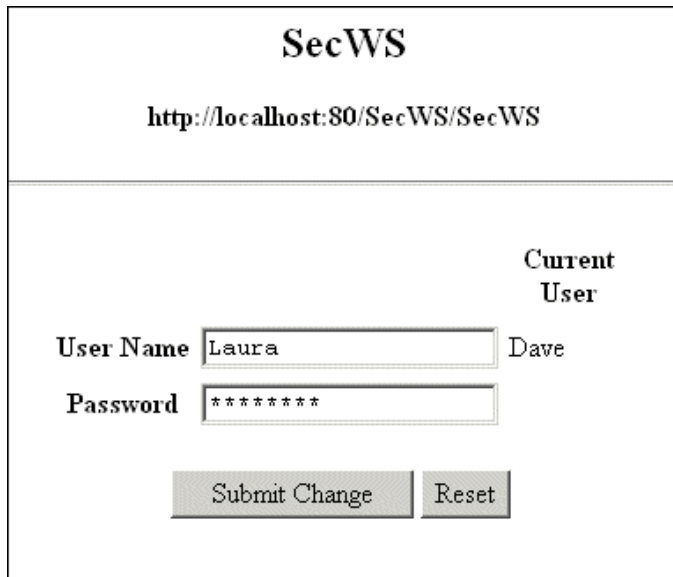
JAX-RPC and the IDE support HTTP Basic Authentication and HTTPS/SSL Authentication and Encryption. This appendix provides an overview of the security technologies and describes how to use the IDE to implement the two kinds of security.

Note – *SSL*, an acronym for *Secure Sockets Layer*, was originally developed for the Netscape web browser, and is now a de facto industry standard.

HTTP Basic Authentication

In basic authentication, the web server authenticates a client user (also called a *principal*) using the user name and password obtained from the web client, as illustrated in FIGURE A-1. Passwords are not protected against interception over the

network. To overcome this limitation, you can run basic authentication in an SSL-protected session, described later in this chapter, which ensures that all message content is encrypted.



The image shows a web browser window with the title "SecWS" and the address bar displaying "http://localhost:80/SecWS/SecWS". The main content area contains a login form. On the right side, there is a label "Current User" above the text "Dave". On the left side, there are two input fields: "User Name" containing "Laura" and "Password" containing seven asterisks. Below these fields are two buttons: "Submit Change" and "Reset".

FIGURE A-1 HTTP Basic Authentication Login Page

To implement HTTP basic authentication, you carry out these tasks:

- Set the web service's Authentication property in the IDE
- Associate roles with the web service in the IDE
- Define users (principals) and user groups in the application server or web server
- Map users or groups to the roles in the application server or web server

The procedures for carrying out these tasks are described in "Using HTTP Basic Authentication" on page 194.

This security mechanism applies to the entire web service and all its methods.

Note – The IDE uses a variation of HTTP basic authentication in which the generated client creates the login screen, and the user password is passed to the client proxy, which in turn passes it to the web service's application server.

HTTPS/SSL Authentication and Encryption

Secure Sockets Layer (SSL) provides data encryption, server authentication, message integrity, and optional client authentication. SSL and the associated technologies solve two fundamental problems of security for networked applications:

- Passwords, keys and data can be intercepted.
- The identity of a sender or recipient can be faked.

Public Key Encryption

SSL uses the technology of *public key encryption*. In this technology, a server or client is in possession of a two-part key: a public key and a private key. The private key is kept secret, while the public key is made known to all interested parties. The two parts of a key are related mathematically and have the following properties:

- A message encrypted with the public key can be decrypted with the private key.
- A message encrypted with the private key can be decrypted with the public key.
- It is computationally impractical to deduce the private key from the public key.

The mathematics of public key encryption are beyond the scope of this book, but the information is readily available on the internet.

If I encrypt my message with your public key and send it to you over a network, only you can decrypt the message. Third parties who might intercept my message over the network can't read it. They also can't fool you by changing my message in mid-stream, because the changed message will be garbled when you decrypt it. In this way, public key encryption supports both data encryption and message integrity.

Public key encryption supports the use of digital signatures. If I encrypt a message with my private key, I have effectively signed it, because any recipient with my public key can verify that I'm the only one who could have done the encryption. The implementation of digital signatures is more complex, but the specifics are beyond the scope of this book.

Although public keys can be distributed over a network, there still must be a way for you to know that a public key advertised in my name is really mine. Another person could distribute a public key claiming it is mine. You need a way to verify that claim. This can be done with *public key certificates* issued by trusted *certificate authorities*.

Public Key Certificates

A server and client authenticate each other by exchanging *public key certificates*. A public key certificate is the digital equivalent of an identity card. It is issued by a trusted organization, called a certificate authority (CA).

A certificate specifies the name of the owner and attests that the public key, included in the certificate, belongs to that owner. A certificate also includes an expiration date, the name of the CA that issued the certificate, and the digital signature of the issuing CA. SSL technology includes software that a client or server can use to validate a certificate and its contents based on the digital signature.

An SSL-enabled client or server has a database (or simple file) called a *trust store*, maintained by an administrator and containing trusted certificates. At runtime the client or server accepts an incoming certificate if it is signed by one of the owners of a certificate in the trust store. SSL uses the public key of the trusted certificate to decrypt the digital signature of the incoming certificate.

Certificates conform to the *X.509 Public Key Infrastructure (PKI)* standard. You can find further information about PKI standards and technologies at the following web sites:

Internet Engineering Task Force (IETF)—

<http://www.ietf.org/html.charters/pkix-charter.html>

Sun Microsystems:—

<http://docs.sun.com/source/816-6156-10/index.html>

Some commercial organizations issue certificates for a fee, and serve as certificate authorities. A commercial CA might offer several categories of certificates. The CA does more background checks on the owner of a more expensive certificate.

A company or public institution can act as a CA for its own applications, using certificate server software to issue certificates. In this case the certificates might not be trusted outside the domain of the issuing company or its business partners.

The following Sun web site contains documentation and download information for Sun Microsystem's certificate server software:

Sun™ ONE Certificate Server 4.7—

<http://docs.sun.com/db/prod/sunone#hic>

The following document explains how to manage certificates in the Sun ONE Application Server:

Administrator's Guide to Security: Sun™ ONE Application Server, Version 7—

<http://docs.sun.com/db/doc/816-7158-10>

SSL Handshaking

Consider what happens when your browser accesses a web site secured with SSL. If the target URL begins with HTTPS instead of HTTP, the browser attempts to connect to the server at a secure port. In general, when a client initiates an SSL session with a server, the connection is made at a port associated with an SSL listener.

Suppose that the server has a valid certificate and the browser does not have a certificate. This is a common scenario when consumers use their browsers to access a secure server and make purchases over the internet. In simple terms, the client authenticates the server. If the client also has a certificate, the server can authenticate the client. This scenario is known as *SSL mutual authentication*.

Assuming that the server is configured for SSL, a *handshaking* process begins. The following steps describe a simplified overview of the handshaking process.

1. The client sends an initial packet to the server containing information such as encryption algorithms that it can use.
2. The server sends a packet to the client containing information such as encryption algorithms that it can use and a copy of its certificate.
3. The client validates the server certificate.

The validation process uses the client's trust store. See "Public Key Certificates" on page 192 for more information.

4. The client generates a random session key and encrypts the session key with the server's public key, obtained from the server's certificate.

The session key is temporary for the duration of the SSL connection. The session key is processed by traditional, two-way encryption-decryption algorithms, and must be known to both parties in the exchange.

5. The client sends the encrypted session key to the server.

Only the server can decrypt the session key, using the server's private key. Now the client and the server both have the session key. They use this key to encrypt and decrypt data for the remainder of the session.

This technique solves two problems:

- The client in this scenario does not own a certificate. Therefore public key encryption by itself cannot be used by the two parties to send data in both directions. The session key is also needed.
- Public key encryption is computationally intensive. It is much more efficient for SSL to apply public key encryption only during the initial handshaking, for verifying a certificate's digital signature and for safe transmission of the generated session key to the server.

Using HTTP Basic Authentication

This explanation assumes that you have created and generated a web service in the IDE. The following tasks are required to secure your web service:

- Set the web service's Authentication property to HTTP Basic Authentication
- Assign authentication role names to the web service
- Define users (principals) and user groups in the application server
- Map the role names to users and groups in the application server
- Create and generate a client to access the secured web service

When you execute the application, the browser welcome screen will provide a link to a web page on which you can enter a user name and password.

Mapping Roles to Users and Groups

To assign HTTP basic authentication roles to your web service:

- 1. Right click the web service node and choose Properties.**
- 2. Set the Authentication property to HTTP Basic Authentication.**
The Authentication dialog box appears, as illustrated in FIGURE A-2.
- 3. Select the radio button for HTTP Basic Authentication.**
The Add and Remove buttons become active.

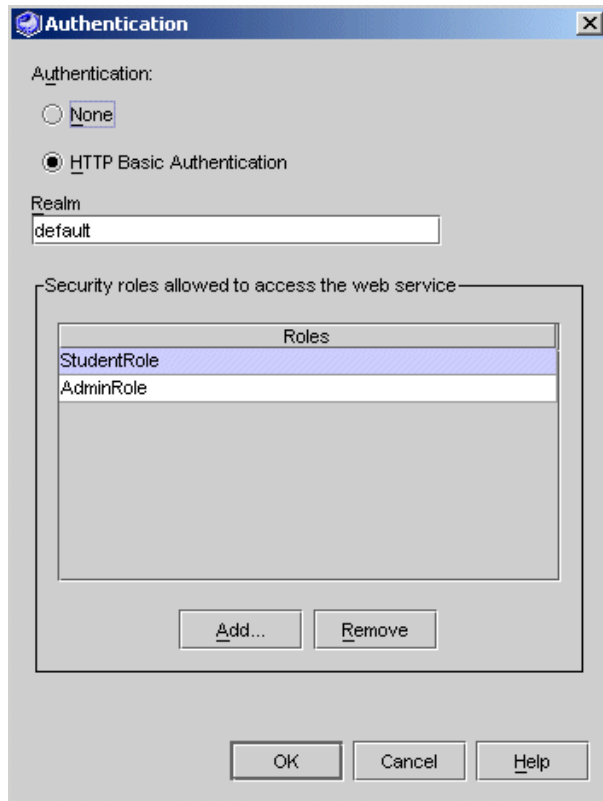


FIGURE A-2 Authentication Dialog Box

4. To add a role, click Add.

The Add Role dialog box is displayed, as illustrated in FIGURE A-3.

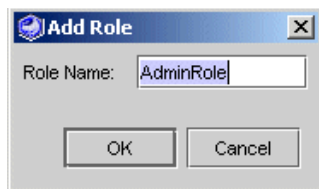


FIGURE A-3 Add Role Dialog Box

5. Enter a value for Role Name and click OK.

The new role name is displayed in the Authentication dialog box.

Map role names to group names and user names. You can do the mapping at the level of the web service or at the level of the J2EE application containing the web service.

Mapping Roles at the Level of the Web Service

To map role names to group names and user names at the level of the web service:

1. **Right click the web service node and choose Properties.**
2. **Click the Sun ONE AS tab.**

The Mapped Security Roles property shows the number of roles that you have assigned to your web service, as illustrated in FIGURE A-4.

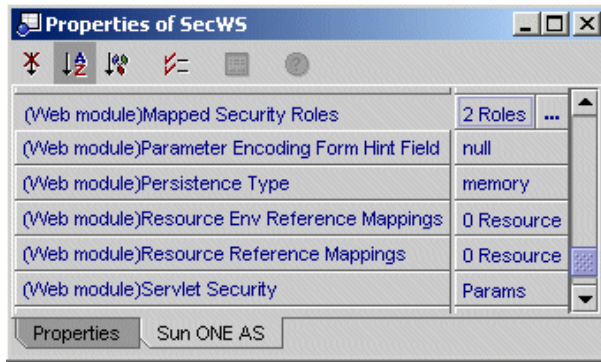


FIGURE A-4 Sun ONE AS Properties

3. **Click the Mapped Security Roles property.**
An ellipsis (...) button is displayed.
4. **Click the ellipsis button.**

The Mapped Security Roles dialog box appears, as illustrated in FIGURE A-5.

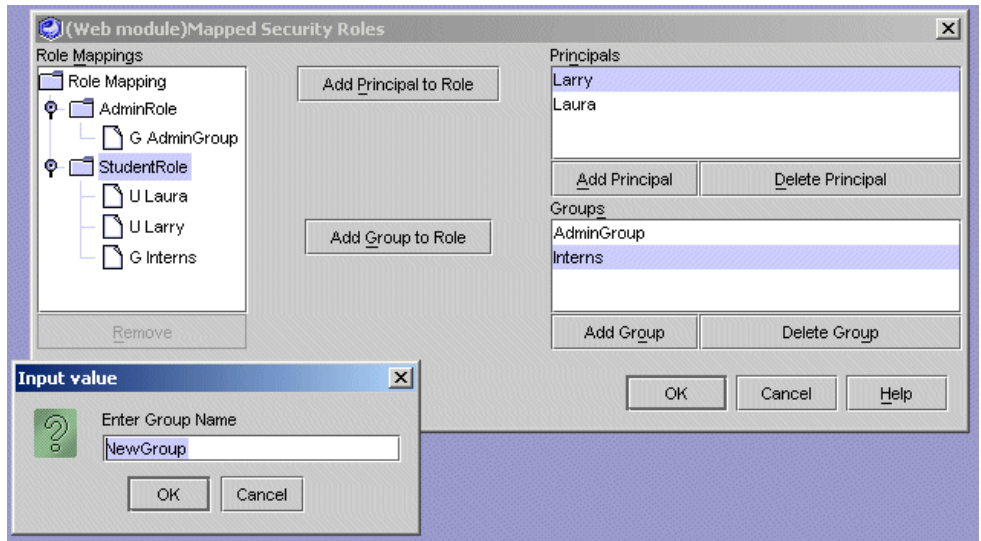


FIGURE A-5 Mapped Security Roles Dialog Box

The Mapped Security Roles dialog box displays principals, groups, roles, and their mappings. You can:

- Add and delete principals (users)
- Add and delete groups
- Add mappings of principals and groups to roles
- Remove mappings of principals and groups to roles

a. To add a principal or group, click Add Principal or Add Group.

The Input Value dialog box appears, as illustrated in FIGURE A-5.

i. Enter a value for the name.

ii. Click OK.

b. To map a principal or a group to a role:

i. Select a principal or a group.

Use ctrl-click to select more than one principal or more than one group.

ii. Select a role.

The Add Principal to Role button or the Add Group to Role button becomes active, depending on whether you selected principals or roles.

iii. Click Add Principal to Role or Add Group to Role.

The selected principals or groups are mapped and displayed under the selected role.

Mapping Roles at the Level of the J2EE Application

Suppose that your web service is deployed in a J2EE application. Suppose also that the web service's Authentication property is `HTTP Basic Authentication` and you have assigned the desired roles to the web service. You can map role names to group names and user names at the level of the J2EE application.

To do this:

1. **Create a J2EE application for the web service** (see “Assembling the J2EE Application” on page 55).
2. **Right click the application node and choose Properties.**
3. **Click the Sun ONE AS tab.**

The Mapped Security Roles property shows the number of roles that you have assigned to your web service.

4. **Click the Mapped Security Roles property.**

An ellipsis (...) button is displayed.

5. **Click the ellipsis button.**

The Mapped Security Roles dialog box appears, as illustrated in FIGURE A-5.

The remaining steps are the same as previously described (see “Mapping Roles at the Level of the Web Service” on page 196).

Use Case for Multiple Roles

HTTP basic authentication applies to a web service as a whole. Technically, the authentication applies to the JAXRPC servlet that implements the web service. All the roles of a given web service have the same purpose: to allow or deny access to the web service and its methods.

You might ask, therefore, what is the use of having more than one role? You could assign a single role to a web service and map all the authorized users and user groups to that role.

Consider an environment with several web services running in a single production application server and accessed by people working in a complex organizational structure. The mapping of users and user groups to web services for security purposes can be quite elaborate. The ability to assign multiple role names to a web service and, conversely, to assign a given role name to multiple web services can simplify the setup and maintenance of complex security relationships.

Another, forward-looking reason for supporting multiple roles is that method-level security is one of the directions for the evolution of web services technology. The ability to map users to different roles within a web service is one of the building blocks of method-level security.

Mapping Roles: Order of Priority

If you map roles to principals and groups at the level of the web service, and you also map roles at the level of the J2EE application, the mappings at the web service level take priority.

Mapping Roles In the Application Server

The mapping of role names to users and user groups is done by the IDE, and the mappings are stored in the application server. If you deploy your web service to another application server, you must redo the mapping using the administrative tools provided by the application server. In a production environment the mapping might be done by an application server administrator.

Note – The procedures described in this appendix use the Sun ONE Application Server that is bundled with your IDE.

Redirecting Role Names

Under certain circumstances, the role names associated with your web service might not work for your application server. For example, the role names might conflict with your shop's production naming conventions, or they might conflict with other role names already in use. If you have a web service purchased from a third party or shared by multiple applications, it might be inconvenient to change the role names directly.

To address this issue, the IDE provides a facility for redirecting role names at the level of the J2EE application. A role name defined to the J2EE application can be linked to a different role name to be used in the application server.

To redirect role names:

- 1. Right click the application node and choose Properties.**

The Properties tab displays the Security Roles property, showing the number of assigned roles.

2. Click the Security Roles property.

An ellipsis (...) button is displayed.

3. Click the ellipsis button.

The Security Roles dialog box appears, as illustrated in FIGURE A-6.

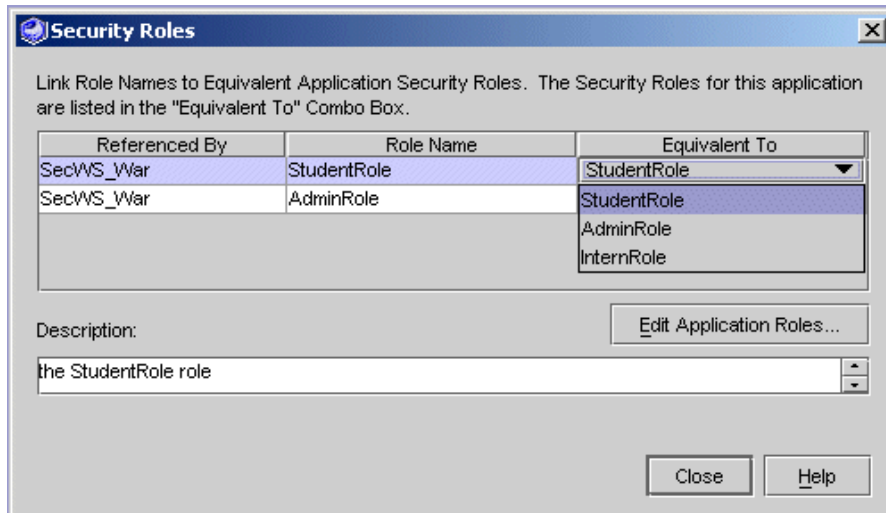


FIGURE A-6 Security Roles Dialog Box

The Security Roles dialog box displays War files, role names, and linked role names. The column headed “Equivalent To” shows the linked target role names, with default values equal to the role names.

To link a role name to a target role name:

a. Click the value in the Equivalent To column.

A list appears with the available target role names.

b. Select the desired target role name.

The new name appears in the Equivalent To column.

The Security Roles dialog box enables you to add or delete entries and their descriptions from the list of available target role names.

To add an entry:

1. Click Edit Application Roles

The Edit Application Roles dialog box appears, as illustrated in FIGURE A-7.

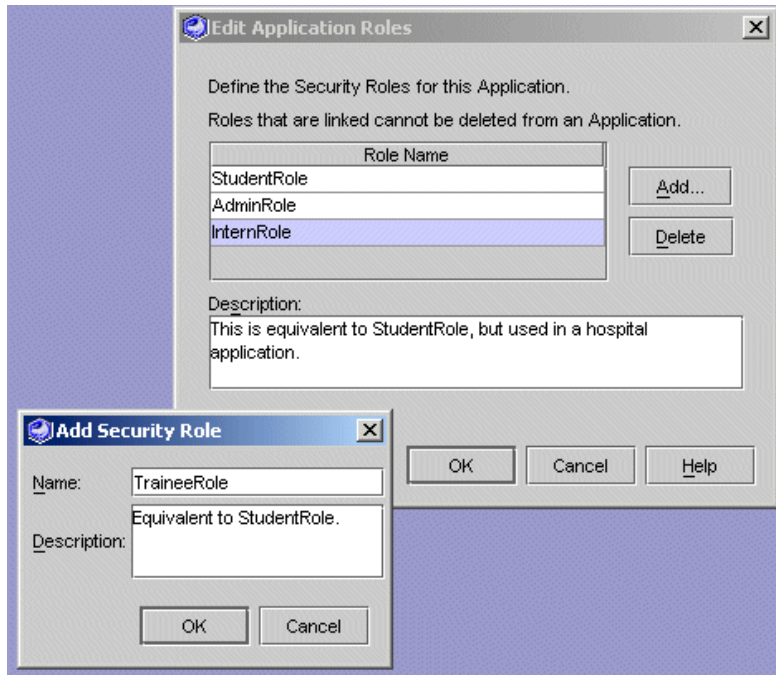


FIGURE A-7 Edit Application Roles Dialog Box

You can add a role name with a description. You can also delete a role name, if it is not currently linked.

2. Click Add.

The Add Security Role dialog box appears.

3. Enter a name and description, and click OK.

The new name is displayed in the list of available roles when you click a value in the Equivalent To column.

Testing Security: HTTP Basic Authentication

The explanation in this section assumes that you have created a client and made it the default test client for your web service. See “Testing a Web Service” on page 58 for the procedures to create and use a test client.

Note – When the web service’s Authentication property has the value HTTP Basic Authentication, the IDE creates a client whose HTTP Basic Authentication property has the value True. The generated client has a JSP login page in the Generated Documents folder.

In order to test HTTP basic authentication, you must define a test user in your application server, using the administrative tool provided by the server. See your application server’s documentation for the procedure. The following example illustrates how to get started if you are using Sun ONE Application Server 7.

To start the application server administration tool:

1. **Right-click your application server instance in the IDE’s runtime tab, and choose Launch Sun ONE Admin GUI.**

The IDE opens your default web browser and displays the administrative tool, as illustrated in FIGURE A-8.

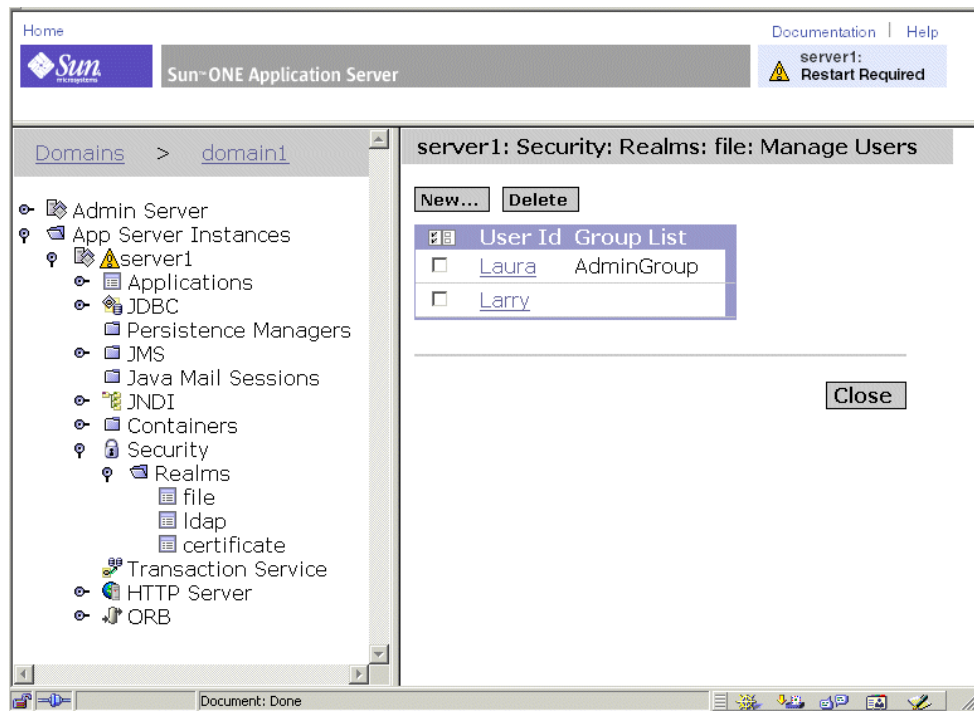


FIGURE A-8 Application Server Administrative Tool: Manage Users

2. Navigate to server1 : Security : Realms : file : Manage Users.

The Manage Users feature of the administrative tool enables you to add a new user with a password, associate a user with a group, and delete a user.

The remainder of this section assumes that you have defined a test user in your application server and mapped the user to a role associated with your web service, as described in “Mapping Roles to Users and Groups” on page 194.

To test the security of your web service:

1. Execute the application.

See “Testing a Web Service” on page 58 for the procedure to execute your application with the test client. There are two different procedures, depending on whether your web service has the multitier architecture or the web-centric architecture.

The test client displays the application’s welcome page in your browser. Near the top of the page is an authentication link: `Input user name and password for HTTP Basic Authentication`. The rest of the page contains buttons and input fields for using your web service’s business methods.

2. Click the authentication link.

The browser displays the login page, as illustrated in FIGURE A-1.

3. Enter an invalid user or password, and click Submit Change.

The browser returns you to the welcome page.

4. Enter any required data and click one of the application’s business method buttons.

Your browser displays an error message.

5. Click the authentication link.

The browser displays the login page.

6. Enter a valid user and password, and click Submit Change.

The browser returns you to the welcome page.

7. Enter any required data and click one of the application’s business method buttons.

The application performs as designed for an authorized user.

Basic Authentication and WSDL

The WSDL standard does not provide for authentication. Therefore, if you generate a client from a UDDI registry or from WSDL, the client's HTTP Basic Authentication property will be set to the default value, `False`. If you know that the web service supports HTTP basic authentication, you can manually set the client's property to `True`.

Basic Authentication and UDDI Publication

By default, when you publish a web service to a UDDI registry, the IDE specifies a WSDL URL that accesses the WSDL file through the web service. If your web service is protected by HTTP basic authentication, this might prevent a client developer from downloading the WSDL file to create a client. The IDE solves this problem as follows:

If you publish a web service whose Authentication property is `HTTP Basic Authentication`, the IDE packages an extra copy of the WSDL in the top directory of the WAR file. As the publisher of the web service, it is your responsibility during the publication process to manually provide a static WSDL URL that points to the copy of the WSDL file without calling the web service.

For the complete sequence of steps to publish a web service to a UDDI registry, see “Publication Procedure” on page 76. For the wizard page in which you must manually edit the WSDL URL, see FIGURE 2-31. The wizard page is titled, Publish Service and WSDL or Service only?

- **Change the WSDL File URL in the wizard as follows:**

From:

```
http://<hostname>:<portnumber>/<wsname>/<wsname>?WSDL
```

To:

```
http://<hostname>:<portnumber>/<wsname>/<wsname>.wsdl
```

Using HTTPS/SSL Authentication and Encryption

In a production environment, the management of SSL security and certificates involves an application server administrator and secure procedures. In a development environment, you might have to take on the role of the application

server administrator or consult with someone who has those skills. It might be helpful to have one person on a development team who handles the certificate management for the team.

The procedures in this section assume that you are working in a development environment and deploying a web service to the Sun ONE application server bundled with your IDE.

The term *trust store* in this section refers to a database (or a simple file) where certificates are kept. The location and format of a trust store, and the tools for managing a trust store, depend on your environment and application server.

Note – The terms *trust store* and *key store* are used interchangeably in this document. In common security terminology, *trust store* refers to a database where an entity keeps trusted certificates of other entities, and *key store* refers to a database where an entity keeps its own keys or certificates. However, the two databases can be the same and the IDE does not make the distinction. FIGURE A-10 shows the IDE option, SSL Trust Store, which serves both purposes.

The following tasks must be completed in order to configure your web service, test client, and application server for SSL authentication and encryption.

- Set the Use HTTPS property to `True` for your web service and test client.
- Create a client trust store to hold a copy of the server's certificate.
- Make the location of the client trust store known to your client.
- Export the server's certificate and import it into the client trust store.
- In the application server, create an SSL listener for each port used by one of your SSL-secured web services.

This enables the client to authenticate the server. In order for the authentication to succeed, the client trust store must contain a copy of the certificate owned by the server.

- (Optional) To enable client authentication:
 - Obtain a client certificate.
 - Import the client certificate into the client trust store.
 - Import the client certificate into the server trust store.
 - In the application server, activate client authentication for the port used by your web service.

This enables the server to authenticate the client. In order for the authentication to succeed, the server trust store must contain a copy of the certificate owned by the client.

Setting Properties for the Web Service and Test Client

To configure your web service for HTTPS/SSL authentication:

1. **Right click the web service node and choose Properties.**
2. **Set the Use HTTPS property to `True`.**

When you create a test client, the IDE sets its Use HTTPS property to match the corresponding property of the web service. For an existing test client:

1. **Right click the client node and choose Properties.**
2. **Set the Use HTTPS property to match the Use HTTPS property of the web service.**

Note – The WSDL standard does not provide for authentication. Therefore, if you generate a client from a UDDI registry or from WSDL, the client's Use HTTPS property will be set to the default value, `False`. If you know that the web service supports HTTPS/SSL authentication, you can manually set the property to `True`.

Setting Up the Test Client Trust Store

You must first create the trust store. This section describes how to create the trust store using the `keytool` utility that is provided with Java™ 2 Software Development Kit, Standard Edition, 1.4 or 1.4.1 (J2SE™ SDK).

Creating the Trust Store

Note – In a production environment, the trust store is typically set up by a security administrator in a secure location.

To create the trust store on a UNIX system:

- **Make the target directory current and enter the following command:**

```
$JAVA_HOME/bin/keytool -genkey -alias localhost -dname "CN=<client name>, OU=<organizational unit>, O=<organization>, L=<locality>, S=<state>, C=<country code>", -keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore
```

To create the trust store on a Windows system:

- **Make the target directory current and enter the following command:**

```
%JAVA_HOME%\bin\keytool -genkey -alias localhost -keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore
```

The keytool utility prompts you to enter the client's server name, organizational unit, organization, locality, state, and country code. You must enter the server name in response to the first prompt, which asks for first and last names.

FIGURE A-9 illustrates the creation of a trust store in a Windows environment.

```
C:\>cd temp
C:\temp>%JAVA_HOME%\bin\keytool -genkey -alias localhost -keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore
What is your first and last name?
  [Unknown]: SIAS?
What is the name of your organizational unit?
  [Unknown]: DevGroup\
What is the name of your organization?
  [Unknown]: Sun
What is the name of your City or Locality?
  [Unknown]: MPK
What is the name of your State or Province?
  [Unknown]: California
What is the two-letter country code for this unit?
  [Unknown]: US
Is CN=SIAS?, OU=DevGroup\, O=Sun, L=MPK, ST=California, C=US correct?
(in): y

C:\temp>dir *.keystore
Volume in drive C has no label.
Volume Serial Number is 6823-8F66

Directory of C:\temp

03/09/2003  04:04p                1,341 client.keystore
             1 File(s)                1,341 bytes
             0 Dir(s) 74,138,053,632 bytes free
```

FIGURE A-9 Create Client Trust Store

Setting the IDE's Global Trust Store Option

To make the location of the client trust store known to the IDE:

1. **Choose Tools → Options → Distributed Application Support → Web Service Settings from the IDE's main window.**

The Options dialog box is displayed, as illustrated in FIGURE A-10.

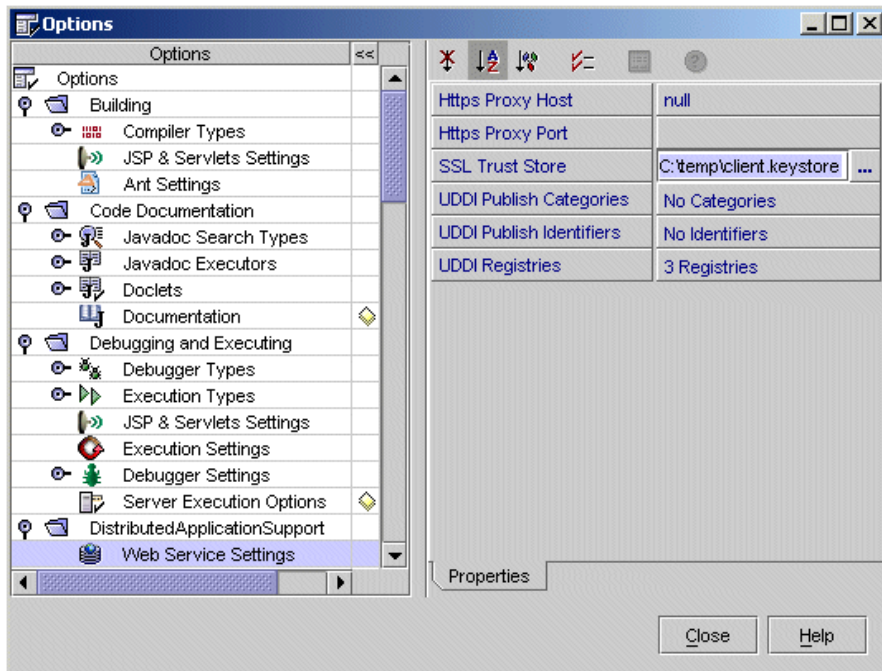


FIGURE A-10 Set SSL Trust Store Option

2. Click the **SSL Trust Store** option, then click the ellipsis (...) button.
The IDE opens a file chooser.
3. Select the trust store file and click **OK**.

Importing the Server's Certificate Into the Test Client Trust Store

The server administrator must first export the certificate to a file, using the server administration tool. If you are doing this yourself in a development environment, see your application server's documentation for the procedure. The following example, based on Sun ONE Application Server 7, is for illustration only.

To start the application server administration tool:

1. **Right-click your application server instance in the IDE's runtime tab, and choose Launch Sun ONE Admin GUI.**

The IDE opens your default web browser and displays the administrative tool.

2. **Navigate to server1 : Security, click the Certificate Management tab, and choose Manage from the menu.**

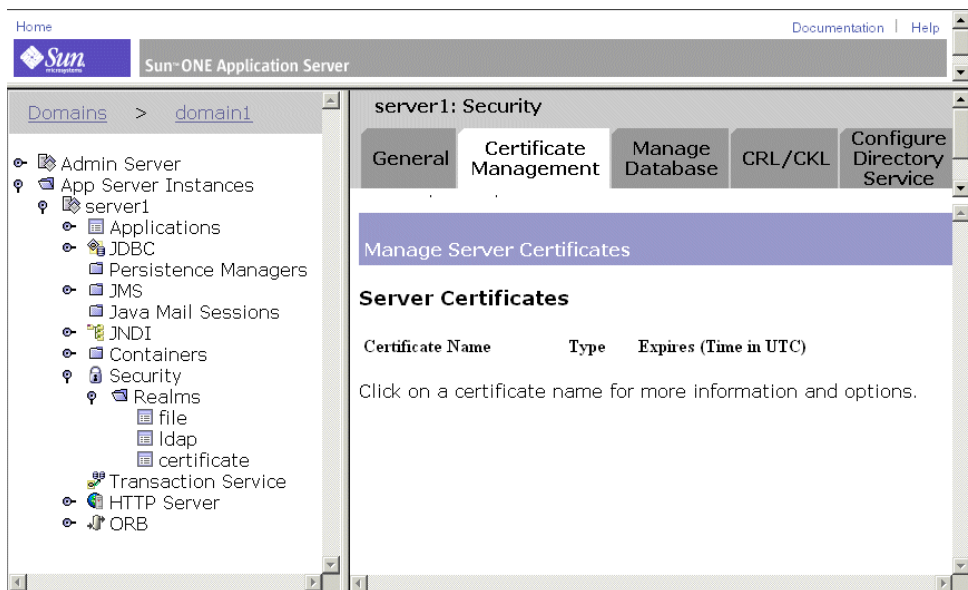


FIGURE A-11 Application Server Administrative Tool: Manage Certificates

3. Export the server certificate to a file.

You can now use the `keytool` utility to import the server certificate into your client trust store. The `keytool` command is the same for UNIX and Windows, except for path naming conventions. The following sample command assumes that the server certificate has been exported to a file, and that `server.cer` is the name of the file.

To import the server certificate into the client trust store:

- **Enter the following command:**

```
$JAVA_HOME/bin/keytool -import -v trustcacerts -alias localhost -file
server.cer -keystore client.keystore -keypass changeit -storepass
changeit
```

A client trust store can contain more than one trusted certificate.

To list the contents of a client trust store:

- **Enter the following command:**

```
$JAVA_HOME/bin/keytool -list -keystore client.keystore
```

The `keytool` utility prompts you to enter your trust store's password.

FIGURE A-12 illustrates the listing of a trust store in a Windows environment.

```
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\SYSTEM32>%JAVA_HOME%\bin\keytool -list -keystore c:\temp\client.keystore
Enter keystore password: changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

localhost, Mar 9, 2003, keyEntry,
Certificate fingerprint (MD5): E7:D6:A7:E2:7C:DC:E5:55:B3:73:8B:E5:9F:1E:97:DC

C:\WINNT\SYSTEM32>
```

FIGURE A-12 List Client Trust Store

Note – In a team of several developers, it might be helpful for one person to carry out the specialized task of certificate management on behalf of the entire team. This person can set up a single client trust store for the team and import all the required certificates.

Setting up the Server

The server administrator must create an SSL listener for the port used by your web service. If you have to do this yourself in a development environment, see your application server's documentation for the procedure. The following example, based on Sun ONE Application Server 7, is for illustration only.

To start the application server administration tool:

1. **Right-click your application server instance in the IDE's runtime tab, and choose Launch Sun ONE Admin GUI.**

The IDE opens your default web browser and displays the administrative tool.

2. **Navigate to server1 : HTTP Server : HTTP Listeners.**

This page, illustrated in FIGURE A-13, lists the existing HTTP listeners and enables you to delete a listener or add a new listener.

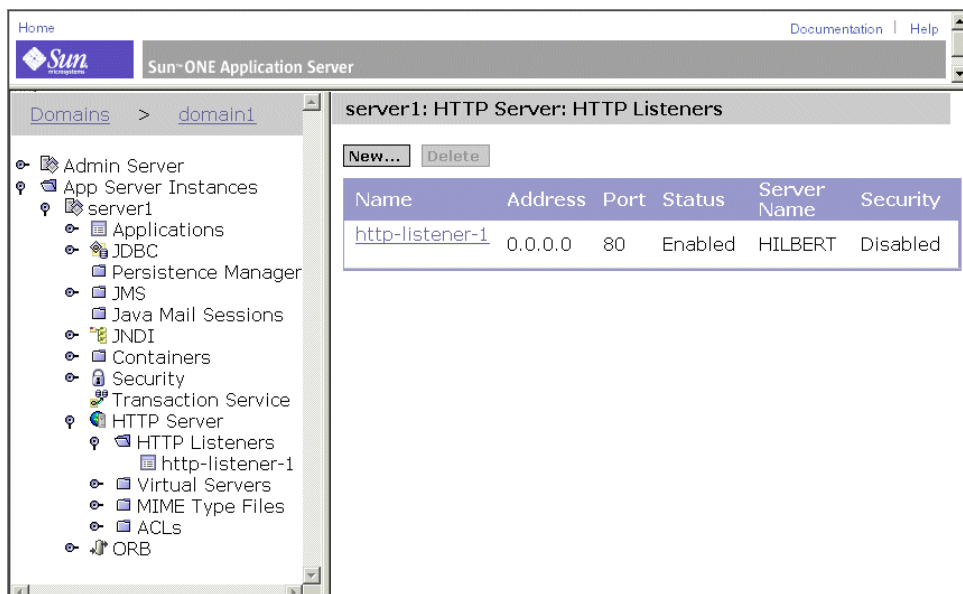


FIGURE A-13 Application Server Administrative Tool: HTTP Listeners

3. Navigate to an existing HTTP listener to display or change its properties, as illustrated in FIGURE A-14.

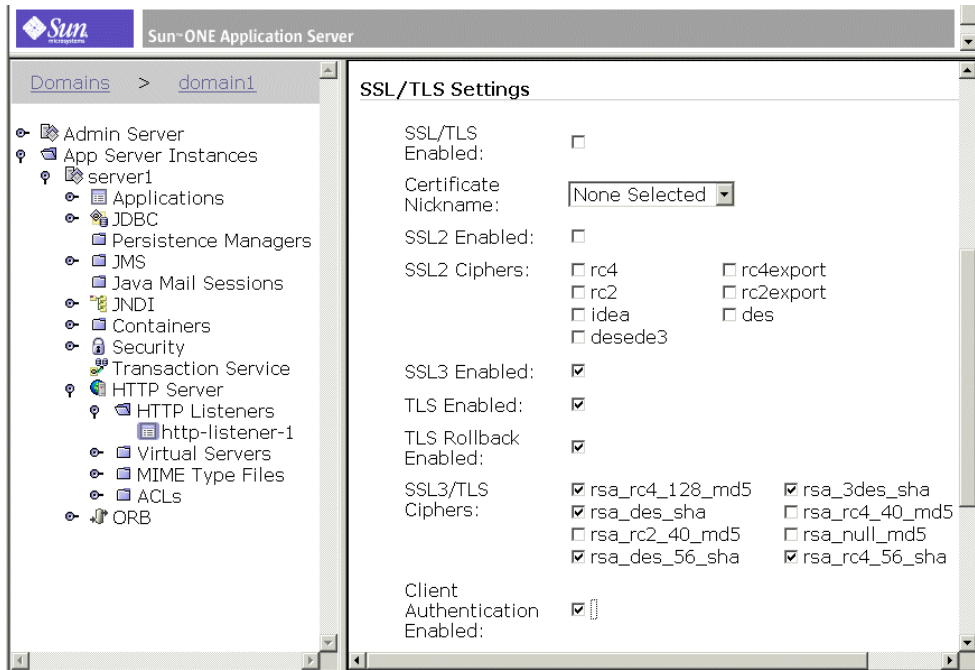


FIGURE A-14 Application Server Administrative Tool: HTTP Listener Properties

Note – To set up the server for mutual (two-way) authentication, check the Client Authentication Enabled checkbox for the listener associated with the port used by your web service.

Testing Security: HTTPS/SSL Authentication and Encryption

To test the security of your web service with SSL server authentication:

1. Execute the application.

See “Testing a Web Service” on page 58 for the procedure to execute your application with the test client. There are two different procedures, depending on whether your web service has the multitier architecture or the web-centric architecture.

The test client displays the application’s welcome page in your browser. Near the top of the page is a link: [Input trust store and password for SSL Certificates](#), as illustrated in FIGURE A-15. The rest of the page contains buttons and input fields for using your web service’s business methods.



FIGURE A-15 Client Welcome Page: Security Links

2. Click the SSL Certificates link.

The browser displays a page in which you can enter the location of the client's trust store, as illustrated in FIGURE A-16.

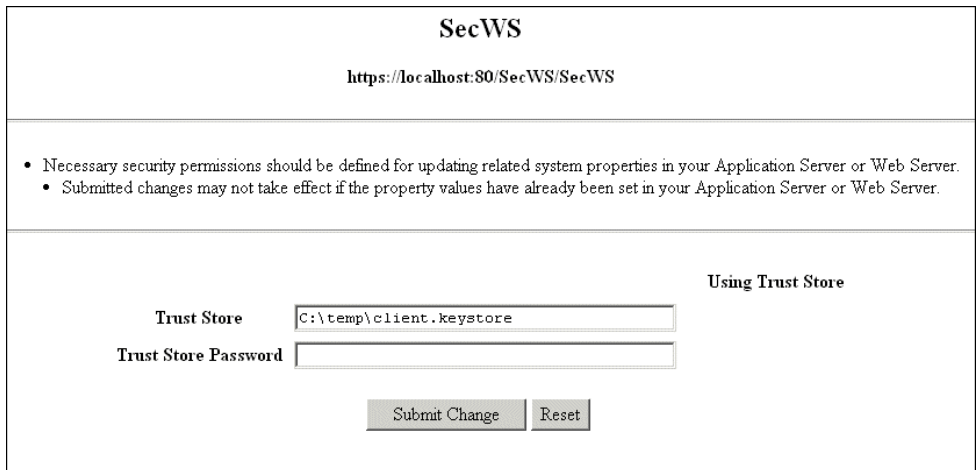


FIGURE A-16 Client Page: Set Trust Store and Trust Store Password

A client generated by the IDE provides this page as a convenience for testing in a development environment. The trust store location that you set in this page is used by the server in which the client is running. In a production environment, the trust store location is typically set by a security administrator, not by a client.

Note – You can set the trust store location only once for a server session. To change the trust store location, you must first stop and start the server. The reason for this is that the server design is based on the assumption that in a production environment the trust store location is set by the security administrator, not by clients.

3. Enter the client's trust store location and password, and click Submit Change.

The browser returns you to the welcome page.

4. Enter any required data and click one of the application's business method buttons.

The application performs as designed for an authorized user.

Note – If the server is also configured for client authentication, the client must own a certificate and you must import the client certificate into the server's trust store, using the server's administration tools.

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 your web service's deployment descriptors in the Source Editor at any time during development. You can also edit the deployment descriptors. This appendix describes the procedures for viewing and editing deployment descriptors.

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>

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 B-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.

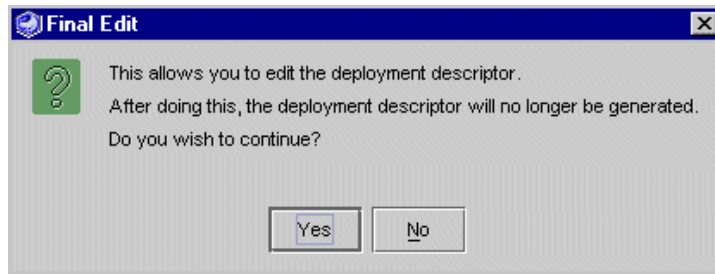


FIGURE B-1 Deployment Descriptor Final Edit Dialog Box

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, 168
- adding method references to an XML operation, 166
- Appendix A, 19
- Application Server property, 56
- architecture
 - multitier, 35, 37, 38, 43, 52, 58, 60, 203, 212
 - web-centric, 37, 38, 43, 52, 53, 58, 60, 123, 124, 203, 212
- arrays, 86
- assembling a web service, 53
- attachments, 123
- Authentication property, 79, 190, 194, 198, 202, 204

C

- casting a method return value, 175
- classes returned by XML operations
 - collapsing, 177
 - expanding, 177
- classes returned by XML operations (deprecated)
 - expanding, 161
- client properties
 - Conversational, 65, 66, 102
 - Generate Presentation, 101, 113
 - HTTP Basic Authentication, 202, 204
 - SOAP Message Handlers, 135, 139, 141, 144, 146
 - Soap Runtime, 93, 94, 95, 110, 112
 - Source, 103
 - Use Data Handler Only, 128

- Use HTTPS, 206

clients

- attachments, 123
- creating from a UDDI registry, 113
- creating from a web service, 91
- creating to test a web service, 58
- HTML error page, 97
- HTML welcome page, 97
- JAX-RPC, 95
- JSP pages, 97
- kSOAP, 110
- MIDlets, 110
- refresh from UDDI, 102
- refresh from web service, 102
- refresh from WSDL, 102
- stateful, 34, 59, 129
- wireless, 110

collections

- serialization classes, 87
- user-defined object types, 87

Context Root property, 54

Conversation Initiator property, 64

Conversation Terminator property, 65

Conversational property, 64, 65, 66, 102

D

data

- retrieving more or less, 161
- returned by XML operation, 160
- returning less to client, 161

deleting references, 46, 101

- deploying a web service, 53
- deployment descriptors
 - editing, 89, 215
 - generated for web service, 89
 - IDE fields propagated to, 215
 - viewing, 89, 215
- development work flow
 - web service clients, 91
 - web services, 41, 132
 - XML operations, 162

E

- elements
 - adding to XML input document, 168
 - deleting from input document, 172
 - excluding from XML output document, 161, 175
 - including in XML output document, 176
- environment entries, 47
- Environment Entries property, 47
- Environment Entry property, 216
- error pages, generating, 97
- example applications, 22
- execution of XML operations, 153
- External Files property, 46

G

- Generate Presentation property, 101, 113
- generating runtime classes, 52

H

- handlers, 131
- HTTP basic authentication, 34, 189, 194
- HTTP Basic Authentication property, 202, 204
- HTTPS/SSL authentication and encryption, 34, 189, 204

I

- inherited methods, 175
- input document elements

- adding, 168
- deleting, 172
- reordering, 172

- Input Document Elements node, 159
- instantiating objects at runtime, 180

J

- Javadoc documentation in the IDE, 22
- JAX-RPC
 - client proxy, 93, 99
 - client runtime property, 93
 - version, 39
- JSP custom tags, 95
- JSP pages, generating, 97
- JSP tag library, 95

K

- kSOAP
 - client front end, 94, 110
 - client proxy, 93
 - client runtime property, 93
 - version, 39
- kSOAP clients, 110

M

- Mapped Security Roles property, 196, 198
- mapping parameters, 172
- message handlers, *See* handlers
- method calls in XML operations
 - adding, 166
 - deleting, 161, 172
 - description, 160
 - order of execution, 160
 - reordering, 172
- Methods node, 159
- MIDlets, 110
- multitier architecture, 35, 37, 38, 43, 52, 58, 60, 203, 212

O

- object references
 - resolving, 180
 - specifying target object, 181
- objects
 - instantiating at runtime, 180
 - scope of references, 184
- operations, *See* XML operations

P

- parameters
 - mapping, 160
 - mapping in web service, 186
 - mapping in XML operation, 160, 172, 186
 - source type, 160
- Parameters node, 160

R

- references
 - adding to web service, 45
 - deleting from web service, 46
 - resolving, 180
- refresh from UDDI, 102
- refresh from web service, 102
- refresh from WSDL, 102
- request-response mechanism, 153
- Resolve Object References dialog box, 181, 183
- resolving object references, 180
- runtime classes, generating, 52

S

- scope of object references, 184
- security
 - HTTP basic authentication, 34, 189, 194
 - HTTPS/SSL authentication and encryption, 34, 189, 204
 - overview, 189
- Security Roles property, 199
- Serialization Classes property, 87
- SOAP
 - client front end, 94, 110

- client proxy, 99
- client runtime property, 93
- description, 27, 28
- IDE features, 34
- JAX-RPC implementation, 39
- kSOAP implementation, 39
- specification, 39

- SOAP Message Handlers property, 135, 139, 141, 144, 146
- SOAP message handlers, *See* handlers
- SOAP RPC URL property, 43, 54, 55, 56, 59, 75, 76
- Soap Runtime property, 93, 94, 95, 110, 112
- Source property, 103
- starting and stopping the internal UDDI registry server, 82
- stateful web services and clients, 34, 59, 129

T

- tag library, 95
- target objects
 - defining new, 183
 - editing definitions, 183
 - instantiating objects, 180
 - resolving references, 180
 - specifying an object reference, 181
- Test Client property, 58

U

- UDDI
 - creating a client from a registry, 113
 - editing the table of registries, 73
 - internal registry overview, 81
 - internal registry sample browser, 82
 - internal registry startup and shutdown, 82
 - managing default categories and identifiers, 70
 - overview of IDE features, 34
 - overview of industry standards, 28, 29
 - overview of registry usage, 31
 - publishing a web service to a registry, 31, 75
 - specification, 39
- Use Data Handler Only property, 128
- Use HTTPS property, 205, 206
- utility methods, static, 178

W

- Web Context property, 55, 56
- web service properties
 - Application Server, 56
 - Authentication, 79, 190, 194, 198, 202, 204
 - Context Root, 54
 - Conversation Initiator, 64
 - Conversation Terminator, 65
 - Conversational, 64, 65, 102
 - Environment Entries, 47
 - Environment Entry, 216
 - External Files, 46
 - Mapped Security Roles, 196, 198
 - Security Roles, 199
 - Serialization Classes, 87
 - SOAP Message Handlers, 135, 139, 141, 144, 146
 - SOAP RPC URL, 43, 54, 55, 56, 59, 75, 76
 - Test Client, 58
 - Use HTTPS, 205, 206
 - Web Context, 55, 56
- web services
 - adding operations, 45
 - architectures, 35
 - assembling, 53
 - bottom-up development, 35, 41
 - creating, 42, 49
 - creating clients for testing, 58
 - creating clients from local local service, 91
 - creating clients from UDDI registry, 113
 - creating clients from WSDL, 112
 - deleting references, 46
 - deploying, 53
 - description, 25
 - development work flow, 41, 132
 - endpoint URL, 103
 - generating runtime classes, 52
 - multitier, 35
 - scope of object references, 184
 - stateful, 34, 59, 129
 - test clients, 58
 - top-down development, 35, 41
 - web-centric, 35
- web-centric architecture, 37, 38, 43, 52, 53, 58, 60, 123, 124, 203, 212
- welcome page, generating, 97
- wireless clients, 110
- WSDL
 - creating clients, 112

- creating web services, 35, 41, 48
- description, 28, 29
- generating from web service, 69
- IDE features, 34
- specification, 39

X

- XML operations (deprecated)
 - adding input document elements, 168
 - adding method calls, 166
 - collapsing classes, 161
 - creating, 163
 - Data Source pane, 158
 - deleting method calls, 161, 172
 - description, 153
 - development work flow, 162
 - editing, 166
 - excluding elements from XML output
 - document, 161
 - expanding classes, 161
 - including elements in output document, 176
 - mapping parameters, 160, 172
 - reordering input document elements, 172
 - reordering methods, 172
 - request-response mechanism, 153
 - specifying return data, 160
 - XML Input Format pane, 158
 - XML Output Format pane, 158
- XML output documents
 - excluding elements, 175
 - including elements, 176