Interstage Application Server
V6.0
SOAP Service User’s Guide
## Trademarks

Trademarks of other companies are used in this user guide only to identify particular products or systems:

<table>
<thead>
<tr>
<th>Product</th>
<th>Trademark/Registered Trademark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft, Visual Basic, Visual C++, Windows, Windows NT, Internet Information Server, and Internet Explorer</td>
<td>Registered trademarks of Microsoft Corporation in the U.S.A. and other countries</td>
</tr>
<tr>
<td>Sun, Solaris, Java, and other trademarks containing Java</td>
<td>Trademarks of Sun Microsystems, Inc., in the U.S.A. and other countries</td>
</tr>
<tr>
<td>UNIX</td>
<td>Registered trademark in the U.S.A. and other countries, licensed exclusively through X/Open Company Ltd.</td>
</tr>
<tr>
<td>Netscape, Netscape FastTrack Server, Netscape Enterprise Server, and Netscape Navigator</td>
<td>Registered trademarks of Netscape Communications Corporation in the U.S.A. and other countries</td>
</tr>
<tr>
<td>CORBA, Object Management Group, OMG, OMG IDL, IIOP, Object Request Broker, and ORB</td>
<td>Trademarks or registered trademarks of Object Management Group Inc. in the U.S.A. and other countries</td>
</tr>
<tr>
<td>Interstage and ObjectDirector</td>
<td>Registered trademarks of Fujitsu Limited</td>
</tr>
</tbody>
</table>

This document contains technology relating to strategic products controlled by export control laws of the producing and/or exporting countries. This document or a portion thereof should not be exported (or re-exported) without authorization from the appropriate government authorities in accordance with such laws.

Fujitsu Limited

The contents of this manual may be revised without prior notice.
All Rights Reserved, Copyright © FUJITSU LIMITED 2003
Preface

Purpose of this Document
This manual explains the procedures for:

- Programming necessary for development of a Web service
- Creating a CORBA/SOAP gateway for linkage between a Web service and component models such as CORBA and J2EE
- Operating a Web service.

Who Should Read this Document?
This manual is intended for users who create and operate SOAP Service applications. It is assumed that readers of this manual have a basic knowledge of:

- Java language
- The Internet
- XML
- SOAP (Simple Object Access Protocol)
- SOAP security extension and digital signatures (XML digital signatures and XML partial encryption)
- J2EE
- object-oriented technology

Windows
- Windows NT® and Windows® 2000

Solaris 0E
- UNIX

Linux
- Linux
Organization of this Document

This document is organized as follows:

- **Chapter 1 - Overview**
  This chapter provides an overview of Interstage SOAP Service and Web services.

- **Chapter 2 - Designing a Web Service**
  This chapter explains the design items that should be considered before creating a Web service.

- **Chapter 3 - Configuring an Environment**
  This chapter explains how to configure an environment necessary for using the functions provided by Interstage SOAP Service.

- **Chapter 4 - Installing Applications that use the Messaging Method**
  This chapter explains how to install applications that use the Messaging method.

- **Chapter 5 - Installing Applications that use the RPC Method**
  This chapter explains how to install applications that use the remote procedure call (RPC) method.

- **Chapter 6 - Installing a CORBA/SOAP Gateway**
  This chapter explains the procedure for installing a CORBA/SOAP gateway used for linkage with conventional components such as CORBA and J2EE. This chapter also provides notes on the linkage.

- **Chapter 7 – How to use the Reliable Messaging Function**
  This chapter explains how to install a Web service using the reliable messaging function.

- **Chapter 8 - Support of Data Types**
  This chapter explains XML and Java data type mapping.

- **Chapter 9 - Managing Web Service Information**
  This chapter explains how to manage Web service information.

- **Chapter 10 - Operating a Web Service**
  This chapter explains how to operate a Web service (operation procedures from starting the Web service to stopping the service).

- **Chapter 11 - Sample Programs**
  This chapter provides an overview of sample programs and explains how to use them.

- **Chapter 12 - Customizing the Web Service Container**
  This chapter explains how to customize the Web service container.

- **Appendix A - Describing Web Service Information**
  This appendix explains the conventions used to describe Web service information.
# Table of Contents

## Chapter 1 Overview
Overview of Functions ................................................................. 1-2

## Chapter 2 Designing a Web Service
Selecting Components that Constitute a Web............................... 2-2
  CORBA/SOAP Server Gateway .............................................. 2-2
  CORBA/SOAP Client Gateway ............................................. 2-3
Selecting SOAP Message Exchange Methods ............................. 2-5
  Messaging Method ............................................................. 2-5
  RPC Method ........................................................................ 2-6
Selecting Operations.................................................................. 2-7
  Oneway and RequestResponse Modes ................................. 2-7
Selecting the Reliable Messaging Function ............................... 2-7
  PUSH Model ....................................................................... 2-9
  PULL Model ....................................................................... 2-10
  Model that Combines both PUSH and PULL ...................... 2-11
Selecting the Web Service Security Function ............................ 2-12

## Chapter 3 Configuring an Environment
Defining the Server System Environment .................................. 3-2
Defining the Client System Environment .................................. 3-4
Defining the Environment for the High-reliability Web Service Function ........................................ 3-7
Configuring the CORBA/SOAP Gateway Environment ................ 3-8
  CORBA/SOAP Server Gateway .......................................... 3-8
  CORBA/SOAP Client Gateway .......................................... 3-9

## Chapter 4 Installing Applications that Use the Messaging Method
Creating a Receiver Application ................................................ 4-2
  Receiver Application that does not Reply (OneWay Mode) ............ 4-3
  Receiver Application that Replies (RequestResponse Mode) ....... 4-4
  Using Attachment Files ...................................................... 4-6
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returning “fault”</td>
<td>4-10</td>
</tr>
<tr>
<td>Handling of SOAP Header</td>
<td>4-12</td>
</tr>
<tr>
<td>Creating a Sender Application</td>
<td>4-13</td>
</tr>
<tr>
<td>Basic Sender Application</td>
<td>4-14</td>
</tr>
<tr>
<td>Using Attachment Files</td>
<td>4-17</td>
</tr>
<tr>
<td>Sender Application Using Existing XML Data</td>
<td>4-20</td>
</tr>
<tr>
<td>Adding HTTP Connection Information including Proxy, Authentication, and Session Information</td>
<td>4-23</td>
</tr>
<tr>
<td>SOAP Message Processing by SAAJ-API</td>
<td>4-27</td>
</tr>
<tr>
<td>Structure of SOAP Message</td>
<td>4-27</td>
</tr>
<tr>
<td>Creating a SOAP Message</td>
<td>4-28</td>
</tr>
<tr>
<td>SOAP Envelope Processing</td>
<td>4-28</td>
</tr>
<tr>
<td>Processing of Attachment Files</td>
<td>4-30</td>
</tr>
<tr>
<td>Fault Processing</td>
<td>4-32</td>
</tr>
<tr>
<td>Analyzing Fault Information</td>
<td>4-33</td>
</tr>
<tr>
<td>Chapter 5 Installing Applications that Use the RPC Method</td>
<td></td>
</tr>
<tr>
<td>Stub Method and DII Method</td>
<td>5-2</td>
</tr>
<tr>
<td>Handling Attachment Files</td>
<td>5-3</td>
</tr>
<tr>
<td>RPC Server Application</td>
<td>5-4</td>
</tr>
<tr>
<td>RPC Client Application Using the Stub Method</td>
<td>5-7</td>
</tr>
<tr>
<td>Basic RPC Client Application (Stub Method)</td>
<td>5-7</td>
</tr>
<tr>
<td>Adding HTTP Connection Information including Proxy and Authentication Information (Stub Method)</td>
<td>5-11</td>
</tr>
<tr>
<td>Using Session Control</td>
<td>5-14</td>
</tr>
<tr>
<td>RPC Client Application Using the DII Method</td>
<td>5-17</td>
</tr>
<tr>
<td>Basic RPC Client Application (DII Method)</td>
<td>5-17</td>
</tr>
<tr>
<td>RPC Client Application using User-defined Types (DII Method)</td>
<td>5-22</td>
</tr>
<tr>
<td>Adding HTTP Connection Information including Proxy and Authentication Information (DII Method)</td>
<td>5-25</td>
</tr>
<tr>
<td>Using Session Control</td>
<td>5-27</td>
</tr>
<tr>
<td>Defining and Analyzing Fault Information</td>
<td>5-31</td>
</tr>
<tr>
<td>Defining Fault Information</td>
<td>5-31</td>
</tr>
<tr>
<td>Analyzing Fault Information</td>
<td>5-33</td>
</tr>
<tr>
<td>Chapter 6 Installing a CORBA/SOAP Gateway</td>
<td></td>
</tr>
<tr>
<td>Generation of Java Source Programs for CORBA/SOAP Gateway</td>
<td>6-2</td>
</tr>
<tr>
<td>Generation of Java Class Files for CORBA/SOAP Gateway</td>
<td>6-4</td>
</tr>
<tr>
<td>Precautions on Linkage with CORBA Application</td>
<td>6-5</td>
</tr>
<tr>
<td>Association of Conversion Mapping for Conversion from Java Values to XML Values and Vice Versa (CORBA)</td>
<td>6-5</td>
</tr>
</tbody>
</table>
Obtaining CORBA Exception from Fault Details Item.........................................................6-6
RPC-based Client Application that Links with Factory Interface .......................................6-12
Linkage with CORBA Server Application that uses TypeCode Type..............................6-16
Linkage with CORBA Server Application that Uses any Type...........................................6-17
Other Precautions..............................................................................................................6-18

Precautions on Linkage with EJB Application..................................................................6-19
Provision of Parameters to be Sent to Method of EJB Server Application.......................6-19
Association of Conversion Mapping for Conversion from Java Values to XML Values and Vice
Versa (EJB).........................................................................................................................6-21
RPC-based Client Application that Links With STATEFUL SessionBean.......................6-22
Obtaining the Exception Class from Fault Details Item....................................................6-27

Chapter 7  How to use the Reliable Messaging Function

Agreement between Client System and Server System.......................................................7-2
Message Type ID................................................................................................................7-2
Sender ID, Receiver ID ......................................................................................................7-3
URL of Receiver Server/Sender Server.............................................................................7-4
SOAP Signature Verification ID .........................................................................................7-4

PUSH Model where Server System Receives Messages ..................................................7-5
Agreement in PUSH Model..............................................................................................7-5
Preparation of Receiver Application on PUSH Model Receiver Server..........................7-6
Preparation of Key Pair and Public Key.............................................................................7-6
Preparation of Receiver Application..................................................................................7-6
Allocation of Receiver Application...................................................................................7-7
Setting of the Reliable Messaging Function.....................................................................7-7
Preparation of Sender Application on PUSH Model Sender Client.................................7-7
Preparation of Key Information.........................................................................................7-7
Preparation of Sender Application.....................................................................................7-8
Setting of Reliable Messaging Function...........................................................................7-10
Execution of Sender Application.......................................................................................7-10
Structure of Application using PUSH Model ..................................................................7-11

PULL Model where Client System Receives Messages....................................................7-12
Agreement Necessary in PULL Model...............................................................................7-12
Preparation of Sender application on PULL Model Sender Server..................................7-13
Preparation of Key Information.........................................................................................7-13
Preparation of Sender Application....................................................................................7-13
Setting of Reliable Messaging Function...........................................................................7-13
Allocation of Sender Application.......................................................................................7-13
Preparation of Sender Application on PULL Model Receiver Client..............................7-14
Preparation of Key Information.........................................................................................7-14
<table>
<thead>
<tr>
<th>Chapter 8  Support of Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping of XML and Java Data Types</td>
</tr>
<tr>
<td>Simple Types</td>
</tr>
<tr>
<td>Enumeration Type</td>
</tr>
<tr>
<td>Structure Type</td>
</tr>
<tr>
<td>Bean Type</td>
</tr>
<tr>
<td>Array Type</td>
</tr>
<tr>
<td>java.util.Vector Type</td>
</tr>
<tr>
<td>Attachment Types</td>
</tr>
<tr>
<td>Multiple Reference</td>
</tr>
</tbody>
</table>

Correspondence of Conversion Mapping Required for Conversion Between Java and XML Values | 8-28

Mapping of XML and CORBA Data Types | 8-31
Basic Data Types                     | 8-31
Enumeration Type                     | 8-32
Structure and Exception Types        | 8-34
Array/Sequence Type                  | 8-36
CORBA Object/Interface Type          | 8-37
TypeCode Type                        | 8-38
Any Type                            | 8-38

Mapping of XML and EJB Data Types | 8-39
Primitive Types                     | 8-39
Class and Exception Types           | 8-40
Array Type                          | 8-41
## Chapter 9 Managing Web Service Information

- Management Web Service Information by the GUI ................................................................. 9-2
- Starting the Web Service Information Edit Tool ................................................................. 9-2
- Deployment in the Server System Environment ................................................................. 9-3
  - Registering Web Service Information ........................................................................... 9-3
  - Displaying the Web Service Identifier ........................................................................ 9-12
  - Displaying the Web Service Information .................................................................... 9-13
  - Updating Web Service Information ............................................................................. 9-14
  - Deleting Web Service Information .............................................................................. 9-16
  - Exiting the Editing of Web Service Information ......................................................... 9-18
- Deployment in the Client System Environment ............................................................... 9-18
- Managing Web Service Information with soapmodifydd .................................................. 9-19
- Web Service Identifier and URL ....................................................................................... 9-20
  - Web Service Identifier and URL Specified in the Deployment of the Server System............... 9-20
  - Web Service Identifier Specified by the Deployment of the Client System ..................... 9-21

## Chapter 10 Operating a Web Service

- Web Service Operation Procedure .................................................................................... 10-2
- Operating the CORBA/SOAP Gateway ............................................................................. 10-3
  - CORBA/SOAP Server Gateway .................................................................................. 10-3
  - CORBA/SOAP Client Gateway .................................................................................. 10-6
- Log ................................................................................................................................... 10-9
  - Server Function Log ................................................................................................... 10-9
  - Logs of the CORBA/SOAP Client Gateway ............................................................... 10-10
  - Logs of the Client Function ........................................................................................ 10-12

## Chapter 11 Sample Program Configuration

- Sample Program Configuration .......................................................................................... 11-2
- Before Using Sample Programs ....................................................................................... 11-4
- RPC Method Sample Programs ....................................................................................... 11-6
  - Sample Program Calc ............................................................................................... 11-6
  - Sample Program ConvertName .................................................................................. 11-7
  - Sample Program CountRequest ................................................................................. 11-10
  - Sample Program ReturnArray .................................................................................... 11-11
  - Sample Program ReturnMonth .................................................................................. 11-13
  - Sample Program SimpleInout ..................................................................................... 11-15
  - Sample Program StringLength ................................................................................... 11-16
  - Sample Program Young3Persons ............................................................................... 11-18
- Sample Program that extends PlusJ2EE Component Models to Web Services .................. 11-20
Sample Program ConvertNameEJB......................................................................................11-20
Sample Program ConvertStrBean .........................................................................................11-24
Messaging Method Sample Programs........................................................................................11-29
Sample Program Reliable ......................................................................................................11-29
Notes on Use ..............................................................................................................................11-34

Chapter 12 Customizing the Web Service Container

SOAP Version Setting ...................................................................................................................12-2
Setting the HTTP Version ..........................................................................................................12-5
Setting the Logs of the Client Function ....................................................................................12-6
  Setting the Log Output Destination ......................................................................................12-6
  Outputting Logs to a File ........................................................................................................12-7
Setting the Return Value of “void” ...........................................................................................12-9
Increasing the Speed of Calling the RPC Application ...............................................................12-10
Selecting a Certificate Used for Client Authentication .............................................................12-11
Editing the Web Application File ............................................................................................12-12

Appendix A Describing Web Service Information

XML Tags Describing Web Service Information ........................................................................A-3
Contents of XML Tags in Web Service Information ..................................................................A-5
Description Example of Web Service Information .....................................................................A-16

Index
Chapter 1

Overview

This chapter provides an overview of the Interstage SOAP Service functions.
Overview of Functions

Interstage SOAP Service provides the following functions:

- Standard Java APIs for Web service
  Interstage SOAP Service supports standard Java APIs (JAXM, JAX-RPC, and SAAJ) used to develop Web services.
  - JAXM: Java(TM) API for XML Messaging
  - JAX-RPC: Java(TM) API for XML-based RPC
  - SAAJ: SOAP with Attachments API for Java(TM)

- J2EE component and CORBA component linkage (CORBA/SOAP gateway) function
  The CORBA/SOAP gateway supports use of STATEFUL Session Bean, STATELESS Session Bean, and CORBA component models as Web services.

- High reliability Web service functions (original assurance function, reliable messaging function and non-repudiation function).
  - Support of original assurance function
    The digital signature (SOAP digital signature) function and SOAP message encryption (XML encryption) function are supported for Web service communication (SOAP messages). These functions are used to prevent alteration and interception of end-to-end SOAP messages (original assurance).
  - Support of reliable messaging function
    This function assures that SOAP messages have surely reached the destination without duplication (reliable messaging). It can also certify that the messages have surely reached the destination (non-repudiation).

Interstage SOAP Service consists of the components shown in Table 1-1.

**Table 1-1 Interstage SOAP Components**

<table>
<thead>
<tr>
<th>Component name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Service Container</td>
<td>Web Service Container analyzes SOAP messages, and finds information on the applicable application and calls the application based on the Web service information managed by Web Service Manager.</td>
</tr>
<tr>
<td>Web Service Manager</td>
<td>Web Service Manager manages Web service information on the application that the Web service container calls</td>
</tr>
<tr>
<td>Web Service Information Management Tool</td>
<td>Web Service Information Management Tool consists of two tools to register or change Web service information: - Web service information edit tool (GUI) - soapmodifydd command</td>
</tr>
</tbody>
</table>
Chapter 2

Designing a Web Service

This chapter explains the design items that should be examined before creating a Web service. This includes:

- Selecting components that constitute a Web service
  Select the CORBA/SOAP gateway function if the Web service to be created includes (links with) STATEFUL Session Bean, STATELESS Session Bean, and CORBA component models.

- Selecting SOAP message exchanging method (Messaging or RPC method)

- Selecting operations (Oneway mode, RequestResponse mode, and reliable messaging function) for Messaging method only.
Selecting Components that Constitute a Web

By using the CORBA/SOAP gateway, CORBA client applications can use Web services and can also deploy CORBA server applications and EJB applications as Web services.

The Web service with which a linkage can be established via the CORBA/SOAP gateway is an RPC server or client application.

The following two types of CORBA/SOAP gateway are used:

- CORBA/SOAP server gateway
- CORBA/SOAP client gateway.

CORBA/SOAP Server Gateway

The CORBA/SOAP server gateway is used to link J2EE components or CORBA component server applications and component transaction (TD) server applications with RPC client applications.

The CORBA/SOAP server gateway is an RPC server application that is called from within Web Service Container. It operates as a client application for a linked component model.

Figure 2-1  The CORBA/SOAP Server Gateway

A CORBA/SOAP server gateway is automatically generated for each interface declaration when an IDL definition is compiled with the IDLc command. IDL compilation also generates Web service information on the CORBA/SOAP server gateway.

If an environment for operating the CORBA/SOAP server gateway has already been defined, all the user has to do to operate the CORBA/SOAP server gateway is to enter the soapmodifydd command to register Web service information generated by IDL compilation.
CORBA/SOAP Client Gateway

The CORBA/SOAP client gateway is used for linkage between a CORBA client application and RPC server application (Web service).

The CORBA/SOAP client gateway is a CORBA server application that runs on JavaVM. It consists of the following three components:

- **CORBA/SOAP client gateway**
  The CORBA/SOAP client gateway runs as a CORBA server application.

- **Data conversion servant (gateway function)**
  The data conversion servant links between an RPC server application and CORBA client application.

  A data conversion servant is automatically generated for each interface declaration when an IDL definition is compiled with the `IDLc` command.

  To enable linkage between a CORBA client application and RPC server application, the data conversion servant must be registered in the CORBA/SOAP client gateway by using a CORBA/SOAP client gateway operator command.
• CORBA/SOAP client gateway operator commands

The CORBA/SOAP client gateway operator commands are used to start or stop the CORBA/SOAP client gateway, register a data conversion servant with the CORBA/SOAP client gateway or cancel the registration, or display information on the registered data conversion servants.

– oapgwstartclgw : Starts the CORBA/SOAP client gateway.
– soapgwadclgw : Registers a data conversion servant with the CORBA/SOAP client gateway.
– soapgwdelclgw : Deletes a data conversion servant from the CORBA/SOAP client gateway.
– soapgwlistclgw : Displays information on the data conversion servants registered with the CORBA/SOAP client gateway.
– soapgwstopclgw : Stops the CORBA/SOAP client gateway.

Refer to the Reference Manual (Command Edition) for more information about the CORBA/SOAP client gateway operator commands.
Selecting SOAP Message Exchange Methods

Two types of SOAP message exchange methods are available for the Web service: Messaging and RPC.

Select the most appropriate SOAP message exchange method according to the Web service configuration and specifications.

**Messaging Method**

The Messaging method is suitable for the following Web service configuration:

- An application that handles existing XML documents is used for the Web service.
- An application that needs to operate the SOAP header (such as for propagation of Web service-specific information).
- The structure and meaning of an XML document included in the SOAP body are variable.
- Data transmission is a main task. (Data means XML documents written in the SOAP body and attachments.)

A Messaging Web service can be installed using JAXM API. The Messaging Web service can freely manipulate SOAP headers and bodies created in accordance with the SOAP specifications and the specifications for SOAP messages with attachments. It thus flexibly supports addition or modification of descriptions to XML documents. On the other hand, the application developer must be familiar with individual specifications such as SOAP and JAXM API to design the following:

- SOAP messages (SOAP header and XML documents to be included in the SOAP body)
- Attachment files
- Application for sending SOAP messages
- Application for receiving SOAP messages

The Messaging method uses the following two application calling modes:

- Oneway
- RequestResponse

Refer to Selecting Operations for more information about these modes.
**RPC Method**

The RPC method is suitable for the following Web service configuration:

- Receiving formatted data with application parameters and returning the processing results in real time.
- Linking with the CORBA/SOAP gateway.

An RPC Web service can be installed using JAX-RPC API. An RPC client application sends a SOAP message (XML document) created in accordance with the SOAP RPC specifications and the specifications for SOAP messages with attachments to Web Service Container. An applicable RPC server application is then called remotely by Web Service Container.

Data exchanging between applications is performed using the in, in/out, and out parameters and return values in the same manner as the ordinary programs calling a method (function). Therefore, the application developer should properly design the Web service parameter data (in, in/out, and out parameters, data types, and return values) to be available. Refer to Chapter 8, Support of Data Types for the data types that can be used for the parameters and return values.
Selecting Operations

The Messaging operations include the Oneway mode, RequestResponse mode, and reliable messaging function (PUSH and PULL models).

Oneway and RequestResponse Modes

Select the Oneway or RequestResponse mode depending on whether the application that receives SOAP messages (receiver application) should return the processing results in the Web service to be created.

The mode in which the receiver application returns the processing results as SOAP messages to the application that sends SOAP messages (sender application) is referred to as RequestResponse. In RequestResponse mode, the sender application sends a SOAP message and then processes the message returned by the receiver application.

On the other hand, the mode in which the receiver application returns nothing to the sender application is referred to as Oneway. In Oneway mode, communication is performed one way and the calling application (sender application) receives no response including return values from the receiver application.

Select the Oneway or RequestResponse mode depending on whether the sender application should process the return messages from the receiver application in the Web service to be created.

Selecting the Reliable Messaging Function

When a Web service is used to construct a system that generally requires reliability such as for commercial transactions between enterprises, the following types of problem may occur.

Figure 2-3 Problems that May Occur

- It is not guaranteed that a SOAP message always arrives.
  If a SOAP message to be exchanged cannot be sent or received even temporarily for some reason, a communication error results and exchanging the SOAP message fails.
The SOAP message receiver usually requires a Web server environment. Regardless of the scale of the SOAP message sender and receiver, a Web server must be installed to receive SOAP messages. Sometime, this requirement may not be accepted by the party with which SOAP messages need to be exchanged.

There is no means to prove the fact of exchange of SOAP messages afterwards. It is very difficult to prove later that a SOAP message has really been exchanged. Even when either party logs the exchange, it does not work if the reliability of the log is doubted. A written contract with stamps or signatures is often used as a certificate of a transaction.

The reliable messaging function of Interstage SOAP Service prevents loss of SOAP messages and enables exchanging SOAP messages in an environment in which either party does not always require a Web server. In addition, the non-repudiation (signature option) function enables both parties to keep certificates that prove the exchange of SOAP messages. Either party thus can prove later that SOAP messages have really been exchanged.

Figure 2-4  Reliable Messaging Prevents Problems

It is recommended to use the reliable messaging function in Web services for the following reasons:

- The Web service on the SOAP message receiver does not always require processing results in real time.
- Communication needs to be proved later.
- The Web service is installed inside a firewall.
- The Web service on the SOAP message receiver is not affected by the chronological order of SOAP messages received.
Notes

- The chronological order of SOAP messages sent and received is not guaranteed.
- When the non-repudiation function (signature option) is enabled, the reliable messaging function keeps SOAP messages for a given period.

If long-term non-repudiation needs to be implemented, an application should be arranged for long-term storage of SOAP messages. If SOAP messages with signatures needs to be kept for a long time, the keys for verification must also be kept for a long time.

When SOAP messages are kept for a long time, use the message management library for reliable messaging to fetch the SOAP messages that the reliable messaging application has confirmed delivered. After the messages have been stored by the application, delete them from the received location as required, starting from the messages no longer required.

Refer to Chapter 7 Installing the Reliable Messaging Function for more information about the message management library for reliable messaging.

**PUSH Model**

A SOAP message is sent from store B (service user) to store A (service provider). Store A sends a SOAP message acknowledging its reception of the message through the same HTTP session.

If the signature option (non-repudiation function) is enabled, a digital signature of store B is added to the SOAP message sent from store B. Store A adds its signature to the SOAP message that acknowledges its reception of the message from store B and sends it through the same HTTP session.

If an acknowledgement SOAP message is not received from store A within a given time, store B sends the same SOAP message to store A again. This function guarantees the arrival of the SOAP message. When the signature option (non-repudiation function) is enabled, both stores automatically store the SOAP digital signatures they attached to the messages. This function can prove the fact of having sent and received the messages afterwards (non-repudiation).

In the following cases, reliable messaging sending fails and an error message is posted to the system log and log file:

- A SOAP message has been sent repeatedly for a given period but a timeout occurred before an acknowledgement SOAP message was received from the server system on the receiving end.
- Resend processing became impossible because a fatal error occurred on the SOAP message receiver. An example of this is when the SOAP digital signature added to the acknowledgement SOAP message from the receiving server system could not be verified.

**PULL Model**

![Figure 2-6 The PULL Model](image)

Store B (service provider) polls store A (service user) at regular intervals to obtain a SOAP message. When a SOAP message to be sent to store B is stored at store A, the message is sent to store B as a response through the same HTTP session.

If the signature option (non-repudiation function) is enabled, a digital signature of store A is added to the SOAP message sent from store A. Store B adds its signature to the SOAP message that acknowledges its reception of the message from store A and sends it through a different HTTP session. Upon receipt of the acknowledgment SOAP message from store B, store A deletes the SOAP message that was delivered completely, using the message management API.

If an acknowledgement SOAP message is not received from store B within a given time, the SOAP message is retained. This function guarantees the arrival of the SOAP message. When the signature option is enabled, both stores automatically store SOAP digital signatures they attached to the messages. This function can provide proof of having sent and received the messages afterwards (non-repudiation).

In the following cases, reliable messaging sending fails and an error message is posted to the server system log and log file:

- A SOAP message has been sent repeatedly for a given period but a timeout occurred before the server system on the sending side received an acknowledgement SOAP message.
- Resend processing became impossible because a fatal error occurred on the SOAP message receiver. An example of this is when the SOAP digital signature added to the acknowledgement SOAP message from the sending server system could not be verified.
Model that Combines both PUSH and PULL

The PUSH and PULL models are used for asynchronous reliable messaging. These two models can be combined to enable synchronous reliable messaging.

When the service user is a client system and the service provider is a server system, use a PUSH model Web service for request sending from the service user to the service provider. Use a PULL model Web service for response sending from the service provider to the service user.

When the service user is a server system and the service provider is a client system, use a PULL model Web service for request sending from the service user to the service provider. Use a PUSH model Web service for response sending from the service provider to the service user.

In either case, allocate a sender application to the service user and a receiver application to the service provider.

Refer to Chapter 7 How to Use the Reliable Messaging Function for more information about the installation method for each model, receiver applications, and sender applications.

The reliable messaging function does not guarantee the chronological order of SOAP messages sent. If the SOAP messages to be exchanged need to be associated with chronological order, IDs used to associate SOAP messages need to be written in the messages. In addition, different message types must be used for request messages and response messages.
Selecting the Web Service Security Function

High-reliability Web Service Function

Application programs need not be modified to use the original assurance (SOAP digital signature, XML encryption) function or non-repudiation (signature option) function.

The high-reliability Web service function can be enabled or disabled as follows:

1. Specify the SSL definition name to be used as the property value of the "JavaVM option" in the IJServer WorkUnit, or specify the SSL definition name in the following file:
   - Create an SSL definition name by selecting [System] > [Security], and then [SSL] on the Interstage Management Console.

   **Windows**
   ```
   %IS_HOME%\F3FMsoap\etc\config.properties
   ```

   **Solaris OE**
   ```
   /opt/FJSVsoap/etc/config.properties
   ```

2. Using the Web service information edit tool, specify whether to enable/disable the original assurance function and non-repudiation (signature option) function.

A pair of public and private keys must be prepared in advance. For more information, refer to the following sections in the Security System Operations Guide:

- Environment Setup for Using the Security Function with Web Service (SOAP)
- How to Use SOAP Digital Signature, and XML Encryption
- How to Use the Reliable Messaging Function
SSL Communication

Encrypted communication (SSL communication) using Secure Socket Layer (SSL) is enabled by performing the following operations in an environment where a certificate management environment is configured:

1. Change the SOAP client application connection destination URL from "http://...." to "https://....".
2. Specify the SSL definition name to be used as the property value of the "JavaVM option" in the IJServer WorkUnit, or specify the SSL definition name in the following file:

   Create an SSL definition name by selecting [System] > [Security], and then [SSL] on the Interstage Management Console.

   - **Windows**
     
     `%IS_HOME%\F3FMsoap\etc\config.properties`

   - **Solaris OE**
     
     `/opt/FJSVsoap/etc/config.properties`

   For more information, refer to Environment Setup for Using the Security Function with Web Service (SOAP) in the Security System Operations Guide.
Chapter 3

Configuring an Environment

This chapter explains the environment setup required for using the Interstage SOAP Service.
Defining the Server System Environment

This section explains the environment setup for server systems (Web Service Container and server applications).

Deploying Web Service Container

1. From the Interstage Management Console, select [System] > [WorkUnits], and the [Create a new WorkUnit] tab, and create an IJServer WorkUnit. (Example: WorkUnit name: MyIJServer)
2. Select the [View WorkUnit Status] tab to list WorkUnit names, and select the IJServer WorkUnit created in 1 from the list.
4. Specify a WAR (Web application ARChive files) file (soap.war or soap_dev.war) of Web Service Container for “Deploy file” and click the “Deploy” button.
   The WAR file (“/” + WAR file prefix) specified for “Deploy file” becomes the Web application name.

The above operations configure a Web Service Container execution environment with Web application name “/soap” on IJServer WorkUnit “MyIJServer”.

The WAR file for Web Service Container is stored in the following directory:

Windows

%IS_HOME%\F3FMsoap\war

Solaris OE  Linux

/opt/FJSVsoap/war

Table 3-1  File Names and Descriptions

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soap.war</td>
<td>This file is used to configure an environment for Web Service Container used in the operating environment.</td>
</tr>
<tr>
<td></td>
<td>Web Service Container reads Web service information only once when the IJServer WorkUnit starts.</td>
</tr>
<tr>
<td>soap_dev.war</td>
<td>This file is used to configure an environment for Web Service Container used in the development environment.</td>
</tr>
<tr>
<td></td>
<td>Each time a request is received from a client application, Web Service Container checks whether Web service information has been updated (added, modified, or deleted). If the Web service information has been updated, Web Service Container reads the information.</td>
</tr>
</tbody>
</table>
Setting Up the Server Application Environment

1. From the Interstage Management Console, select [System] > [WorkUnit] > [MyIJS] (IJS WorkUnit name) > [Settings] tab, and [WorkUnit Settings] where "Classpath" is displayed. For "Classpath," specify the server application, path of a user-defined class file, or JAR file name.

2. Register Web service information.

3. Restart the IJS WorkUnit in which Web Service Container is deployed.

Note

The server application must be deployed in advance in the server system by some means (such as FTP).
Defining the Client System Environment

Defining Environment Variables

Check that the following paths (directory names and/or file names) are set in environment variables:

Windows

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSPATH</td>
<td>Add the following JAR file for Interstage SOAP service:</td>
</tr>
<tr>
<td></td>
<td>%IS_HOME%\J2EE\lib\isj2ee.jar</td>
</tr>
<tr>
<td></td>
<td>[For JDK1.3.1]</td>
</tr>
<tr>
<td></td>
<td>%IS_HOME%\F3FMsoap\lib\issoap.jar</td>
</tr>
<tr>
<td></td>
<td>[For JDK1.4]</td>
</tr>
<tr>
<td></td>
<td>%IS_HOME%\F3FMsoap\lib\issoap4.jar</td>
</tr>
<tr>
<td></td>
<td>- %JAVA_HOME%\jdk\lib\tools.jar</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>Set the following directory:</td>
</tr>
<tr>
<td></td>
<td>- JDK installation directory</td>
</tr>
<tr>
<td>PATH</td>
<td>Set the following directory:</td>
</tr>
<tr>
<td></td>
<td>- %JAVA_HOME%\bin</td>
</tr>
<tr>
<td></td>
<td>- %IS_HOME%\F3FMsoap\bin</td>
</tr>
</tbody>
</table>
## Defining the Client System Environment

### Table 3-3 Environment Variables

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| CLASSPATH            | Add the following JAR file for Interstage SOAP service:  
|                      | /opt/FJSVj2ee/lib/isj2ee.jar  
|                      | [For JDK1.3.1]  
|                      | /opt/FJSVsoap/lib/issoap.jar  
|                      | [For JDK1.4]  
|                      | /opt/FJSVsoap/lib/issoap4.jar  
|                      | - %JAVA_HOME%/jdk/lib/tools.jar |
| JAVA_HOME            | Set the following directory:  
|                      | - JDK installation directory |
| PATH                 | Set the following directory:  
|                      | - $JAVA_HOME/bin  
|                      | - /opt/FJSVsoap/bin |

For information about environment definitions required for SSL communication, refer to Chapter 14 How to Prepare PKI Environment for Web Services (SOAP) in the Security System Guide.

### Definition for Using a Proxy for the Reliable Messaging Function

If communication via a proxy is to be performed for the reliable messaging function of a high-reliability Web service function, add information on the proxy to the file shown below.

For communication via a proxy for SOAP message exchange methods excluding the reliable messaging function, see Chapter 4 Installing Applications that Use the Messaging Method or Chapter 5 Installing Applications that Use the RPC Method.

**Windows**

%IS_HOME%\F3FMsoap\conf\reliable.conf

**Solaris OE Linux**

/opt/FJSVsoap/conf/reliable.conf
Example

A sample statement of the reliable.conf file is shown below:

```plaintext
com.fujitsu.interstage.soapx.proxy.use = true
com.fujitsu.interstage.soapx.proxy.host = proxy.somewhere.com
com.fujitsu.interstage.soapx.proxy.port = 8080
com.fujitsu.interstage.soapx.proxy.user = user
com.fujitsu.interstage.soapx.proxy.passwd = pass
```

Proxy information is summarized below:

**Table 3-4 Proxy Information**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.use</td>
<td>true / false</td>
<td>Specify &quot;true&quot; for communication via a proxy.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.host</td>
<td>Host name</td>
<td></td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.port</td>
<td>Port number</td>
<td></td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.user</td>
<td>User name</td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.passwd</td>
<td>Password</td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
</tbody>
</table>

Note

If the proxy definition is changed while the application using the reliable messaging function is active, restart the application after the definition is changed.
Defining the Environment for the High-reliability Web Service Function

The high-reliability Web service function is roughly classified into the following two functions:

- Original assurance function
- Reliable messaging function

A pair of public and private keys must be prepared to use the original assurance function to prevent unauthorized modification or wiretapping of SOAP messages. A pair of these keys must also be prepared to use the non-repudiation (signature option) function of the reliable messaging function to prevent remote party’s repudiation in terms of SOAP message exchanging.

For more information, refer to Chapter 14 How to Prepare PKI Environment for Web Services (SOAP), Chapter 15 XML Digital Signature and XML Encryption for Web Services (SOAP) and Chapter 16 How to Use Reliable Messaging Function for Web Services (SOAP) in the Security System Guide.
Configuring the CORBA/SOAP Gateway Environment

CORBA/SOAP Server Gateway

This section explains the environment to be defined for linkage between the CORBA/SOAP server gateway and EJB, CORBA, or TD (component transaction) server applications.

- Use the `soapmodifydd` command to store Web service information on the CORBA/SOAP server gateway that was automatically generated during IDL compilation.

- To link with a CORBA server application
  - When storing Web service information on the CORBA/SOAP server gateway, match the Web service identifier with the naming service object name registered in the CORBA naming service (with the `OD_or_adm` command with the `-n` option).

- To link with a TD application
  - Allocate an EJB application to IJServer WorkUnit type IJServer (Web+EJB [separate VM]) or IJServer (EJBOnly).
  - To link with a TD server application
    - When storing Web service information on the CORBA/SOAP server gateway, match the Web service identifier with the naming service object name registered in the CORBA naming service.
    - Transaction management using the database linkage function and the user authentication function using the `odlogin` command cannot be used.

- To link with an EJB application
  - Allocate an EJB application to IJServer WorkUnit type IJServer (Web+EJB [separate VM]) or IJServer (EJBOnly).
  - To link with an EJB server application, match the VL of the Java runtime environment in which the EJB server application runs with the VL of the Java runtime environment in which the CORBA/SOAP server gateway runs.
  - If the Java class is specified for the method parameter or return value, no private member is transferred. Specify a class having only the public member for the method parameter and return value.
  - To store Web service information on the CORBA/SOAP server gateway, use an enterprise bean name for the Web service identifier.
  - Among EJB applications, only session beans can be linked via the CORBA/SOAP server gateway.
    - Access an entity bean or message-driven bean via a session bean.
CORBA/SOAP Client Gateway

Before starting operation of the CORBA/SOAP client gateway, allocate a Java class file or JAR file that works as a data conversion servant as shown below:

- **JAR file**: From the Interstage Management Console, select [System] > [WorkUnits] > [MyIJServer] (IJServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings] where "Classpath" is displayed. Add a JAR file name to "Classpath."

- **Java class file**: From the Interstage Management Console, select [System] > [WorkUnit] > [MyIJServer] (IJServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings] where "Classpath" is displayed. Allocate a Java class file under the directory specified in "Classpath" or under the following directory:

  - **Windows**
    
    `%IS_HOME%\F3FMsoap\classes`

  - **Solaris OE**
    
    `/opt/FJSVsoap/classes`

  - **Linux**
    
    `/opt/FJSVsoap/classes`
Chapter 4

Installing Applications that Use the Messaging Method

This chapter explains how to create applications that use the Messaging method (Messaging applications).

The following two types of Messaging applications are used:

- **Sender application**
  A sender application uses SOAP to perform processing for sending XML messages to the remote application.

- **Receiver application**
  A receiver application performs processing for SOAP messages received from the sender application.

For details of Java (TM) API for XML Messaging (JAXM) and SOAP with Attachments API for Java(TM) (SAAJ) API used for application installation, refer to the JavaDOC prepared for each API.
Creating a Receiver Application

A receiver application must be created as a Java class that contains the default constructor and that is equipped with one of the following interfaces (OnewayListener and ReqRespListener).

- If the application only receives SOAP messages and sends no reply SOAP messages (Oneway mode), install the OnewayListener interface.

- If the application receives SOAP messages and sends reply messages (RequestResponse mode), install the ReqRespListener interface.

Even if the ReqRespListener interface is installed, the application can refrain from sending a SOAP reply message by returning null as the return value of the onMessage method.

- Installing the OnewayListener interface

```java
package javax.xml.messaging;
public interface OnewayListener {
    public void onMessage( javax.xml.soap.SOAPMessage receivedMessage );
}
```

- Installing the ReqRespListener interface

```java
package javax.xml.messaging;
public interface ReqRespListener {
    public javax.xml.soap.SOAPMessage onMessage( javax.xml.soap.SOAPMessage receivedMessage );
}
```

A SOAP message is represented as a javax.xml.soap.SOAPMessage object.

The receiver application adds necessary processing to the onMessage method defined in the above interface.
Receiver Application that does not Reply (OneWay Mode)

This section uses sample program SampleMsgSV1Oneway.java as an example for explaining processing.

Example

```java
import javax.xml.soap.*;
import javax.xml.messaging.*;

public class SampleMsgSV1Oneway implements OnewayListener //**(1)**
{
    public SampleMsgSV1Oneway () { }

    public void onMessage( SOAPMessage message ) //**(2)**
    {
        //**(3)**
        try{
            message.writeTo(System.out);
        }catch( Exception e ) {
            e.printStackTrace();
        }
    } //**(4)**
}
```

Installing Oneway Listener

For a receiver application that only receives SOAP messages and sends no reply SOAP messages to the sender application, the javax.xml.messaging.OnewayListener interface is installed and then the public void onMessage method is installed. (1), (2)

Processing Received SOAP Messages

The SOAPMessage object that represents a received SOAP message is passed as a method parameter. (2)

The receiver application uses SAAJ-API to perform necessary processing including the analysis of the SOAPMessage object. See SOAP Message Processing by SAAJ-API for the analysis of the SOAPMessage object.

In this example, the SOAP messages received are placed on the standard output. (3)

No method return value is returned. (4)
Receiver Application that Replies (RequestResponse Mode)

This section uses sample program SampleMsgSV2ReqResp.java as an example for explaining processing.

Example

```java
import javax.xml.soap.*;
import javax.xml.messaging.*;

public class SampleMsgSV2ReqResp implements ReqRespListener
{
    public SampleMsgSV2ReqResp () { }

    public SOAPMessage onMessage( SOAPMessage message ) {
        try{
            message.writeTo(System.out);
            MessageFactory mf = MessageFactory.newInstance();
            SOAPMessage resp = mf.createMessage();
            SOAPEnvelope env = resp.getSOAPPart().getEnvelope();
            SOAPBody body = env.getBody();
            SOAPBodyElement elm = body.addBodyElement(
                    env.createName( "ResponseMessage", "m",
                    "urn:SampleMsg" ));
            elm.addChildElement( env.createName("Response") ).
                    addTextNode("response string...");
            elm.addNamespaceDeclaration( "m", "urn:SampleMsg" );
            resp.saveChanges();
            return resp;
        } catch( Exception e ) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Installing ReqRespListener

For a receiver application that receives and processes SOAP messages and then returns some SOAP messages to the sender application, the javax.xml.messaging.ReqRespListener interface is installed and then the public SOAPMessage onMessage method is installed. (1), (2)
Processing Received SOAP Messages

The SOAPMessage object that represents a received SOAP message is passed as a method parameter. (2)

The receiver application uses SAAJ-API to analyze the SOAPMessage object and perform necessary processing. See SOAP Message Processing by SAAJ-API for the analysis of the SOAPMessage object. In this example, the SOAP messages received are placed on the standard output. (3)

Creating and Sending a Reply SOAP message

The receiver application creates a SOAP message to be returned to the sender application. It creates the message as a SOAPMessage object by using SAAJ-API.

There are two ways to create a SOAPEnvelope object representing a SOAP envelope. One is to assemble it with application logic using an API such as the createName or addBodyElement method. One is to retrieve XML data such as DOM and SAX, or XML data that has been read from existing files, as the contents of an SOAP envelope. See SOAP Message Processing by SAAJ-API for more information.

This example uses the method of assembling with application logic to create the SOAP message shown below. Note that the following example inserts a linefeed and null space between elements for better understanding. The SOAP message actually created by the sample program has no linefeed and null space shown here. To create a SOAP message in the format shown below, text nodes for line feeds and null spaces need to be added explicitly. (4)

```xml
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance>
  <soapenv:Body>
    <m:ResponseBody xmlns:m="urn:SampleMsg">
      <Response>response string...</Response>
    </m:ResponseBody>
  </soapenv:Body>
</soapenv:Envelope>
```

The receiver application returns the SOAPMessage object created as the return value of a method. (5)
Using Attachment Files

An attachment file means data that can be attached to a SOAP message. Receiver applications in either Oneway or RequestResponse mode can use attachment files.

This section uses sample program SampleMsgSV3useAttachment.java as an example for explaining processing.

Example

```java
import javax.xml.soap.*;
import javax.xml.messaging.*;
import java.util.*;
import java.io.*;
import java.awt.Image;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;

public class SampleMsgSV3useAttachment implements ReqRespListener {**(1)**

  public SampleMsgSV3useAttachment () { }

  public SOAPMessage onMessage( SOAPMessage message )
  {

    String basePath = "C:\\temp\\";

    try{**<br/>

    //**(2)**
    Iterator _i = message.getAttachments();
    int _num = 0;
    while(_i.hasNext()){
      StringBuffer _name = new StringBuffer();
      AttachmentPart _at = (AttachmentPart)_i.next();
      Object _content = _at.getContent();
      String _contentType = _at.getContentType();
      if ( _contentType == null ) {
        return makeFault( "Server.Internal",
                         "attachment has no content type." );
      }
      _contentType = _contentType.toLowerCase();
      _name.append("attachment").append(_num++);
    }**<br/>

    //**(3)**
    if( _content instanceof String ){
      String _data = (String)_content;
      _name.append(".txt");
    }

```
FileOutputStream atFile = new FileOutputStream(basePath+_name);
atFile.write(_data.getBytes("UTF-8"));
atFile.close();
_attachments.put( new String(basePath+_name),
_at.getContentType());
}/**(4)**
else if ( _contentType.startsWith( "text/xml" )
    || _contentType.startsWith( "application/xml" ) ) {
StreamSource _source = ( StreamSource ) _at.getContent();
_name.append("\.xml");
FileOutputStream atFile =
    new FileOutputStream(basePath+_name);
InputStream _is = _source.getInputStream();
byte[] _ba = new byte[_is.available()];
_is.read(_ba);
atFile.write(_ba);
atFile.close();
_attachments.put( new String(basePath+_name),
_at.getContentType());
}/**(5)**
else if ( _contentType.startsWith( "image/jpeg" ) ) {
    Image _image = (Image)_content;
    _name.append("\.jpeg");
    FileOutputStream atFile =
        new FileOutputStream(basePath+_name);
    DataHandler _imageHandler = _at.getDataHandler();
    _imageHandler.writeTo( atFile );
atFile.close();
Attachments.put( new String(basePath+_name),
_at.getContentType());
} else{
    return makeFault( "Server.Internal",
    "unexpected attachment type.");
}/**while
/**(6)**
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage resp = mf.createMessage();
SOAPEnvelope env = resp.getSOAPPart().getEnvelope();
SOAPBody body = env.getBody();
SOAPBodyElement elm = body.addBodyElement(
    env.createName( "ResponseBody", "m", "urn:Sample1" ));
elm.addChildElement( env.createName("Response")).addTextNode("response string...");
/**(7)**
Enumeration _e = _attachments.keys();
while(_e.hasMoreElements()) {
    String _path = (String)_e.nextElement();
    String _mime = (String)_attachments.get(_path);
    FileDataSource _fds = new FileDataSource(_path);
    AttachmentPart _ap = resp.createAttachmentPart(
new DataHandler(_fds));
_ap.setContentType(_mime);
resp.addAttachmentPart(_ap);
}

resp.saveChanges();
return resp;

} catch( Exception e ) {
e.printStackTrace();
return makeFault( "Server.Internal", e.toString());
}

public SOAPMessage makeFault(String faultCode,String faultString )
{
try{
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage resp = mf.createMessage();
SOAPEnvelope env = resp.getSOAPPart().getEnvelope();
SOAPBody body = env.getBody();
SOAPFault fault = body.addFault();
fault.setFaultCode(faultCode);
fault.setFaultString(faultString);
resp.saveChanges();
return resp;
} catch( Exception e ) {
e.printStackTrace();
return null;
}
}

Installing OnewayListener or ReqRespListener
As with a receiver application that does not process attachment files, the OnewayListener interface (for Oneway mode) or the ReqRespListener interface (for RequestResponse mode) is installed. (1)

In Oneway mode, the receiver application can process the attachment file of the received SOAP message. In RequestResponse mode, the receiver application can process the attachment file of the received SOAP message, and in addition, can return a SOAP message with an attachment file.

Extracting Attachment Files from a Received SOAP Message
Attachment files are extracted from the SOAPMessage object that represents the SOAP message received. (2)
The attachment files that have been extracted using the getAttachments method are collectively represented in the java.util.Iterator object. Null is returned if no attachments are found.

The element of the attachment file that is returned as Iterator is retained as a javax.xml.soap.AttachmentPart object. The application obtains the actual attachment files using the AttachmentPart.getContent method.
Creating a Receiver Application

Processing to be Performed when the MIME Type is text/plain

When the MIME type of the attachment file received is text/plain, the attachment file object obtained by the AttachmentPart.getContent method is represented as a java.lang.String object. (3)

The sample program outputs attachment files to a text file.

Processing to be Performed when the MIME Type is text/xml or application/xml

When the MIME type of the attachment file received is text/xml or application/xml, the attachment file object obtained by the AttachmentPart.getContent method is represented as a javax.xml.transform.Source (javax.xml.transform.stream.StreamSource in this example) instance. (4)

The sample program outputs attachment files to an XML file.

Processing to be Performed when the MIME Type is image/gif or image/jpeg (Acquisition as a DataHandle Object)

The attachment files received can be obtained as a javax.activation.DataHandler object by the AttachmentPart.getDataHandler method regardless of the MIME type. javax.activation.DataHandler is an API class of Java Activation Framework (JAF). An API can be used to obtain stream data. See the JavaDoc for information on API. (5)

The sample program outputs attachment files as stream data to a file.

If the attachment file is obtained using the AttachmentPart.getContent method when the MIME type is image/gif or image/jpeg, the attachment file object is represented as a java.awt.image instance.

Creating and Sending a Reply SOAP Message

A reply SOAP message to be returned to the sender application is created as a SOAPMessage object. (6)

Setting Attachment Files in a Reply SOAP Message

SAAJ-API is used to set attachment files in the Message object that represents a reply SOAP message to be returned to the sender application. Any number of attachment files can be set in one SOAP message. Each attachment file is represented as a javax.xml.soap.AttachmentPart object and can be generated using the SOAPMessage.createAttachmentPart method.

The AttachmentPart object thus generated is set in the SOAPMessage object by the SOAPMessage.addAttachmentPart method. The sample program assumes setting the attachment file, which was sent from the sender application, in the reply SOAPMessage object. (7)
Returning "fault"

If a problem occurs during processing by the receiver application, error information "fault" can be used, instead of the ordinary processing results, as the reply data to be sent to the sender application.

This section uses sample program SampleMsgSV4retFault.java as an example for explaining processing.

Example

```java
import java.util.*;
import javax.xml.soap.*;
import javax.xml.messaging.*;

public class SampleMsgSV4retFault implements ReqRespListener {
    public SampleMsgSV4retFault () { }

    public SOAPMessage onMessage( SOAPMessage message ) {
        try{
            boolean isFault = false;
            String beName = null;
            SOAPEnvelope env = message.getSOAPPart().getEnvelope();
            SOAPBody body = env.getBody();
            Iterator it = body.getChildElements();
            while(it.hasNext()) {
                SOAPElement elm = (SOAPElement)it.next();
                beName = elm.getElementName().getLocalName();
                if( !beName.equals("RequestBody") ) {
                    isFault = true;
                    break;
                }
            }
        
            MessageFactory mf = MessageFactory.newInstance();
            SOAPMessage resp = mf.createMessage();
            env = resp.getSOAPPart().getEnvelope();
            body = env.getBody();

            //**(1)**
            if( isFault ){
                SOAPFault fault = body.addFault();
                fault.setFaultCode("BadElementName");
                fault.setFaultString("BodyElement name is invalid.");
                Detail detail = fault.addDetail();
                DetailEntry entry = detail.addDetailEntry(
                    env.createName( "FaultDetail", "m", "urn:SampleMsg" ));
                entry.addChildElement( env.createName( "Reason") ).
                    addTextNode("BodyElement name is unexpected:" + beName);
            }
            else{
                SOAPBodyElement elm = body.addBodyElement(
                    env.createName( "ResponseBody", "m", "urn:SampleMsg" ));
                elm.addChildElement( env.createName("Response") ).
            }
        }
    }
}
```
Setting "fault" in the Reply SOAP Message

To return "fault" to the sender application, SAAJ-API is used to set fault information in the body of the SOAP message to be returned.

To set fault information, the SOAPBody.addFault method is used to generate a SOAPFault object and then each SOAPFault method is used to assemble detailed information. The sample program sets a fault code, fault character string, and Detail information. (1)
Handling of SOAP Header

Processing the SOAP Header Using Web Service Container

The high-reliability Web service function uses a SOAP header element (SOAPHeaderElement) to add SOAP digital signature and XML encryption information to the SOAP message.

The handling of the SOAP header element (SOAPHeaderElement) can be specified using "Processed HeaderElement" while Web service information is stored. If "Delete" is selected, the SOAP header element (SOAPHeaderElement) used by the high reliability Web service function is deleted before the receiver application is called.

mustUnderstand processing

If the mustUnderstand attribute of the SOAP header element (SOAPHeaderElement) addressed to the Web service is true or 1, the SOAP header element (SOAPHeaderElement) is processed as follows:

1. A check is made to see whether the SOAPHeaderElement can be interpreted.
2. If it cannot be interpreted, the predetermined "fault" is generated.
3. If it can be interpreted, necessary processing is performed.

The following method can be used to check whether the mustUnderstand attribute of the SOAP header element (SOAPHeaderElement) is true or 1:

```
javax.xml.soap.SOAPHeaderElement.getMustUnderstand()
```

"Do not delete" may be selected for "Processed HeaderElement" in Web service information. In this case, the SOAP header element (SOAPHeaderElement) for which interpretation processing has already been done by the high reliability Web service function is also included in the SOAP message to be received by the receiver application.

Using the isProcessed method after casting the javax.xml.soap.SOAPHeaderElement object to the following class can check whether interpretation processing has already been done for the SOAP header element (SOAPHeaderElement).

- Package name: com.fujitsu.interstage.soapx.message

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPHeaderElement</td>
<td>public boolean isProcessed()</td>
<td>Returns information on whether the SOAP header element (SOAPHeaderElement) has already been processed by the high reliability Web service function.</td>
</tr>
</tbody>
</table>
Creating a Sender Application

Messaging Web services run in either Oneway or RequestResponse mode.

Create a sender application according to the mode (Oneway or RequestResponse) of the send destination Web service (receiver application).

The method of creating a sender application is basically the same between the Oneway and RequestResponse modes, except the following:

- In Oneway mode, no reply SOAP message is sent from the receiver application. The sender application does not perform processing for receiving reply SOAP messages.
- In RequestResponse mode, reply SOAP messages are sent from the receiver application. The sender application performs processing for receiving reply SOAP messages.

Unlike the receiver application, the sender application need not install a special interface. The sender application can use any method in any Java class to perform send processing.

A general flow for basic processing is as follows:

1. Specifying the connection destination and setting up a virtual connection
2. Obtaining the SOAPMessage object to be sent
3. Setting information in the SOAPMessage object to be sent
4. Sending the SOAPMessage
5. Processing the reply SOAP message received (only in RequestResponse mode)
6. Postprocessing the virtual connection

Note

The reliable messaging function can be used in Oneway mode. The sender application that uses this function is a little different from the above program. See Chapter 7 How to Use the Reliable Messaging Function.
Basic Sender Application

This section uses sample program SampleMsgCL1.java as an example for explaining processing.

Example

```java
import javax.xml.soap.*;
import java.net.URL;
import java.io.*;
import java.util.*;

public class SampleMsgCL1 {
    public static void main( String[] args ) { // (1)
        try{
            //**(2)**
            String endPoint = new String("http://localhost/soap/services/Sample1");
            //**SSL**
            //**String endPoint = new
            String(endPoint = new
            String("https://localhost/soap/services/Sample1");
            //**(3)**
            SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
            SOAPConnection conn = scf.createConnection();
            //**(4)**
            MessageFactory mf = MessageFactory.newInstance();
            SOAPMessage msg = mf.createMessage();
            //**(5)**
            SOAPPart part = msg.getSOAPPart();
            SOAPEnvelope env = part.getEnvelope();
            env.getHeader().detachNode();
            SOAPBody body = env.getBody();
            SOAPBodyElement elm = body.addBodyElement(
                env.createName( "RequestBody", "m", "urn:SampleMsg" ));
            elm.addChildElement( env.createName("Request") ).
                addTextNode("request string…");
            elm.addNamespaceDeclaration( "m", "urn:SampleMsg" );
            msg.saveChanges();
            FileOutputStream requestFile = new FileOutputStream("request.msg");
            msg.writeTo(requestFile);
            requestFile.close();
            //**(6)**
            SOAPMessage reply = conn.call( msg, endPoint );
            //**(7)**
            FileOutputStream replyFile = new FileOutputStream("reply.msg");
            reply.writeTo(replyFile);
            replyFile.close();
            //**(8)**
        }
    }
}
```
Creating a Sender Application

```java
part = reply.getSOAPPart();
env = part.getEnvelope();
body = env.getBody();
if( body.hasFault()){  
    System.out.println("Body is SOAPFault.");
    SOAPFault fault = body.getFault();
    String faultActor = fault.getFaultActor();
    String faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
    System.out.println("faultActor :"+faultActor);
    System.out.println("faultCode  :"+faultCode);
    System.out.println("faultString:"+faultString);
    Iterator it = fault.getChildElements();
    while(it.hasNext()){  
        SOAPElement detailElm = (SOAPElement)it.next();
        System.out.println("Detail:"+detailElm);
    }
}
else{  
    System.out.println("Body is not SOAPFault.");
    Iterator it = body.getChildElements();
    while(it.hasNext()){  
        SOAPElement bodyElm = (SOAPElement)it.next();
        System.out.println("Element:"+bodyElm);
    }
}

/**(9)**
conn.close();

/**(10)**
} catch( SOAPException e ) {  
    System.out.println("SOAPException raised.");
    System.out.println("  Message:"+e.getMessage());
    System.out.println("  Cause  :"+e.getCause());
e.printStackTrace();
} catch( Throwable t ) {  
t.printStackTrace();
}
```

Any Method of any Class Usable for Implementation

Unlike the receiver application, the sender application requires no class to be installed or inherited. It can use any method of any class to perform send processing. (1)

Specifying the Connection Destination and Setting Up a Virtual Connection

A virtual connection to the SOAP message send destination is set up.
The URL of the send destination is prepared. (2)
The SOAPConnection object is obtained form the SOAPConnectionFactory object. (3)
If the reliable messaging function is to be used, obtain the ProviderConnection object from the ProviderConnectionFactory object. See Chapter 7 Installing the Reliable Messaging Function for more information.

Obtaining the SOAP Message Object to be Sent

The SOAPMessage object that represents the SOAP message to be sent is obtained. The SOAPMessage object is obtained from the MessageFactory object. (4) ProviderConnection is required when the reliable messaging function is used. See Chapter 7 Installing the Reliable Messaging Function for more information.

Setting Information in the SOAP Message to be Sent

SAAJ-API is used to create a SOAP message to be sent. (5) This example uses a method of creating the contents of SOAPEnvelope directly on the program. A method of using existing XML data is also available. See Sender Application Using Existing XML Data for more information about this method.

In a client system, Web service information having the Web service identifier that matches the name space name (urn: SampleMsg in this example) of the SOAP body sub-element is applied. See Chapter 9 Managing Web Service Information. The sample program creates the SOAP message shown below. Note that the following example inserts a linefeed and null space between elements for better understanding.

```
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
     xmlns:xsd=http://www.w3.org/2001/XMLSchema
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <m:RequestBody xmlns:m="urn:SampleMsg">
      <Request>request string…</Request>
    </m:RequestBody>
  </soapenv:Body>
</soapenv:Envelope>
```

Sending a SOAP Message

The SOAPConnection.call method is used for sending messages. When the send destination Web service runs in RequestResponse mode and has a reply message, a SOAP message returned from the Web service is returned. If no reply message is to be sent such as when the send destination Web service runs in Oneway mode, null is returned. (6)

When the reliable messaging function is used, the send method of the ProviderConnection class is used for sending messages. See Installing the Reliable Messaging Function for more information.

Processing the Reply SOAP Message that has been Returned

In RequestResponse mode, the reply SOAP message sent from the Web service is analyzed using SAAJ-API. Refer to SOAP Message Processing by SAAJ-API for information on the analysis of SOAP messages.

This example outputs the received SOAP message to a file named “reply.msg”. (7)
Checking for “fault”

“Fault” may be sent back from the server system in response to the SOAP message sent to the server system. “Fault” is set in the SOAP body and returned to the client system. The sender application first checks whether “fault” is set in the SOAP body and, if “fault” is found, performs error processing. SAAJ-API is used to fetch “fault” information. The sample program outputs the detailed information on “fault” to the standard output. (8)

Post-processing of Connection

The virtual connection that is no longer needed is closed. (9)

Exception Processing

Exception SOAPException may occur while a SOAP message is created or analyzed using SAAJ-API. SOAPException is returned from the runtime module if the runtime module detects a message error during transmission of a SOAP message or if a connection failure occurs. Applications that use SAAJ-API must catch SOAPException. The sample program outputs exception information and stack trace to the standard output. (10)

Using Attachment Files

An attachment file means data that can be attached to a SOAP message.

This section uses sample program SampleMsgCL2useAttachment.java as an example for explaining processing.

Example

```java
import javax.xml.soap.*;
import java.net.URL;
import java.io.*;
import javax.activation.DataHandler;

public class SampleMsgCL2useAttachment {
    public static void main( String[] args ) {
        try{
            String endPoint = new String("http://localhost/soap/services/Sample1");
            //**SSL**
            //**String endPoint = new
            String endPoint = new
            String(https://localhost/soap/services/Sample1");

            SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
            SOAPConnection conn = scf.createConnection();
            MessageFactory mf = MessageFactory.newInstance();

            SOAPMessage msg = mf.createMessage();
            SOAPPart part = msg.getSOAPPart();
            SOAPEnvelope env = part.getEnvelope();
            env.getHeader().detachNode();

            SOAPBody body = env.getBody();
            SOAPBodyElement elm = body.addBodyElement(
                env.createName("RequestBody", "m", "urn:SampleMsg"));
```
Obtaining the SOAP Message Object to be Sent

The SOAPMessage object that represents the SOAP message to be sent is obtained. The SOAPMessage object is obtained from MessageFactory.

If the reliable messaging function is used, ProviderConnectionFactory is used to obtain the SOAPMessage object. Refer to Chapter 7 How to Use the Reliable Messaging Function for more information.

Setting Information in the SOAP Message to be Sent

SAAJ-API is used to create a SOAPMessage object that represents the SOAP message to be sent. This example uses a method of directly programming the contents of SOAPEnvelope. A method of using existing XML data is also available. Refer to Sender Application Using Existing XML Data for more information about this method.
Setting Attachment Files in the SOAP Message to be Sent

SAAJ-API is used to set attachment files in the Message object that represents a SOAP message to be sent. Any number of attachment files can be set in one SOAP message. Each attachment file is represented as a javax.xml.soap.AttachmentPart object and can be generated using the SOAPMessage.createAttachmentPart method. The sample program generates attachment files using the javax.activation.DataHandler object. (1), (2), (3)

The AttachmentPart object thus generated is set in the SOAPMessage object using the SOAPMessage.addAttachmentPart method.
Chapter 4: Installing Applications that Use the Messaging Method

Sender Application Using Existing XML Data

One method of creating the SOAP envelope of a SOAP message is to assemble a SOAPEnvelope object based on application logic using SAAJ-API such as the createName or addBodyElement method. Another method is also available. It retrieves XML data such as DOM and SAX, or XML data that has been read from existing files, as the contents of the SOAP envelope.

This section uses sample program SampleMsgCL3importXML.java as an example for explaining processing.

Example

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import java.net.URL;
import java.io.*;
import javax.xml.transform.Source;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;

public class SampleMsgCL3importXML
{
    //**(1)**
    private static Source createStreamSource() throws java.io.FileNotFoundException
    {
        FileInputStream _reqMsg = new FileInputStream("send.txt");
        StreamSource _ss = new StreamSource(_reqMsg);
        return _ss;
    }

    //**(2)**
    private static Source createDOMSource() throws ParsingException
    {
        DocumentBuilderFactory _dbf = DocumentBuilderFactory.newInstance();
        _dbf.setNamespaceAware(true);
        _dbf.setValidating(false);
        DocumentBuilder _db = _dbf.newDocumentBuilder();
        Document _doc = _db.newDocument();
        Element _elm1 = _doc.createElementNS( "urn:SampleMsg",
            "m:RequestBody" );
        Element _elm2 = _doc.createElement("Request");
        org.w3c.dom.Text _txt = _doc.createTextNode("request string...");
        _elm2.appendChild(_txt);
        _elm1.appendChild(_elm2);
        _doc.appendChild(_elm1);

        Document _envDoc = _db.newDocument();
        Element _envElm = _envDoc.createElementNS(
            SOAPConstants.URI_NS_SOAP_ENVELOPE, "soapenv:Envelope" );
        Element _bodyElm = _envDoc.createElementNS( 
            SOAPConstants.URI_NS_SOAP_ENVELOPE, "soapenv:Body" );
```
org.w3c.dom.Node _envNode =
    _envDoc.importNode(_doc.getDocumentElement(), true);
_bodyElm.appendChild(_envNode);
_envElm.appendChild(_bodyElm);
_envDoc.appendChild(_envElm);
Element _envRootElm = _envDoc.getDocumentElement();
DOMSource _ds = new DOMSource(_envRootElm.getOwnerDocument());
return _ds;
}

public static void main( String[] args )
{
    try{
        if( args.length < 1 ) usage();
        int _mode = 0;
        if( args[0].equals("DOM") ) _mode = 1;
        else if( args[0].equals("Stream") ) _mode = 2;
        else _mode = 3;
        if( _mode > 2 ) usage();

        String endPoint = new String("http://localhost/soap/services/Sample1");
        //**SSL**
        //**String endPoint = new
        String endPoint = new String("https://localhost/soap/services/Sample1");

        SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
        SOAPConnection conn = scf.createConnection();
        MessageFactory mf = MessageFactory.newInstance();
        SOAPMessage msg = mf.createMessage();
        SOAPPart part = msg.getSOAPPart();

        Source _source = null;
        switch(_mode) {
        case 1:
            _source = createDOMSource();
            break;
        case 2:
            _source = createStreamSource();
            break;
        }

        //**(3)**
        part.setContent(_source);
        msg.saveChanges();
        SOAPMessage reply = conn.call( msg, endPoint );

        SOAPPart _respPart = reply.getSOAPPart();
        //**(4)**
        _source = _respPart.getContent();
        if(_source instanceof StreamSource ){
            InputStream _is = ((StreamSource) _source).getInputStream();
            byte[] _ba = new byte[_is.available()];
            _is.read(_ba);
            FileOutputStream _replyMsg = new FileOutputStream("reply.txt");
            _replyMsg.write(_ba);
Creating a SOAP Message Using Existing XML Data

Sample program SampleMsgCL3importXML.java uses a method of creating a SOAP message based on the data read from an existing XML file as well as a method of using a DOM object.

Creating a SOAP Message Based On Data Read from an Existing XML File

A javax.xml.transform.stream.StreamSource object is used to create a SOAP message based on such XML data read from files for which a java.io.InputStream object becomes the input source.

The javax.xml.transform.stream.StreamSource class is a class for installing a javax.xml.transform.Source interface. Sample program SampleMsgCL3importXML.java creates a javax.xml.transform.stream.StreamSource object based on the contents of the file named “send.txt”. (1)

Creating a SOAP Message Using a DOM Object

A javax.xml.transform.dom.DOMSource object is used to create a SOAP message using the DOM object that has been assembled using DOM-API.

The javax.xml.transform.dom.DOMSource class is a class for installing a javax.xml.transform.Source interface. Sample program SampleMsgCL3importXML.java creates the SOAP message shown below using DOM-API and then generates a javax.xml.transform.dom.DOMSource object from the DOM object. (2)

```xml
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance>
  <soapenv:Body>
    <m:RequestBody xmlns:m="urn:SampleMsg">
      <Response>request string…</Response>
    </m:RequestBody>
  </soapenv:Body>
</soapenv:Envelope>
```
Setting XML Data in, and Extracting it from, a SOAP Message

A javax.xml.transform.Source object created based on the XML object that uses XML data read from files and DOM objects can be set in a SOAP message by using the SOAPPart.setContent method. (3)

Similarly, the data that has been received, using the SOAPPart.getContent method, from the SOAPMessage object that represents the received SOAP message can be extracted as a javax.xml.transform.Source object. (4)

Adding HTTP Connection Information including Proxy, Authentication, and Session Information

The following classes can be used for the sender application to define HTTP connection information.

- Package name: com.fujitsu.interstage.soapx.soap

Table 4-1  Sender Application Classes for Defining HTTP Connection Information

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPConnectionImpl</td>
<td>public void setProxy(String hostName, int port)</td>
<td>Defines the proxy host name (hostName) and port number (port).</td>
</tr>
<tr>
<td></td>
<td>public void setProxy(String hostName, int port, String authName, String authPassword)</td>
<td>Defines the proxy host name (hostName), port number (port), HTTP basic authentication user name (authName), and HTTP basic authentication password (authPassword).</td>
</tr>
<tr>
<td></td>
<td>public void setProxyUserName(String authName)</td>
<td>Defines the HTTP basic authentication user name (authName) of the proxy.</td>
</tr>
<tr>
<td></td>
<td>public void setProxyPassword(String authPassword)</td>
<td>Defines the HTTP basic authentication password (authPassword) of the proxy.</td>
</tr>
<tr>
<td></td>
<td>public void setUserName(String authName)</td>
<td>Defines the authentication user name (authName) of the Web service. This definition is valid for HTTP basic authentication.</td>
</tr>
<tr>
<td></td>
<td>public void setPassword(String authPassword)</td>
<td>Defines the authentication password (authPassword) of the Web service. This definition is valid for HTTP basic authentication.</td>
</tr>
<tr>
<td></td>
<td>public void setMaintainSession(boolean maintain)</td>
<td>Defines whether to continue the HTTP session.</td>
</tr>
<tr>
<td></td>
<td>public void setTimeout(int timeout)</td>
<td>Sets the connection timeout time in milliseconds.</td>
</tr>
</tbody>
</table>
Using a Proxy

To use a proxy connection, cast the javax.xml.soap.SOAPConnection object to the com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl class and then call the proxy information definition method.

```java
import javax.xml.soap.*;
import com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl;

...;
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection conn = scf.createConnection();

//**(1)define proxy information. **
((SOAPConnectionImpl)conn).setProxy( "proxyHostName", 8080 );

//**(2)define proxy information. **
((SOAPConnectionImpl)conn).setProxy( "proxyHostName", 8080, "suzuki", "taro" );

//**(3)define proxy information. **
((SOAPConnectionImpl)conn).setProxy( "proxyHostName", 8080 );
((SOAPConnectionImpl)conn).setProxyUserName( "suzuki" );
((SOAPConnectionImpl)conn).setProxyPassword( "taro" );
...;
```

To use a proxy connection, define the following items:

- Proxy host name (java.lang.String): Required
- Proxy port number (int): Required
- Proxy user name (java.lang.String): Optional
- Proxy password (java.lang.String): Optional

The proxy host name and port number are required items. If a connection is to be set up with the proxy server without using the user name and password, define only the host name and port number ((1) in the above example). If the user name and password are required to set up a connection with the proxy server, one of two definition methods can be selected. One method defines all items together ((2) in the above example), and one method defines the items individually ((3) in the above example).

Defining the User Name and Password of a Web Service

Basic authentication may be used for web. If so, cast the javax.xml.soap.SOAPConnection object to the com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl class and then call the password authentication information definition method to define the user name and password.

An example is shown below.
Example

```java
import javax.xml.soap.*;
import com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl;

....
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection conn = scf.createConnection();

// Define password authentication information.
((SOAPConnectionImpl)conn).setUserName("suzuki");
((SOAPConnectionImpl)conn).setPassword("taro");

....
```

Using Session Control

If session control is used, multiple requests from the same sender application can be processed by the same instance of the receiver application. In addition, using session control can take over the results of the previous processing and thus enables continuous processing.

To use session control, cast the `javax.xml.soap.SOAPConnection` object to the `com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl` class and then call the session control definition method.

```java
import javax.xml.soap.*;
import com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl;

....
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection conn = scf.createConnection();

// Define session control information.
((SOAPConnectionImpl)conn).setMaintainSession(true);

....
```

The session can be continued by calling the receiver application with the instance of the `com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl` class for which the retention of session control information has been defined.

If the `com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl.setMaintainSession` method is called with "false" specified as a parameter, the instance of the `com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl` class abandons the session control information and discontinues the session.
Setting the Connection Timeout

If no response is made from the connection partner for a predetermined time, a timeout occurs and the connection is released. To set the timeout time, cast the javax.xml.soap.SOAPConnection object to the com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl class and then call the timeout setting method to set the timeout time in milliseconds. If the timeout time is not set explicitly, the default timeout time (5 minutes) is used.

```java
import javax.xml.soap.*;
import com.fujitsu.interstage.soapx.soap.SOAPConnectionImpl;

....
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection conn = scf.createConnection();

// Set the timeout time (15 minutes).
((SOAPConnectionImpl)conn).setTimeout( 15*60*1000 );

....
```
SOAP Message Processing by SAAJ-API

Note

If MIME type image/jpeg or image/gif or java.awt.Image type attachment files are to be used, an environment in which X Window system applications run is required.

Structure of SOAP Message

![SOAP Message with Attachment Files](image)

Figure 4-1 SOAP Message with Attachment Files
The above figure shows the structure of a SOAP message having attachment files. SAAJ-API maps the individual elements of the SOAP message to the Java classes shown at right in the figure.

The SOAPMessage object that represents a SOAP message consists of the SOAPPart object that contains the contents of the SOAP envelope and AttachmentPart objects that represent individual attachment files.

### Creating a SOAP Message

The MessageFactory object is used to create a SOAPMessage object.

A MessageFactory object can be obtained using the newInstance class method of the MessageFactory class.

```java
import javax.xml.soap.*;
......
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
SOAPEnvelope env = part.getEnvelope();
......
```

An application that uses the reliable messaging function can generate a MessageFactory object from the ProviderConnection object as shown below. See Chapter 7 Installing the Reliable Messaging Function for information on the application that uses the reliable messaging function.

```java
//For application that uses the reliable messaging function
import javax.xml.soap.*;
import javax.xml.messaging.*;
......
ProviderConnectionFactory pcf = ProviderConnectionFactory.newInstance();
ProviderConnection pc = pcf.createConnection();
MessageFactory mf = pc.createMessageFactory(...);
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
SOAPEnvelope env = part.getEnvelope();
......
```

### SOAP Envelope Processing

The SOAPEnvelope object is an object that represents a SOAP envelope. One SOAP message always includes one SOAP envelope.

A SOAPEnvelope object can be obtained using the SOAPPart.getEnvelope method. The SOAPEnvelope object API is used to operate the SOAP header or SOAP body directly. The SOAP envelope includes zero or one SOAP header and one SOAP body.

```java
import javax.xml.soap.*;
......
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
```
SOAPEnvelope env = part.getEnvelope();

// Processing when SOAPHeader is not used
SOAPHeader header = env.getHeader();
header.detachNode();

// Processing when SOAPHeader is used
// SOAPHeader header = env.getHeader();
// SOAPHeaderElement helm = header.addHeaderElement(
//     env.createName(“Header1”, “ns1”, “urn:Sample”));
// helm.addNamespaceDeclaration(“ns1”, “urn:Sample”);

// SOAPBody setting
SOAPBody body = env.getBody();
SOAPBodyElement belm = body.addBodyElement(
    env.createName(“Body1”, “ns1”, “urn:Sample”));
belm.addNamespaceDeclaration(“ns1”, “urn:Sample”);

If the SOAP header is not to be used, call the SOAPHeader.detachNode method explicitly or do nothing. If nothing is done, a null header (<xxx:Header/>)) is output.

The SOAPHeader, SOAPHeaderElement, SOAPBody, and SOAPBodyElement interfaces each inherit the SOAPElement interface and therefore can continue to set or obtain subelements by using the API of the SOAPElement interface.

Note
- If a required name space declaration is not performed in or under the SOAPEnvelope object, a required name space declaration is automatically added to the SOAP envelope data to be sent or sent back when it is sent or sent back.
- When adding a SOAP digital signature or using XML encryption, be sure to perform a required name space declaration in or under the SOAPEnvelope object.

For processing the SOAP envelope, the API of the SOAPEnvelope interface can be used to directly manipulate the SOAPHeader and SOAPBody objects. In addition, XML data can be used to set the XML object as is in the SOAP envelope or the contents of the SOAP envelope can be extracted as XML data. For this operation, use the setContent/getContent method of the SOAPPart object.

import javax.xml.soap.*/
import javax.xml.transform.Source;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;

.....
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();

// Create a SAXSource object.
SAXSource source = .....;

// Create a DOMSource object.
DOMSource source = .....;
Chapter 4: Installing Applications that Use the Messaging Method

// Create a StreamSource object.
StreamSource source = .....;

// Create SOAPMessage from the Source object.
part.setContent(source);
.....

// Obtain SOAPMessage as a Source object.
Source source = part.getContent();
.....

SAXSource, DOMSource, and StreamSource each are classes in which a Source interface is installed. An object in which a Source interface is installed can be defined as a SOAP envelope by the SOAPPart.setContent method. Similarly, the contents of a SOAP envelope can be obtained as a Source object by the SOAPPart getContent method. See the sample for a usage example.

Processing of Attachment Files

A SOAPMessage object can include one SOAPPart object and zero or more AttachmentPart objects. javax.xml.soap.AttachmentPart is an object that represents each attachment file. To generate an AttachmentPart object, use the SOAPMessage.createAttachmentPart method. To set AttachmentPart in the SOAPMessage object, use the SOAPMessage.addAttachmentPart method.

To extract attachment files from the SOAPMessage object, use the SOAPMessage.getAttachments method. The SOAPMessage.countAttachments method can be used to obtain the number of attachment files.

```java
import javax.xml.soap.*;
import java.net.URL;
import java.io.FileInputStream;
import java.util.Iterator;
import javax.activation.DataHandler;

MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
.....
// Set AttachmentPart.(1)
URL url = new URL("file:///C:/temp/sample.txt");
AttachmentPart ap1 = msg.createAttachmentPart( new DataHandler(url));
ap1.setContentType("text/plain");
msg.addAttachmentPart(ap1);
.....
// Set AttachmentPart.(2)
AttachmentPart ap2 = msg.createAttachmentPart();
```
Two or more AttachmentPart objects can be set for one SOAPMessage object. The above example sets two AttachmentPart objects.

**Attachment File Data**

When the getContent and setContent methods are used for AttachmentPart objects, data conversion is performed according to the class used. The data of the received attachment file may thus differ from that of the attachment file sent from the sender application.

If the same data as that sent needs to be received (to prevent data conversion), use the getDataHandler and setDataHandler methods.
Fault Processing

The application that sends a SOAP message may receive a fault message as return information from
the server system. SAAJ-API represents a fault as a SOAPFault object. The SOAPFault object is set
as a subelement of the SOAPBody object in a SOAPMessage object.

The application that has caused a fault generates a SOAPFault object using the SOAPBody.addFault
method. The application that receives a fault can take out the SOAPFault object using the
SOAPBody.getFault method. The sender application can use the SOAPBody.hasFault method to check
whether a fault is included in the SOAPMessage object received.

The SOAPFault object retains the following types of information, which can be set or obtained by
applications:

- Fault code (java.lang.String)
- Fault description (java.lang.String)
- Fault actor (java.lang.String)
- Fault detail (java.xml.soap.Detail)

```
import javax.xml.soap.*;
import java.util.Iterator;

MessageFactory mf = MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
SOAPPart part = msg.getSOAPPart();
SOAPEnvelope env = part.getEnvelope();
env.getHeader().detachNode();
SOAPBody body = env.getBody();

// Set SOAPFault.
SOAPFault fault = body.addFault();
fault.setFaultActor(Fault);
fault.setFaultCode(Fault);
fault.setFaultString(Fault);
Detail detail = fault.addDetail();
DetailEntry entry = detail.addDetailEntry( ..... );
entry.addChildElement(...);

SOAPMessage reply = .....;
// Check whether a fault occurred.
env = reply.getSOAPPart().getEnvelope();
body = env.getBody();
if( body.hasFault() ) {
// Fault occurrence
    fault = body.getFault();
    String faultActor = fault.getFaultActor();
    String faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
    detail = fault.getDetail();
    Iterator it = detail.getDetailEntries();
    while( it.hasNext() ){
        entry = (DetailEntry)it.next();
```
The SOAPFault interface inherits the SOAPBodyElement interface that represents a subelement of SOAPBody. SOAPFault, SOAPFaultElement, Detail, and DetailEntry interfaces each inherit the SOAPElement interface and therefore permit fault information to be assembled the same as for assembling normal SOAP messages.

Analyzing Fault Information

If an error occurs during invocation processing of the receiver application (application in the server system), fault information is set in the reply message received by the sender application (application in the client system). The SOAPBody.hasFault method is used to check whether fault information is set in the reply message received. If the return value from the method is "true," fault information is set in the reply message.

Fault information is set as a sub-element of the SOAP body the same as with a normal SOAP message. Fault information is represented as a SOAPFault object and can be extracted using the SOAPBody.getFault method. The following shows a program for receiving a reply message and extracting fault information.

```java
import javax.xml.soap.*;
......
// Receive response data.
SOAPMessage response = conn.call( msg, endPoint );
SOAPEnvelope env = response.getSOAPPart().getEnvelope();
// Extract the SOAPBody
SOAPBody body = env.getBody();
// Check whether fault information is set.
if( body.hasFault() ) {
    // Extract the fault information.
    SOAPFault fault = body.getFault();
    // Extract the fault code.
    String faultCode = fault.getFaultCode();
    // Extract the fault description.
    String faultString = fault.getFaultString();
    // Extract the fault detail.
    Detail detail = fault.getDetail();
    Iterator it = detail.getDetailEntries();
    ......
} else{
    // Fault information is not set (normal).
    ......
}
```

Fault information includes the following items:

- Fault code
- Fault description
- Fault actor
- Fault detail

**Table 4-2 Fault Information**

<table>
<thead>
<tr>
<th>Information type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Fault code       | Classification of fault. One of the following character strings is set:  
|                  | - “Server”  
|                  | - “Client”  
|                  | See the "Fault code classification" shown below for the meanings of these character strings. |
| Fault description| Character strings that describe the fault |
| Fault actor      | URI that indicates the error location (such as the receiver application) |
| Fault detail     | Error information items defined for each Web service |

**Note**

For fault information under SOAP1.2, the getFaultCode method returns a fault code Value element.
Extracting the Fault Code

Use the following method of the SOAPFault object to extract the fault code:

```java
java.lang.String getFaultCode()
```

- Fault code classification

**Table 4-3 Fault Codes**

<table>
<thead>
<tr>
<th>Fault code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Indicates that a problem was detected in Web Service Container or during processing by the receiver application. Returned to indicate that, for instance, Web Service Container contains a setting error.</td>
</tr>
<tr>
<td>Client</td>
<td>Indicates that the sender application detected an error in the message it sent. If this fault code is returned to the sender application, check the message sent to the receiver application.</td>
</tr>
<tr>
<td>Other fault code value</td>
<td>A fault code other than the above may be returned if:</td>
</tr>
<tr>
<td></td>
<td>- The Web service has defined its own fault code.</td>
</tr>
</tbody>
</table>

Extracting the Fault Description

To extract the fault description, use the following method of the SOAPFault object:

```java
java.lang.String getFaultString()
```

Extracting the Fault Actor

To extract the fault actor, use the following method of the SOAPFault object:

```java
java.lang.String getFaultActor()
```
Chapter 4: Installing Applications that Use the Messaging Method

Extracting the Fault Detail

The fault detail represents the information items specific to a Web service consisting of an arbitrary number of fault detail items.

To extract the fault detail, use the following method of the SOAPFault object:

```java
c javax.xml.soap.Detail  getDetail()
```

The individual detail items included in the fault detail are represented as javax.xml.soap.DetailEntry objects and can be extracted by using the following method of the Detail object:

```java
c java.util.Iterator  getDetailEntries()
```

The individual fault detail items are retained as javax.xml.soap.DetailEntry objects in the java.util.Iterator object. The following shows a program for extracting fault detail items.

```java
import javax.xml.soap.*;
......
if( body.hasFault()) {
    SOAPFault fault = body.getFault();
    String faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
    // Extract the fault information
    Detail detail = fault.getDetail();
    // Extract the fault detail.
    Iterator it = detail.getDetailEntries();
    while(it.hasNext()) {
        DetailEntry entry = (DetailEntry)it.next();
        ......
    }
} else{
    ......
}
```

The javax.xml.soap.DetailEntry interface inherits the javax.xml.soap.SOAPElement interface. Therefore, the individual detail items included in the fault detail can be operated as pure SOAPElement objects.
Chapter 5

Installing Applications that Use the RPC Method

This chapter explains how to create an application that uses the remote procedure call (RPC) method. Refer to the JavaDOC prepared for each API for details of the "Java(TM) API for XML-based RPC(JAX-RPC)" and "SOAP with Attachments API for Java(TM)(SAAJ)" that are used to install applications.

The following two types of RPC applications are used:

- RPC client application
  - This application issues a remote procedure call to a Web service.
  - This application is created as an arbitrary Java application and placed in a client system.
  - This type of application can be created in one of two methods: stub method and DII method.

- RPC server application
  - This application is called remotely from an RPC server application.
  - This application is created as a Java class and placed in the server system.
  - This application is called upon arrival of a remote procedure call SOAP message under control of Web Service Container. See Chapter 9 Managing Web Service Information for more information about the creation and placement.
  - This application can return a return value or out parameter to the remote procedure calling application.
Chapter 5: Installing Applications that Use the RPC Method

Stub Method and DII Method

The stub generation function of the integrated development environment (IDE) tools (such as Interstage Apworks that supports the JAX-RPC stub method) is used to generate a stub for calling an RPC server application. The RPC client application can use this stub to easily call the RPC server application remotely.

The DII method uses API for high-level developers. It implements remote procedure calling by dynamically assembling calling interfaces such as parameters and method names as desired.

- If the interface for the called RPC server application is already decided, the stub method is recommended.
- If the interface of the called RPC server application is not decided until it is executed, the DII method is used.
Handling Attachment Files

To allow an RPC application to handle attachment files, use the attachment file type for RPC application parameters (including return values).

If the attachment file type is used for a parameter of the sender application, the applicable data is sent as an attachment file of a SOAP message (rather than as a SOAP body).

The receiver application prepares the attachment file type as a parameter and receives it as parameter information.

For more information about the attachment file type, see Chapter 8 Support of Data Types. The following sections provide examples of coding an RPC server application and RPC client application that handle attachment files.
**RPC Server Application**

This section explains how to create an RPC server application.

**Defining the Interface**

First, the RPC server application interface (method name, parameter data type, etc.) needs to be defined.

Create an interface that inherits the javax.rmi.Remote interface and write a method that accepts remote procedure calls. For this method, declare RemoteException with throws. For the data types that can be used as parameters, see Chapter 8 Support of Data Types.

Below, is an example of using parameters of simple types `int` and `UnsignedInt`, and attachment file type `Source`.

**Example**

Example of definition of interface that accepts remote procedure calls

- Method name: opInt
- Parameter: `int` (in), `UnsignedInt` (in), `int` (inout or out), `javax.xml.transform.Source` (inout or out)
- Return value: `int`

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import javax.xml.rpc.holders.IntHolder;
import com.fujitsu.interstage.soapx.types.UnsignedInt;
import com.fujitsu.interstage.soapx.holders.SourceHolder;

{
    public int opInt( int arg1, UnsignedInt arg2, IntHolder arg3, SourceHolder arg4 )
            throws RemoteException;    // Declare java.rmi.RemoteException with throws.
}
```

**Installing the RPC Server application**

An RPC server application must be created with a Java class that includes the default constructor, and in which the interface defined above is installed.

```java
import java.rmi.RemoteException;
import javax.xml.rpc.holders.IntHolder;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import com.fujitsu.interstage.soapx.holders.SourceHolder;
import com.fujitsu.interstage.soapx.types.UnsignedInt;
import java.io.FileInputStream;

public class IntSampleImpl implements IntSample
{
}
```
public IntSampleImpl()
{
}

public int opInt(int arg1, UnsignedInt arg2, IntHolder arg3, SourceHolder arg4 )
                      
    throws RemoteException
{
    try {
        System.out.println("opInt");
        System.out.println(" arg1 = "+arg1);
        System.out.println(" arg2 = "+arg2.longValue());
        arg3.value = arg1 + arg2.intValue();
        arg4.value = new StreamSource( new FileInputStream("attach.xml") );
        return arg1 + arg2.intValue();
    } catch( Exception e ) {
        return -1;
    }
}

The RPC server application is called when the Web Service Container detects a remote procedure call SOAP message (in the same way that an ordinary method is called).

Initializing and Postprocessing an Instance

The RPC server application is instantiated in accordance with the method specified in Web service information. Install the following interface to initialize and postprocess an instance.

```java
package javax.xml.rpc.server;
public interface ServiceLifecycle {
    public void init(Object context) throws javax.xml.rpc.ServiceException;
    public void destroy();
}
```

After the RPC server application is instantiated, the init method is called. State initialization processing in the init method. The ServletEndPoingContext object shown below is passed to the parameter.

The RPC server application is called when Web Service Container detects a remote procedure call SOAP message (in the same way that an ordinary method is called).

When the instance of the RPC server application is released, the destroy method is called. State postprocessing in the destroy method.

```java
package javax.xml.rpc.server;
public interface ServletEndpointContext {
    public java.security.Principal getUserPrincipal();
    public javax.xml.rpc.handler.MessageContext getMessageContext();
    public javax.servlet.http.HttpSession getHttpSession();
    public javax.servlet.ServletContext getServletContext();
}
```
Using Session Control

If session control is used, multiple requests from the same RPC client application can be processed by the same instance of the RPC server application. Therefore, using session control can take over the results of the previous processing and enables continuous processing.

Defining Web Service Information

In the Web service information entry window of Web Service Information Edit Tool, select "session" for "Web service application life cycle."

Refer to Chapter 9 Managing Web Service Information for more information.

Setting the Session Timeout

By default, the timeout time for session control is 30 minutes. To change the timeout time, edit the Web application (web.xml) file.

For more information, see Chapter 12 Customizing the Web Service Container.
RPC Client Application Using the Stub Method

The Integrated Development Environment (IDE) stub generation function generates an RPC server application stub. The RPC client application can use this stub to easily call the server application remotely in the same way as when calling an ordinary method.

For information about stub generation, refer to the document prepared for the IDE used.

Basic RPC Client Application (Stub Method)

This section uses sample program SampleRpcCL1.java as an example for explaining processing.

Example

```java
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.holders.IntHolder;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.Detail;
import javax.xml.namespace.QName;
import javax.xml.transform.Source;
import com.fujitsu.interstage.soapx.types.UnsignedInt;
import com.fujitsu.interstage.soapx.holders.SourceHolder;
import java.util.Iterator;

public class SampleRpcCL1 {
    public static void main( String[] args ) //**(1)**
    {
        try {
            //**(2)**
            QName serviceQName = new QName( "urn:IntSample", "IntSampleService" );

            //**(3)**
            ServiceFactory factory = ServiceFactory.newInstance();
            IntSampleServiceLocator locator = (IntSampleServiceLocator)
                factory.createService(serviceQName);
            IntSample target = locator.getIntSamplePort();

            //**(4)**
            int arg1 = 100;
            UnsignedInt arg2 = new UnsignedInt(200l);
            IntHolder arg3 = new IntHolder();
            SourceHolder arg4 = new SourceHolder();

            System.out.println( "arg1 = " + arg1 );
            System.out.println( "arg2 = " + arg2.longValue() );

            //**(5)**
            int result = target.opInt( arg1, arg2, arg3, arg4 );

            //**(6)**
            System.out.println( "arg3 = " + arg3.value );
        }
    }
}
```
System.out.println( "ret = " + result );

  /**(7)** *
} catch( javax.xml.rpc.ServiceException e ){
  e.printStackTrace();
  if( e.getLinkedCause() != null ){
    e.getLinkedCause().printStackTrace();
  }
} catch( javax.xml.rpc.soap.SOAPFaultException e ){
  e.printStackTrace();
  System.out.println("  Code = " + e.getFaultCode());
  System.out.println("  String = " + e.getFaultString());
  System.out.println("  Actor = " + e.getFaultActor());
  if( e.getDetail() != null ){
    Detail detail = e.getDetail();
    Iterator it = detail.getDetailEntries();
    if( it != null ){
      while(it.hasNext()){
        SOAPElement elm = (SOAPElement)it.next();
        System.out.println("    Detail = " + elm);
      }
    }
  }
} catch( javax.xml.rpc.JAXRPCException e ){
  e.printStackTrace();
  if( e.getLinkedCause() != null ){
    e.getLinkedCause().printStackTrace();
  }
} catch( java.rmi.RemoteException e ){
  e.printStackTrace();
  if( e.detail != null ){
    e.detail.printStackTrace();
  }
} catch( Throwable e ) {
  e.printStackTrace();
}
}

Any Method of Any Class Usable for Installation

The RPC client application requires no class to be installed or inherited. It can use any method of any class to perform send processing. (1)
Specifying the Remote Application to be Called

Specify the remote RPC server application to be called, using the javax.xml.namespace.QName object. (2)

To create a javax.xml.namespace.QName object, the following types of information indicated by WSDL are required:

- Name space name
  Value with the namespace attribute of the SOAP body (soap: body) element in the operation elements (urn:IntSample in this example)

- Local name
  Value with the name attribute of the service element associated with a binding element including an operation element (IntSampleService in this example)

Obtaining a Stub object of the Remote Application to be Called

Obtain a stub object for the remote application to be called. (3)

To do so, first use the createService method of the javax.xml.rpc.ServiceFactory object to obtain a locator object.

Specify the javax.xml.namespace.QName object prepared in (2) in the parameter of the createService method.

Next obtain a stub object from the locator object.

Preparing Parameters Used for Remote Procedure Calling

Prepare the parameters to be set for remote procedure calling. (4)

The parameters include not only the ordinary in parameter that carries the value set by the calling application to the called application but also the out and inout parameters. The out parameter carries no value from the calling application and carries a value set by the called application to the calling application. The inout parameter carries a value set by the calling application to the called application and carries the value updated by the called application back to the calling application.

The holder class in the javax.rpc.soap.holders or com.fujitsu.intertstage.soapx.holders package is used for the out and inout parameters. In this example, arg3 and arg4 are used as out parameters.

See Chapter 8 Support of Data Types for the data types that can be used.

Executing a Remote Procedure Call

Call the RPC server application method (operation) in the same way as for Java method calling. (5)

Calling the stub object method (opInt in this example) executes remote procedure calling.

The name of the called method (operation) is the value with the name attribute of an operation element in the portType elements associated with a WSDL binding element.

Obtaining the Results of Remote Procedure Calling

Obtain the results of calling the RPC server application. To do so, obtain the return value and out/inout parameter with the value member of the holder object. (6)
Exception Processing

The RPC client application needs to catch the following exceptions: (7)

- **javax.xml.rpc.ServiceException**
  
  This exception is thrown when an error occurred during processing of the javax.xml.rpc.ServiceFactory object or locator object. The getLinkedCause method can be used to obtain the exception instance that actually occurred. In this example, a stack trace is placed on the standard output.

- **javax.xml.rpc.JAXRPCException**
  
  This exception is thrown when SOAP message communication failed (because of a parameter error or connection failure) during remote procedure calling. The getLinkedCause method can be used to obtain the exception instance that actually occurred. In this example, a stack trace is placed on the standard output.

- **javax.xml.rpc.soap.SOAPFaultException**
  
  This exception is thrown when SOAPFault was returned from the RPC server application or Web Service Container during remote procedure calling.
  
  SOAPFault is exception information defined by SOAP. It holds the following types of information:
  
  - Fault actor (java.lang.String)
  - Fault code (java.lang.String)
  - Fault description (java.lang.String)
  - Fault detail (java.xml.soap.Detail)
  
  The RPC client application can obtain these types of exception information. In this example, fault information is placed on the standard output.

- **java.rmi.RemoteException**
  
  This exception is thrown to the RPC client application when the RPC server application causes java.rmi.RemoteException or an exception arising from it.
  
  The java.rmi.RemoteException object holds a throwable object named detail. In this example, a stack trace is placed on the standard output.
Adding HTTP Connection Information including Proxy and Authentication Information (Stub Method)

This section explains how the RPC client application using a stub can define additional HTTP connection information.

Specifying the Connection Destination URL

A stub issues a remote procedure call to the default connection destination URL defined by WSDL. If the connection destination URL needs to be specified explicitly, use the following method:

```java
import javax.xml.rpc.Stub;

.....
ServiceFactory factory = ServiceFactory.newInstance();
IntSampleServiceLocator locator = (IntSampleServiceLocator)
    factory.createService(serviceQName);

// To use the default connection destination URL
IntSample target = locator.getIntSamplePort();

// To specify the connection destination URL
IntSample target = locator.getIntSamplePort( new
javainet.URL("http://otherhost/anywhere"));

.....
// To change the connection destination URL
((Stub)target)._setProperty("javax.xml.rpc.service.endpoint.address",
    "http://anotherhost:8080/somewhere/");
```

If a method (getIntSamplePort in this example) that obtains a stub object from the locator object is called without a parameter, the connection destination URL defined by WSDL is used. If the method is called with the connection destination URL (java.net.URL object) specified as a parameter, a stub object used to set up a connection to the specified destination URL can be obtained.

If it is desired to set a connection destination URL in the stub object once obtained, use the setProperty method of the javax.xml.rpc.Stub interface to set the connection destination URL as a property.

The property related to the connection destination URL is shown below:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.rpc.service.endpoint.address</td>
<td>Connection destination URL</td>
</tr>
</tbody>
</table>
**Using a Proxy**

When using a proxy for connection, define proxy information as a property for the stub object.

```java
import javax.xml.rpc.Stub;

......
ServiceFactory factory = ServiceFactory.newInstance();
IntSampleServiceLocator locator = (IntSampleServiceLocator)
    factory.createService(serviceQName);
IntSample target = locator.getIntSamplePort();

// To define proxy information
((Stub)target)._setProperty("com.fujitsu.interstage.soapx.proxy.use", "true");
// Host name
((Stub)target)._setProperty("com.fujitsu.interstage.soapx.proxy.host", "myProxyHost");
// Port number
((Stub)target)._setProperty("com.fujitsu.interstage.soapx.proxy.port", "8080");
// User name
((Stub)target)._setProperty("com.fujitsu.interstage.soapx.proxy.user", "suzuki");
// Password
((Stub)target)._setProperty("com.fujitsu.interstage.soapx.proxy.passwd", "taro");
......
```

Proxy information is summarized in the table below.

**Table 5-1  Proxy Information**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.use</td>
<td>“true” / “false”</td>
<td>Specify “true” for communication via a proxy.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.host</td>
<td></td>
<td>Host name</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.port</td>
<td></td>
<td>Port number</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.user</td>
<td></td>
<td>User name</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.passwd</td>
<td></td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.passwd</td>
<td></td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
</tbody>
</table>
Defining the User Name and Password of a Web Service

If a Web service uses basic authentication is used for SOAP messages, define authentication information as a property for the stub object.

```java
import javax.xml.rpc.Stub;

.....
ServiceFactory factory = ServiceFactory.newInstance();
IntSampleServiceLocator locator = (IntSampleServiceLocator) factory.createService(serviceQName);
IntSample target = locator.getIntSamplePort();

// To define basic authentication information
// User name
((Stub)target)._setProperty( "javax.xml.rpc.security.auth.username", "suzuki ");
// Password
((Stub)target)._setProperty( "javax.xml.rpc.security.auth.password", "taro")
.....
```

The properties related to the user name and password of a Web service are as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.xml.rpc.security.auth.username</code></td>
<td>User name</td>
</tr>
<tr>
<td><code>javax.xml.rpc.security.auth.password</code></td>
<td>Password</td>
</tr>
</tbody>
</table>

Setting the Connection Timeout

If no response is made from the connection partner for a predetermined time, a timeout occurs and the connection is released. To set the timeout time, set a value in milliseconds as a property for the stub object.

```java
import javax.xml.rpc.Stub;

.....
ServiceFactory factory = ServiceFactory.newInstance();
IntSampleServiceLocator locator = (IntSampleServiceLocator) factory.createService(serviceQName);
IntSample target = locator.getIntSamplePort();

// To set the timeout time (15 minutes)
((Stub)target)._setProperty( "com.fujitsu.interstage.soapx.socket.timeout", new Integer(15*60*1000) );
```

The property related to the timeout time is as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.fujitsu.interstage.soapx.socket.timeout</code></td>
<td>Timeout time (milliseconds)</td>
</tr>
</tbody>
</table>

If the timeout time is not set explicitly, the default timeout time (5 minutes) is used.
Using Session Control

If session control is performed, a session can be used by casting the user stub class to the com.fujitsu.interstage.soapx.client.Stub class and using the following method to hold session information.

- **Package name**: com.fujitsu.interstage.soapx.client

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stub</td>
<td>implements javax.xml.rpc.Stub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>public void setMaintainSession (boolean maintainsSession)</td>
<td>Specifies whether to continue an HTTP session.</td>
</tr>
</tbody>
</table>

**Note**

When an RPC client application ends, cookie including session control information is not saved.

Session control can be maintained while the RPC client application is active.

**Setting**

Cast the stub object to the com.fujitsu.interstage.soapx.client.Stub class using the setMaintainSession method with "true" specified in the parameter. Doing this operation sets the stub object so that it holds session control information.

A session can be continued by calling an RPC server application using the stub object that has thus been set to hold session control information.

If the com.fujitsu.interstage.soapx.client.Stub.setMaintainSession method is called with "false" specified as a parameter, the instance of the com.fujitsu.interstage.soapx.client.Stub class abandons the session control information and discontinues the session.

**Note**

The number of times (times) the SOAP server application has been called from the SOAP client application using session control is compared with the value (rettimes) received from the SOAP server application as the number of times it has been called.

- **RPC server application**

```java
// RPC server application
//
import java.rmi.RemoteException;
/*
 * <code>CountRequestServer</code>
 * Deploy this with 'Session' scope
 */
public class CountRequestServer implements CountIf {
    private long n = 0;
    /*
     * return the number of requested times in the session.
     * @return number of the requested times in the session.
     */
```
RPC Client Application Using the Stub Method

public long count() throws RemoteException
{
    this.n++;
    return this.n;
}

• RPC client application

    // RPC client application
    // import javax.xml.rpc.Service;
    import javax.xml.rpc.ServiceFactory;
    import javax.xml.namespace.QName;
    import javax.xml.soap.SOAPElement;
    import javax.xml.soap.Detail;
    import com.fujitsu.interstage.soapx.client.Stub;
    import java.util.Iterator;
    import java.util.Service;

    public class CountRequestClient
    {
        static final int stopTimes=10;
        static void run( String targetURL )
        {
            try {
                QName serviceQName = new QName("urn:sample-count-request", "CountRequestService");
                ServiceFactory factory = ServiceFactory.newInstance();
                CountRequestServiceLocator locator = (CountRequestServiceLocator)factory.createService(serviceQName);
                CountRequest target = locator.getCountRequestPort();

                // Holding of session information
                ((Stub)target).setMaintainSession(true);
                System.out.println( "started. " );

                for( int times = 1; times <= stopTimes; times++ ){
                    boolean succeeded;
                    try {
                        long rettimes = target.count();
                        if( times != rettimes ){
                            succeeded = false;
                        }
                        succeeded = true;
                    } catch(SOAPException se) {
                        System.out.println("SOAPException throwed.");
                        System.out.println(se);
                        succeeded = false;
                    }

                    if( !succeeded ){
                        System.out.println("Call failed ! called " + times + ", times. (last one may be trying only)");
                        break;
                    }
                }
            }
        }
    }
public static void main(String[] args) throws Exception{
    if(args.length<1){
        System.out.println("Usage: java CountRequestClient <RPC router's URL>");
        System.exit(1);
    }
    String targetURL = args[0];
    run( targetURL );
}

Note

- If processing is continued after a session timeout occurs, a new session is established.
- Session control can be used through a linkage between an RPC client application and RPC server application of the SOAP service.
RPC Client Application Using the DII Method

With the DII method, not only a parameter value but also a desired parameter type and method name can dynamically be set for remote procedure calling. A remote procedure call can thus be issued by dynamically obtaining information on the called application by any means.

Basic RPC Client Application (DII Method)

This section uses sample program SampleRpcCL2.java as an example for explaining processing.

Example

```java
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.NamespaceConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.Detail;
import javax.xml.namespace.QName;
import javax.xml.transform.Source;
import java.util.Map;
import java.util.Iterator;
import com.fujitsu.interstage.soapx.Constants;
import com.fujitsu.interstage.soapx.types.UnsignedInt;

public class SampleRpcCL2 {
    public static void main( String[] args ) //**(1)**
    {
        try {
            //**(2)**
            String endpointURL = "http://localhost/soap/servlet/WSContainer";
            //**SSL**
            //**String endpointURL = "https://localhost/soap/servlet/WSContainer";
            QName operationName = new QName( "urn:IntSample", "opInt" );
            QName dataType = new QName( NamespaceConstants.NSPREFIX_SCHEMA_XSD, "int" );
            QName unsignedDataType = new QName( NamespaceConstants.NSPREFIX_SCHEMA_XSD, "unsignedInt" );
            //**(3)**
            Service service = ServiceFactory.newInstance().createService(new QName( "urn:IntSample", "IntSampleService" ));
            Call call = service.createCall();
            //**(4)**
            call.setTargetEndpointAddress(endpointURL);
            call.setOperationName(operationName);
            //**(5)**
            call.addParameter( "arg1", dataType, ParameterMode.IN );
        }
    }
}
```
call.addParameter( "arg2", unsignedDataType, ParameterMode.IN );
call.addParameter( "arg3", dataType, ParameterMode.OUT );
call.addParameter( "arg4", Constants.MIME_SOURCE , ParameterMode.OUT );
call.setReturnType(dataType);

//**(6)**
Integer arg1 = new Integer(100);
UnsignedInt arg2 = new UnsignedInt(200l);
Object[] param = new Object[] {arg1, arg2};

System.out.println( "arg1 = " + arg1 );
System.out.println( "arg2 = " + arg2.longValue() );

//**(7)**
Integer ret = (Integer)call.invoke( param );

//**(8)**
Map outParams = call.getOutputParams();
Integer arg3 = (Integer)outParams.get("arg3");
Source arg4 = (Source)outParams.get("arg4");

System.out.println( "arg3 = " + arg3 );
System.out.println( "ret  = " + ret );

//**(9)**
} catch( javax.xml.rpc.ServiceException e ) {
    e.printStackTrace();
    if( e.getLinkedCause() != null ) {
        e.getLinkedCause().printStackTrace();
    }
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
    e.printStackTrace();
    System.out.println("  Code = " + e.getFaultCode());
    System.out.println("  String = " + e.getFaultString());
    System.out.println("  Actor = " + e.getFaultActor());
    if( e.getDetail() != null ) {
        Detail detail = e.getDetail();
        Iterator it = detail.getDetailEntries();
        if( it != null ){
            while(it.hasNext()){
                SOAPElement elm = (SOAPElement)it.next();
                System.out.println("    Detail = " + elm);
            }
        }
    }
} catch( javax.xml.rpc.JAXRPCException e ) {
    e.printStackTrace();
    if( e.getLinkedCause() != null ) {
        e.getLinkedCause().printStackTrace();
    }
```java
} catch( java.rmi.RemoteException e ){
    e.printStackTrace();
    if( e.detail != null ) {
        e.detail.printStackTrace();
    }
} catch( Throwable e ) {
    e.printStackTrace();
}
}
```

This example issues a remote procedure call with URL “http://localhost/soap/servlet/WSContainer” operation name “opInt”, and service name “IntSample”.

**Any Method of Any Class Usable for Installation**

The RPC client application requires no class to be installed or inherited. It can use any method of any class to perform send processing. (1)

**Specifying the Remote Application to be Called**

To specify the RPC server application to be called remotely, prepare the following types of information from WSDL: (2)

- **Connection destination URL** ("http://localhost/soap/servlet/WSContainer" in this example)
  
  Specify the URL of the Web service to be connected.
  
  Value with the location attribute of the SOAP address (soap: address) element in the service elements associated with a binding element including an operation element

- **Operation name with name space name**
  
  - **Name space name** ("urn:IntSample" in this example)
    
    Value with the namespace attribute of the SOAP body (soap: body) element in the operation elements
    
    This value becomes the namespace name of a SOAP body sub-element.
  
  - **Local name**
    
    Value with the name attribute ("opInt" in this example) of an operation element in the portType elements
    
    This value becomes an operation name.

Refer to Chapter 9 Managing Web Service Information for more information about the Web service identifier and URL.

**Obtaining Remote Procedure Call object Call**

Obtain a Call object that represents a remote procedure call. (3)

The Call object manages information related to remote procedure calls and actually has a remote procedure call function.
Defining Service information for Remote Procedure Call Object Call

The connection destination URL, operation namespace name, and javax.xml.namespace.QName object that represents the service name attribute that was previously prepared. Define these elements for the Call object. (4)

Defining Operation Information for Remote Procedure Call Object Call

Specify interface information, such as the parameters of the operation for remote procedure calling and the type of the return value, for the Call object. (5)

For the return value of the operation, use the setReturnType method to specify the XML data type. For the parameters of the operation, use the addParameter method to specify the parameter name, XML data type, and in/inout/out types sequentially.

See Chapter 8 Support of Data Types for the data types that can be used for parameters and return values.

The in parameter carries a value set by the calling application to the called application. The out parameter carries no value from the calling application and carries a value set by the called application to the calling application. The inout parameter carries a value set by the calling application to the called application and carries the value updated by the called application back to the calling application.

In this example, the first parameter of the remote procedure call operation is the XML int type in parameter with parameter name "arg1". The second parameter is the XML unsignedInt type in parameter with parameter name "arg2". The third parameter is the XML int type out parameter with parameter name "arg3". The fourth parameter is the out parameter with parameter name "arg4" representing XML data (javax.xml.transform.Source type under Java) with attachment files.

Preparing an Array of Send Parameter Values for Remote Procedure Calling

Prepare an object type array of the values of the in and inout parameters for the operation to be sent to the called application. This array does not include an out parameter. (6)

Executing a Remote Procedure Call

Use the invoke method of the Call object to execute a remote procedure call. Transfer the array of send parameter values prepared in Preparing an Array of Send Parameter Values for Remote Procedure Calling as a parameter for the invoke method. (7)

If the operation in the called application has a return value, obtain it by casting the return value of the invoke method to the type of the operation return value.

Extracting the Receive Parameters for Remote Procedure Calling

If the operation in the called application has the inout or out parameter, use the getOutputParams method from the Call object, which called the invoke method, to extract the returned value.

The value of the inout or out parameter is returned with the java.util.Map object. Extract the parameter value using the parameter name as the key. (8)
Exception Processing

The client application that executes an RPC call needs to catch the following exceptions: (9)

- javax.xml.rpc.ServiceException
  This exception is thrown when an error occurred during processing of the javax.xml.rpc.ServiceFactory object or locator object. The getLinkedCause method can be used to obtain the exception instance that actually occurred. In this example, a stack trace is placed on the standard output.

- javax.xml.rpc.JAXRPCException
  This exception is thrown when SOAP message communication failed (because of a parameter error or connection failure) during RPC calling. The getLinkedCause method can be used to obtain the exception instance that actually occurred. In this example, a stack trace is placed on the standard output.

- javax.xml.rpc.soap.SOAPFaultException
  This exception is thrown when SOAPFault was returned from the RPC server application or Web Service Container during RPC calling.
  SOAPFault is exception information defined by SOAP. It holds the following types of information:
  - Fault actor (java.lang.String)
  - Fault code (java.lang.String)
  - Fault description (java.lang.String)
  - Fault detail (java.xml.soap.Detail)
  The RPC client application can obtain these types of exception information. In this example, fault information is placed on the standard output.

- java.rmi.RemoteException
  This exception is thrown to the client application when the RPC server application causes java.rmi.RemoteException or an exception arising from it.
  The java.rmi.RemoteException object holds a throwable object named detail. In this example, a stack trace is placed on the standard output.
RPC Client Application using User-defined Types (DII Method)

This section explains the RPC client application that uses user-defined types for parameters and return values. See Chapter 8 Support of Data Types for the data types that can be used as user-defined types.

The following uses sample program SampleRpcCL3.java, which uses the structure type, as an example for explaining the processing of user-defined types.

Example

```java
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.NamespaceConstants;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.Detail;
import javax.xml.namespace.QName;
import java.util.Map;
import java.util.Iterator;
import com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory;
import com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory;

public class SampleRpcCL3
{
    public static void main( String[] args )
    {
        try {
            String endpointURL = "http://localhost/soap/servlet/WSContainer";
            //**SSL**
            //**String endpointURL = "https://localhost/soap/servlet/WSContainer";
            String serviceURI = "urn:StructSample";
            QName operationName = new QName( serviceURI, "opStruct1" );
            //**(1)**
            QName dataType = new QName( serviceURI, "Struct1" );

            Service service = ServiceFactory.newInstance()
                .createService(new QName( serviceURI, "StructSampleService" ));
            Call call = service.createCall();
            //**(2)**
            TypeMappingRegistry tmr = service.getTypeMappingRegistry();
            TypeMapping tm = tmr.createTypeMapping();
            tm.register( Struct1.class, dataType,
                new BeanSerializerFactory( Struct1.class, dataType ),
                new BeanDeserializerFactory( Struct1.class, dataType ));
            tmr.register( NamespaceConstants.NSURI_SOAP_ENCODING, tm );
            call.setTargetEndpointAddress(endpointURL);
            call.setOperationName(operationName);
            call.addParameter( "arg1", dataType, ParameterMode.IN );
        }
    }
}
```
call.addParameter("arg2", dataType, ParameterMode.IN);
call.addParameter("arg3", dataType, ParameterMode.OUT);
call.setReturnType(dataType);

Struct1 arg1 = new Struct1(100, "aaaaaaaaaa");
Struct1 arg2 = new Struct1(200, "bbbbbbbbbb");
Object[] param = new Object[] {arg1, arg2};

System.out.println("arg1.a = "+ arg1.a);
System.out.println("arg1.b = "+ arg1.b);
System.out.println("arg2.a = " + arg2.a);
System.out.println("arg2.b = " + arg2.b);

Struct1 ret = (Struct1) call.invoke(param);

Map outParams = call.getOutputParams();
Struct1 arg3 = (Struct1) outParams.get("arg3");

System.out.println("arg3.a = " + arg3.a);
System.out.println("arg3.b = " + arg3.b);
System.out.println("ret.a  = " + ret.a);
System.out.println("ret.b  = " + ret.b);

} catch( javax.xml.rpc.ServiceException e ) {
e.printStackTrace();
if( e.getLinkedCause() != null ) {
e.getLinkedCause().printStackTrace();
}
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
e.printStackTrace();
System.out.println("  Code = " + e.getFaultCode());
System.out.println("  String = " + e.getFaultString());
System.out.println("  Actor = " + e.getFaultActor());
if( e.getDetail() != null ) {
  Detail detail = e.getDetail();
  Iterator it = detail.getDetailEntries();
  if( it != null ) {
    while(it.hasNext()) {
      SOAPElement elm = (SOAPElement)it.next();
      System.out.println("    Detail = " + elm);
    }
  }
}
} catch( javax.xml.rpc.JAXRPCException e ) {
e.printStackTrace();
if( e.getLinkedCause() != null ) {
e.getLinkedCause().printStackTrace();
}
} catch( java.rmi.RemoteException e ) {
e.printStackTrace();
if( e.detail != null ) {
e.detail.printStackTrace();
}
User-defined type used

The above sample program SampleRpcCL3.java uses the following structure type for the parameters and return value of the called RPC server application.

```java
public class Struct1 {
    public int a;
    public String b;

    public Struct1() {}  
    public Struct1( int a, String b ) {
        this.a = a;
        this.b = b;
    }  
    public void setA( int a )  
    {  
        this.a = a;
    }  
    public int getA()  
    {  
        return this.a;
    }  
    public void setB( String b )  
    {  
        this.b = b;
    }  
    public String getB()  
    {  
        return this.b;
    }  
}
```

If the structure type is used as a user-defined type, the Java classes must be defined in accordance with the JavaBean rules. If a class member is defined as a public member or defined as private, it must have public set and get methods. The Struct1 class used in this example has only public members and also has public set and get methods.
Preparing User-Defined Type Information

Prepare information on the user-defined parameter type used for RPC calling. (1)

The information to be prepared includes the following items defined by service WSDL:

- Namespace name and local name of user-defined type name (dataType in this example)

Defining Information on the Association Between Java Data Types and XML Data Types

Information on the association between user-defined Java data types and XML data types used for conversion between them is referred to as "type mapping" and represented by the TypeMapping object.

Java data type information (Class object) and the corresponding XML data type information (QName object) are defined in the TypeMapping object. In addition, a serializer factory and deserializer factory according to the data type are defined in the TypeMapping object. The user-defined data types include the structure, Bean, and enumeration types.

Because sample program SampleRpcCL3.java uses the structure type, the serializer factory is com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory and the deserializer is com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory. See Chapter 8 Support of Data Types for more information. (2)

Type mapping is validated by registering the TypeMapping object in the TypeMappingRegistry object using the register method. Because sample program SampleRpcCL3.java uses only the Struct1 class as the user-defined type, only one type mapping is registered. If multiple user-defined types are used, as many type mappings as there are user-defined types must be registered.

Adding HTTP Connection Information including Proxy and Authentication Information (DII Method)

This section explains how the RPC client application using DII can define additional HTTP connection information.

Using a Proxy

When using a proxy for connection, define proxy information as a property for the Call object.

```java
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Service;
import javax.xml.rpc.Call;

.....
Service service = ServiceFactory.newInstance()
      .createService(proxySampleServiceQName);
Call call = service.createCall();

// To define proxy information
call.setProperty( "com.fujitsu.interstage.soapx.proxy.use", "true");
// Host name
```
call.setProperty("com.fujitsu.interstage.soapx.proxy.host", "myProxyHost");
// Port number
call.setProperty("com.fujitsu.interstage.soapx.proxy.port", "8080");
// User name
call.setProperty("com.fujitsu.interstage.soapx.proxy.user", "suzuki");
// Password
call.setProperty("com.fujitsu.interstage.soapx.proxy.passwd","taro");
.....

Proxy information is summarized in the table below.

Table 5-2  Proxy Information

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.use</td>
<td>&quot;true&quot; / “false&quot;</td>
<td>Specify &quot;true&quot; for communication via a proxy.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.host</td>
<td>Host name</td>
<td></td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.port</td>
<td>Port number</td>
<td></td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.user</td>
<td>User name</td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
<tr>
<td>com.fujitsu.interstage.soapx.proxy.passwd</td>
<td>Password</td>
<td>Required when the proxy uses basic authentication.</td>
</tr>
</tbody>
</table>

Defining the User Name and Password of a Web Service

If basic authentication is used for Web services is used for SOAP messages, define authentication information as a property for the Call object.

```java
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Service;
import javax.xml.rpc.Call;

......
Service service = ServiceFactory.newInstance()
        .createService(basicAuthSampleServiceQName);
Call call = service.createCall();

// To define basic authentication information
// User name
    call.setProperty( "javax.xml.rpc.security.auth.username","suzuki");
// Password
    call.setProperty( "javax.xml.rpc.security.auth.password","taro");
```
The properties related to the user name and password of a Web service are as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.rpc.security.auth.username</td>
<td>User name</td>
</tr>
<tr>
<td>javax.xml.rpc.security.auth.password</td>
<td>Password</td>
</tr>
</tbody>
</table>

### Setting the Connection Timeout

If no response is made from the connection partner for a predetermined time, a timeout occurs and the connection is released. If the timeout time is to be specified, define authentication information as a property for the Call object.

```java
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Service;
import javax.xml.rpc.Call;

......
Service service = ServiceFactory.newInstance()
    .createService(timeoutSampleServiceQName);
Call call = service.createCall();

// To specify the timeout time (15 minutes)
call.setProperty("com.fujitsu.interstage.soapx.socket.timeout",
    new Integer(15*60*1000) );
......
```

The property related to the timeout time is as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fujitsu.interstage.soapx.socket.timeout</td>
<td>User name</td>
</tr>
</tbody>
</table>

If the timeout time is not set explicitly, the default timeout time (5 minutes) is used.

### Using Session Control

If session control is performed, a session can be used by casting the javax.xml.rpc.Call to the com.fujitsu.interstage.soapx.client.Call class and using the following method to hold session information.

- **Package name:** com.fujitsu.interstage.soapx.client

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>public void setMaintainSession (boolean maintainsSession)</td>
<td>Specifies whether to continue an HTTP session.</td>
</tr>
</tbody>
</table>

**Example**

When an RPC client application ends, cookie including session control information is not saved.

Session control can be maintained while the RPC client application is active.
Setting

Call the com.fujitsu.interstage.soapx.client.Call class instance using the setMaintainSession method with "true" specified in the parameter. Doing this operation sets the com.fujitsu.interstage.soapx.client.Call class instance so that it holds session control information.

A session can be continued by calling an RPC server application using the com.fujitsu.interstage.soapx.client.Call class instance that has thus been set to hold session control information.

If the com.fujitsu.interstage.soapx.client.Call.setMaintainSession method is called with "false" specified as a parameter, the instance of the com.fujitsu.interstage.soapx.client.Call class abandons the session control information and discontinues the session.

Example

The number of times (calltimes) the RPC server application has been called from the RPC client application using session control is compared with the value (resp) received from the SOAP server application as the number of times it has been called.

• RPC server application

```java
// RPC server application
//
import java.rmi.RemoteException;
/**
 * <code>CountRequestServer</code>
 * Deploy this with 'Session' scope
 */
public class CountRequestServer implements CountIf {
    private long n = 0;
    /*
     * return the number of requested times in the session.
     * return number of the requested times in the session.
     */
    public long count() throws RemoteException {
        this.n++;
        return this.n;
    }
}
```

• RPC client application

```java
// RPC client application
//
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import java.util.Map;
public class CountRequestClient {
    static final String targetOPName = "count";
    static final String targetURN = "urn:sample-countRequest";
    public void main(String[] args)
```
static final int stopTimes = 10;
static final QName datatype = new QName("http://www.w3.org/2001/XMLSchema", "long");

static void run( String targetURL )
{
    try {
        Service service = ServiceFactory.newInstance().createService(
            new QName( "urn:sample-countRequest", "count" ));
        javax.xml.rpc.Call call = service.createCall();
        call.setTargetEndpointAddress(targetURL);
        call.setOperationName( new QName( targetURN, targetOPName ));
        call.setReturnType(datatype);
        // Holding of session information
        ((com.fujitsu.interstage.soapx.client.Call)call).setMaintainSession(true);
        Object[] param = new Object[] {};
        System.out.println( "started." );
        for( int times = 1; times <= stopTimes; times++ ){
            boolean succeeded;
            Long resp = (Long)call.invoke( param );
            succeeded = proccessResponse( resp, times);
            if( !succeeded ){
                System.out.println( "Call failed ! called "+ times +" times. (last one may be trying only)" );
                break;
            }
        }
        System.out.println( "ended." );
    } catch( javax.xml.rpc.ServiceException e ) { e.printStackTrace();
    if( e.getLinkedCause() != null ) { e.getLinkedCause().printStackTrace();
    }
    } catch( javax.xml.rpc.soap.SOAPFaultException e ) { e.printStackTrace();
    System.out.println(" Code = " + e.getFaultCode());
    System.out.println(" String = " + e.getFaultString());
    System.out.println(" Actor = " + e.getFaultActor());
    if( e.getDetail() != null ) {
        Detail detail = e.getDetail();
        Iterator it = detail.getDetailEntries();
        if( it != null ) {
            while(it.hasNext()) {
                SOAPElement elm = (SOAPElement)it.next();
                System.out.println(" Detail = " + elm);
            }
        }
    }
    } catch( javax.xml.rpc.JAXRPCException e ) { e.printStackTrace();
    if( e.getLinkedCause() != null ) { e.printStackTrace();
    }
e.getLinkedCause().printStackTrace();
}
} catch( java.rmi.RemoteException e ) {
    e.printStackTrace();
    if( e.detail != null ) {
        e.detail.printStackTrace();
    }
} catch( Throwable e ) {
    e.printStackTrace();
}
}

static boolean processResponse( Long resp, int calltimes ){
    // check the response
    if( resp == null ){
        System.out.println( "--------    the call failed    --------" );
        return false;
    }
    else {
        if( resp.longValue() == calltimes ){
            System.out.println( " requested " + resp + " times." );
            return true;
        }
        else {
            System.out.println( "ERROR:Server called counter value("+resp") " +
            "differs from really called times(" +calltimes+")!");
            return false;
        }
    }
}

public static void main(String[] args) throws Exception{
    if(args.length<1){
        System.out.println("Usage: java CountRequestClient <RPC router's URL>"); System.exit(1);
    }
    String targetURL = args[0];
    run( targetURL );
}

Note

- If processing is continued after a session timeout occurs, a new session is established.
- Session control can be used through a linkage between an RPC client application and RPC server application of the SOAP service.
Defining and Analyzing Fault Information

If an error occurs during processing of the RPC server application, fault information (rather than the processing results) can be sent to the RPC client application to notify the error.

Invocation can be implemented by dynamically and freely defining fault information in the RPC server application and the types of fault information analysis parameters in the RPC client application.

Defining Fault Information

When a method of the RPC server application throws an exception, fault information is sent to the RPC client application. Fault information notifies the RPC client application of information on the problem that occurred.

The RPC server application can define the fault information to be sent by throwing javax.xml.rpc.soap.SOAPFaultException as an exception.

If the method of the RPC server application throws a throwable object other than javax.xml.rpc.soap.SOAPFaultException, fault information including the following is sent:

<table>
<thead>
<tr>
<th>Fault code</th>
<th>“Server”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault description</td>
<td>“Throwable is thrown in server application(1220)”</td>
</tr>
</tbody>
</table>

Defining Fault Information

The following information can be set in the fault information.

<table>
<thead>
<tr>
<th>Information type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault code</td>
<td>Classification of fault. One of the following character strings is set:</td>
</tr>
<tr>
<td></td>
<td>- “Server”</td>
</tr>
<tr>
<td></td>
<td>- “Client”</td>
</tr>
<tr>
<td>Fault description</td>
<td>Arbitrary character string that describes the error content</td>
</tr>
<tr>
<td></td>
<td>No limitation applies to character string length.</td>
</tr>
<tr>
<td>Fault actor</td>
<td>URI that indicates the error location (such as the server application)</td>
</tr>
<tr>
<td>Fault detail</td>
<td>Error information items defined for each Web service</td>
</tr>
</tbody>
</table>
Specify fault information as a parameter of the javax.xml.rpc.soap.SOAPFaultException constructor.

- **Fault code**
  - Local name: "Server", Namespace: ""
    Specify this code when a request from the RPC client application is valid but cannot be processed (because of a problem in the RPC server application).
  - Local name: "Client", Namespace: ""
    Specify this code when a request from the RPC client application is invalid and cannot be processed.

- **Fault description**
  Specify the character string that describes the error content. Any information can be specified.

- **Fault actor**
  Specify an arbitrary character string representing the RPC server application in URI format. Using the Web service identifier defined in Web service information is recommended.

- **Fault detail**
  Specify arbitrary information on errors specific to the application. The fault detail is represented as the javax.xml.soap.Detail object. Error information included in this object is referred to as fault detail items. As many fault detail items as needed can be specified.

  Create individual fault detail items as a javax.xml.soap.DetailEntry object,

  Set individual fault detail items as javax.xml.soap.DetailEntry objects in the javax.xml.soap.Detail object. Also, specify the javax.xml.soap.Detail object in the javax.xml.rpc.soap.SOAPFaultException constructor.

### Setting Fault Detail Items

A fault detail item is generated as a javax.xml.soap.DetailEntry object. Because the javax.xml.soap.DetailEntry object has an interface that inherits SAAJ-API javax.xml.soap.SOAPElement, operations over javax.xml.soap.SOAPElement objects can be performed directly over the javax.xml.soap.DetailEntry objects. For more information about SAAJ-API, see Chapter 4 Installing Applications that Use the Messaging Method

The following shows a sample program of an RPC server application method that throws javax.xml.rpc.soap.SOAPFaultException. When the array of in parameter String contains a specific value, this method throws javax.xml.rpc.soap.SOAPFaultException, and fault information is sent to the RPC client application.

**Example of a SOAP server application method for javax.xml.rpc.soap.SOAPFaultException**

- Throws javax.xml.rpc.soap.SOAPFaultException when a specific character ("carrot") is included in the java.lang.String array (foods).

```java
public void eat(String[] foods) throws java.rmi.RemoteException // SOAPFaultException derives from RemoteException.
{
    for (int i=0; i<foods.length; i++)
    {
        if (foods[i].equalsIgnoreCase("carrot") )
        {
            try{
                SOAPFactory sf= SOAPFactory.newInstance();
```

5-32
Detail detail = sf.createDetail();
Name name = se.createName("index","carrot","urn:sample-service-carrot");
DetailEntry idx = detail.addDetailEntry(name);
idx.addTextNode(Integer.toString(i));
throw new SOAPFaultException("Client", "I don’t like carrots!",
detail);
} catch (SOAPException se) {
    se.printStackTrace();
}

Analyzing Fault Information

If an error occurs during remote procedure call processing of the RPC server application, exception
javax.xml.rpc.soap.SOAPFaultException is thrown. Fault information that represents the results of a
remote procedure call is retained in javax.xml.rpc.soap.SOAPFaultException.

Fault information includes the following items:

- Fault code
- Fault description
- Fault detail

Specify the character string that describes the error content. Any information can be specified.

<table>
<thead>
<tr>
<th>Information type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault code</td>
<td>Classification of fault. One of the following character strings is set:</td>
</tr>
<tr>
<td></td>
<td>- Server</td>
</tr>
<tr>
<td></td>
<td>- Client</td>
</tr>
<tr>
<td></td>
<td>See the &quot;Fault code classification&quot; shown below for the meanings of these</td>
</tr>
<tr>
<td></td>
<td>character strings.</td>
</tr>
<tr>
<td>Fault description</td>
<td>Character strings that describe the fault</td>
</tr>
<tr>
<td>Fault actor</td>
<td>URI that indicates the error location (such as the server application)</td>
</tr>
<tr>
<td>Fault detail</td>
<td>Error information items defined for each Web service</td>
</tr>
</tbody>
</table>
Extracting the Fault Code

Use the following method of the javax.xml.rpc.soap.SOAPFaultException class to extract the fault code:

```java
javax.xml.namespace.Qname getFaultCode()
```

- Fault code classification

<table>
<thead>
<tr>
<th>Fault code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Indicates that a problem was detected in the Web Service Container or during processing by the receiver application. For instance, the Web Service Container contains a setting error, or Web service information contains a registration error.</td>
</tr>
<tr>
<td>Client</td>
<td>Indicates that the Web service detected an error in the message it sent. If this fault code is returned to the RPC client application, check the message sent to the Web service.</td>
</tr>
<tr>
<td>A fault code other than the above</td>
<td>A fault code other than the above may be returned if:</td>
</tr>
<tr>
<td></td>
<td>- The Web service sets an original fault code.</td>
</tr>
</tbody>
</table>

Extracting the Fault Description

Use the following method for the javax.xml.rpc.soap.SOAPFaultException class to extract the fault description:

```java
java.lang.String getFaultString()
```

Extracting the Fault Actor

Use the following method of the javax.xml.rpc.soap.SOAPFaultException class to extract the fault actor:

```java
java.lang.String getFaultActor()
```

Extracting the Fault Detail

The fault detail represents the information items specific to a Web service, consisting of an arbitrary number of fault detail items.

Use the following method of the javax.xml.rpc.soap.SOAPFaultException class to extract the fault detail:

```java
javax.xml.soap.Detail getDetail()
```

The individual detail items included in the fault detail are represented as javax.xml.soap.DetailEntry objects and can be extracted using the following method for the Detail object:

```java
java.util.Iterator getDetailEntries()
```
The individual fault detail items are retained as javax.xml.soap.DetailEntry objects in the java.util.Iterator object. Below, is a program for extracting fault detail items.

```java
import javax.xml.soap.*;
try{
    .....
}catch(SOAPFaultException  fault){
    javax.xml.namespace.Qname faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
    // Extracting the fault detail
    Detail detail = fault.getDetail();
    // Extracting the fault detail items
    Iterator it = detail.getDetailEntries();
    while(it.hasNext()){
        DetailEntry entry = (DetailEntry)it.next();
        .....
    }
}
```

The javax.xml.soap.DetailEntry object inherits javax.xml.soap.SOAPElement. Therefore, the individual detail items included in the fault detail can be operated as pure SOAPElement objects.
Chapter 6

Installing a CORBA/SOAP Gateway

This chapter describes the implementation procedure for the CORBA/SOAP gateway and precautions for linkage with STATEFUL Session Bean, STATELESS Session Bean and CORBA component models.

Figure 6-1 CORBA/SOAP Gateway Installation Overview

IDL compilation of IDL file
(Creation of Java source of gateway)

Java compilation of all created Java sources
Generation of Java Source Programs for CORBA/SOAP Gateway

The CORBA/SOAP gateway is generated by specifying an IDL definition and CORBA/SOAP gateway generation option (-soap) in the IDLc command of CORBA service.

The IDL definition has two types:

- IDL generated in development of the CORBA application
- IDL generated using the soapwsdl2idl command from WSDL (Web Services Description Language) derived during generation of a Web service

For details about the soapwsdl2idl command, refer to the Reference Manual (Command Edition).

IDLc –soap sample.idl

This generates a Java source program for the CORBA/SOAP gateway that links a CORBA application, an RPC-based client application, and Web service information. In addition a file is generated in the Java language mapping for the CORBA service that is developed using sample.idl.

The following lists Java source files for the CORBA/SOAP gateway generated in addition to the files created in the Java language mapping for the CORBA service:

Table 6-1  Files Generated from Interface Declaration

<table>
<thead>
<tr>
<th>Without –MCL or -MSV option</th>
<th>-MCL option specified</th>
<th>-MSV option specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM/_yySOAPGWstubif.java</td>
<td>None</td>
<td>MM/_yySOAPGWstubif.java</td>
</tr>
<tr>
<td>MM/_yySOAPGWStub.java</td>
<td>None</td>
<td>MM/_yySOAPGWStub.java</td>
</tr>
<tr>
<td>MM/yySOAPGWservant.java</td>
<td>MM/yySOAPGWservant.java</td>
<td>None</td>
</tr>
<tr>
<td>MM/_yySOAPGW_dd.xml</td>
<td>None</td>
<td>MM/_yySOAPGW_dd.xml</td>
</tr>
</tbody>
</table>

MM: Module name; yy: Interface name

Files generated from enumeration type

MM/ttSOAPEnumImpl.java
MM/ttSOAPEnumImplHolder.java

tt: Data type name of iDL

Files generated from structure type

MM/ttSOAPGWType.java
MM/ttSOAPGWTypeHolder.java

tt: Data type name of iDL
Files generated from exception type

MM/ttSOAPGWType.java

tt: Data type name of idL

Note
Use the Interstage Management Console to generate the CORBA/SOAP server gateway that links with EJB applications:

Select [System] > [WorkUnits] > [MyIJServer] (IJServer WorkUnit name) > [Deployment] tab, and [EJB Application Settings] to generate a CORBA/SOAP server gateway.

IDL definition using Factory interface

To generate the CORBA/SOAP gateway from an IDL definition for a CORBA application that uses the Factory interface, specify:

- an IDL definition
- CORBA/SOAP gateway generation option (-soap), and
- CORBA/SOAP gateway generation option

For the Factory interface (-factory) in the IDLc command for the CORBA service:

IDLc –soap –factory samplefactory.idl

The files generated are same as those created when specifying a standard CORBA/SOAP gateway generation, except for the following:

- The CORBA/SOAP server gateway is for the Factory interface only.
- The CORBA/SOAP client gateway is not generated.
- Web service information files are only those corresponding to the Servant interface.
Generation of Java Class Files for CORBA/SOAP Gateway

Use the `javac` command to perform Java compilation on all the Java source files generated using the `IDLc` command:

**Windows**

```
javac -d .\MM\*.java
```

**Solaris OE  Linux**

```
javac -d ./MM/*.java
```

**Note**
The Java class file generated is required for generating an RPC-based client application that links with the CORBA/SOAP server gateway.
Precautions on Linkage with CORBA Application

This section describes the precautions of generating an RPC-based client application that links with a CORBA application.

Association of Conversion Mapping for Conversion from Java Values to XML Values and Vice Versa (CORBA)

An RPC-based client application (that links with a CORBA server) performs association of the conversion mapping of an IDL-defined enumeration type, structure type and exception type.

Note:
The CORBA server application includes a component/transaction server application. Also, the XML data type structure is mapped to the XML and CORBA Data Types.

- Name space name: urn:Fujitsu-Soap-Service-Data
- Name: module name-interface name-IDL type name

```java
module ODdemo{
    interface calculator{
        exception ZEROPARAM {};
        enum CALCKIND {ADD, SUB, MULTI, DEVIDE};
        struct result {
            long add_result;
            long subtract_result;
            long multiple_result;
            float divide_result;
        };
        result calculate( in long a, in long b, in CALCKIND c )
        raises( ZEROPARAM );
    };
}
```

Association of the IDL-defined structure type and exception type mapping is shown below.

```java
// import statement declaration
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;
:
Service service = null;
try {
    service = ServiceFactory.newInstance().createService(new QName("",""));
} catch( javax.xml.rpc.ServiceException e ){
    :
}
```

Chapter 6: Installing a CORBA/SOAP Gateway

Obtaining CORBA Exception from Fault Details Item

A CORBA exception reported by a CORBA server application (including a component/transaction server application) has information set in the predetermined Fault details item.

Name space name: com.fujitsu.soap.gateway.CORBAException

<table>
<thead>
<tr>
<th>Table 6-2  Fault Details Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local name</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td>major</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>minor</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Local name | Local name | Description
--- | --- | ---
status | int | Status of the method that reported the CORBA system exception is set. Valid only in case the CORBA system exception is notified.
org.omg.CORBA..CompletionStatus._COMPLETED_MAYBE: Completion status unknown
org.omg.CORBA..CompletionStatus._COMPLETED_YES: Method complete
org.omg.CORBA..CompletionStatus._COMPLETED_NO: Method incomplete
message | String | Detailed information of the CORBA system exception is set. Valid only in case CORBA system exception is notified.

For a CORBA user-defined exception, the exception class reported by the CORBA server application is set in the Fault details item. Specify the following to obtain the Fault details item and then obtain CORBA used-defined exception:

- Name space name: urn:Fujitsu-Soap-Service-Data
- Local name: IDL-defined exception type name
- An RPC-based client application that links with a CORBA server application requires the above processing for obtaining the Fault details items.
- In a CORBA used-defined exception, conversion mapping must be developed before calling a method.

**Example**

The following code shows how to obtain Fault details items when the add method of a CORBA server application (generated from the IDL definition below) is called.

In this example, java.lang.Class.forName() is used to obtain java.lang.Class from the exception class name, where the CORBA exception class is set in Fault details item: id to generate the instance of the CORBA system exception.

```java
module ODDemo{
    interface calculator{
        exception ZEROPARAM {};
        struct result {
            long add_result;
            long subtract_result;
            long multiple_result;
            float divide_result;
        };
        result calculate( in long a, in long b ) raises( ZEROPARAM );
    };
}
```
// import statement declaration
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.soap.SOAPFaultException;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;
import javax.xml.soap.Detail;
import javax.xml.namespace.QName;
import java.util.Iterator;
import javax.xml.soap.SOAPElement;

try {
    Service service = null;
    service = ServiceFactory.newInstance().createService(new QName("",""));
    TypeMappingRegistry tmr = service.getTypeMappingRegistry();
    TypeMapping tm = tmr.createTypeMapping();
    // user-defined exception generate QNAME
    QName _qname =
        new QName("urn:Fujitsu-Soap-Service-Data", "ODdemo-calculator-ZEROPARAM");
    // user-defined exception Register ZEROPARAM mapping
tm.register(ODdemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname,
        new com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory(
            ODDemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname),
        new com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory(
            ODDemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname));
    // user-defined exception Register ZEROPARAM mapping
tmr.register("http://schemas.xmlsoap.org/soap/encoding/", tm );

    //method calling
    ..... = call.invoke(...);
}

} catch( javax.xml.rpc.ServiceException e ) {

}

} catch( SOAPFaultException e ) {

} //obtain Fault information
QName _faultCode = e.getFaultCode();
String _faultString = e.getFaultString();
String _faultActor = e.getFaultActor();
Detail _detail = e.getDetail();
if( _detail != null ){
    //obtain Fault details item
    String _ex_id = null;
    int _ex_major = 0;
    int _ex_minor = 0;
    int _ex_status = org.omg.CORBA.CompletionStatus._COMPLETED_MAYBE;
    String _ex_message = null;

    //Faults detail item for CORBA exception generate QNAME
    String NS_URI_CORBAException = "com.fujitsu.interstage.soap.gateway.CORBAException";
    QName Q_CORBAEXCEPTION_ID = new QName( NS_URI_CORBAException, "id" );
    QName Q_CORBAEXCEPTION_MAJOR =
new QName( NS_URI_CORBAException, "major" );
QName Q_CORBAEXCEPTION_MINOR =
new QName( NS_URI_CORBAException, "minor" );
QName Q_CORBAEXCEPTION_STATUS =
new QName( NS_URI_CORBAException, "status" );
QName Q_CORBAEXCEPTION_MESSAGE =
new QName( NS_URI_CORBAException, "message" );

Iterator _it = _detail.getChildElements();
if( _it != null ) {
    while(_it.hasNext()) {
        SOAPElement _elm = (SOAPElement)_it.next();
        String _elmName = _elm.getElementName().getLocalName();
        if( _elmName.equals("id") ) {
            _ex_id = _elm.getValue();
        } else if( _elmName.equals("major") ){
            _ex_major = new Integer(_elm.getValue()).intValue();
        } else if( _elmName.equals("minor") ) {
            _ex_minor = new Integer(_elm.getValue() ).intValue();
        } else if( _elmName.equals("status") ) {
            _ex_status = new Integer(_elm.getValue() ).intValue();
        } else if( _elmName.equals("message") ) {
            _ex_message = _elm.getValue();
        }
    } // while
    //generate CORBA exception
    if( _ex_id != null ) {
        java.lang.Object _ex_obj = null;
        try {
            java.lang.Class _ex_class = java.lang.Class.forName( _ex_id );
            _ex_obj = _ex_class.newInstance();
        } catch( java.lang.ClassNotFoundException nie ) {
            //describe exception processing assumed when a class indicated by
            _ex_id is not found
        } catch( java.lang.InstantiationException nie ) {
            // describe exception processing assumed when an attempt failed to
generate the instance of ex class
        } catch( java.lang.IllegalAccessException nie ) {
            // describe exception processing assumed when access to _ex class or
constructor of _ex class failed
        } catch( java.lang.SystemException nie ) {
            throw (org.omg.CORBA.SystemException)_ex_obj;
        } // SystemException
    }
} // SystemException
Chapter 6: Installing a CORBA/SOAP Gateway

The following code shows how to obtain a user-defined exception class (ZEROPARAM) from the Fault details item. Two methods can be used:

Method 1: Obtaining each piece of member information of an exception class and developing a user-defined exception class; and

Method 2: Using an extended API provided by the Interstage SOAP service to obtain a user-defined exception class.

If more than one member is defined in a user-defined exception class, Method 2 simplifies the process.

- Name space name of user-defined exception: urn-Fujitsu-Soap-Service-Data
- Local name of user-defined exception: ODdemo-calculator-ZEROPARAM
- Java class name of user-defined exception: ODdemo-calculatorPackage.ZEROPARAMSOAPGWType

```java
// import statement declaration
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.soap.SOAPFaultException;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;
import javax.xml.soap.Detail;
import javax.xml.soap.DetailEntry;
import javax.xml.namespace.QName;
import java.util.Iterator;
import javax.xml.soap.SOAPElement;

try {
    Service service = null;
    service = ServiceFactory.newInstance().createService(new QName("",""));
    TypeMappingRegistry tmr = service.getTypeMappingRegistry();
    TypeMapping tm = tmr.createTypeMapping();
    // user-defined exception     generate QNAME
    QName _qname =
        new QName( "urn:Fujitsu-Soap-Service-Data", "ODdemo-calculator-
                   ZEROPARAM" );
    // user-defined exception     Register ZEROPARAM mapping
tm.register(ODdemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname, new com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory(ODdemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname),
            new com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory(ODdemo.calculatorPackage.ZEROPARAMSOAPGWType.class, _qname ) );
    // user-defined exception     Register ZEROPARAM mapping
tmr.register( "http://schemas.xmlsoap.org/soap/encoding/", tm );
    
    //method calling
    ..... = call.invoke(...);

} catch( javax.xml.rpc.ServiceException e ) {
    
```
} catch( SOAPFaultException e ) {
    //obtain Fault information
    QName _faultCode = e.getFaultCode();
    String _faultString = e.getFaultString();
    String _faultActor = e.getFaultActor();
    Detail _detail = e.getDetail();
    if( _detail != null ){
        //obtain Fault details item
        Iterator _it = _detail.getChildElements();
        if(_it != null ) {
            while(_it.hasNext()) {
                SOAPElement _elm = (SOAPElement)_it.next();
                String _elmName = _elm.getName().getLocalName();
                if( _elmName.equals("ODdemo-calculator-ZEROPARAM") ) {
                    //method for obtaining each piece of member information of an exception class and //developing a user-defined exception class
                    ODdemo.calculatorPackage.ZEROPARAMSOAPGWType excep;
                    //generate user-defined exception class
                    excep = new ODdemo.calculatorPackage.ZEROPARAMSOAPGWType();
                    _it = _elm.getChildElements();
                    //obtain member information of user-defined exception class
                    //obtaining member information by the member name defined in user-defined exception class
                    // modifying getValue() to suit member type in order to return String type
                    while(_it.hasNext()) {
                        _elm = (SOAPElement)_it.next();
                        _elmName = _elm.getName().getLocalName();
                        if( _elmName.equals("ODdemo-calculator-ZEROPARAM") ) {
                            //setting when member name of exception class is String type
                            excep.MemberNameOfExceptionClas = _elm.getValue();
                        } else if( _elmName.equals("ODdemo-calculator-ZEROPARAM") ) {
                            //setting when member name of exception class is int type
                            excep.MemberNameOfExceptionClas = new Integer((String)_elm.getValue()).intValue();
                        }
                    } // while
                    //throw exception class
                    throw excep;
                } else if( _elmName.equals("ODdemo-calculator-ZEROPARAM") ) {
                    //obtain exception class
                    try{
                        excep = ((com.fujitsu.interstage.soapx.message.SOAPElementImpl)_elm).getValueAsType(ExcepQName);
                    }catch( java.lang.Exception ee ){
                        // describe exception processing assumed
                    } //throw exception class
                    throw (ODdemo.calculatorPackage.ZEROPARAMSOAPGWType)excep;
                } //if
            } //while
        } //if
    } //if
RPC-based Client Application that Links with Factory Interface

A CORBA server application using the Factory interface has two interfaces:
- The Factory interface that generates a server object (Servant interface), and
- The Servant interface generated from the Factory interface.

To generate an RPC-based client application that links with the Factory interface, CORBA server application operations must be called in the order shown below.

Figure 6-2 Order of CORBA Server Application Operations

The following example shows an RPC-based client application that links with a CORBA server application (generated from the IDL definition shown below).

Example

```idl
module sample{
  interface demoservant {
    struct result {
      long add_result;
      long subtract_result;
    };
  }
```
result calculate( in long a, in long b );
void remove();
);
interface demofactory {
  demoservant create ();
};
}

Obtaining URI of Servant Interface

Calls an operation used for generating the Servant interface defined in the Factory interface. This is used to obtain the URI of the Servant interface.

Example

Calls the create operation of the demofactory interface to obtain the URI of the demoservant interface. Store the obtained URI in the interval variable: nskey.

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;
:
String nskey = null;
try {
  :
  //generate Service object
  Service service = ServiceFactory.newInstance().createService(new QName("",""));
  //generate Call object
  Call call = service.createCall();
  call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
  call.setOperationName( new QName("sample-countRequest", "create" ));
  call.setReturnType(XMLType.XSD_STRING);
  Object[] param = new Object[0];
  String tmp1 = (String)call.invoke( param );
  //obtain URI of demoservant interface
  nskey = call.getOperationName().getNamespaceURI();
  :
} catch( javax.xml.rpc.ServiceException e ) {
  :
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
  :
} catch( javax.xml.rpc.JAXRPCException e ) {
  :
} catch( java.rmi.RemoteException e ) {
  :
} catch( java.lang.Exception e ) {
  :
}
```
Calling the Remote Method of Servant Interface

Use the URI of the Servant interface (obtained by calling the operation for generating the Servant interface of the Factory interface) to call the remote method for the Servant interface. This may be repeated until the Servant interface is released.

Example

Use the URI of the demoservant interface (set in the internal variable: nskey) to call the calculate method for the demoservant interface.

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

try {
    //generate Call object
    Call call = service.createCall();

    call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
    call.setOperationName( new QName(nskey, "calculate" ));
    //register parameter information
    call.addParameter( "a", XMLType.XSD_INT, ParameterMode.IN );
    call.addParameter( "b", XMLType.XSD_INT, ParameterMode.IN );
    call.setReturnType( new QName("urn:Fujitsu-Soap-Service-Data", "sample-demoservant-result") );
    //generate parameter
    Object[] param = new Object[]{new java.lang.Integer(1), new java.lang.Integer(2)};
    //call Web service
    sample.demoservantPackage.resultSOAPGWType result = (sample.demoservantPackage.resultSOAPGWType)call.invoke( param );
} catch( javax.xml.rpc.ServiceException e ) {
    :
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
    :
} catch( javax.xml.rpc.JAXRPCException e ) {
    :
} catch( java.rmi.RemoteException e ) {
    :
} catch( java.lang.Exception e ) {
    :
} :
```
Servant Interface Release Processing

When all remote methods for the Servant interface have been called, release the generated Servant interface. Do this by calling the release method defined in the Servant interface.

Example

Use the URI of the Servant interface (set in the internal variable: nskey) to call the remove method as the release method for the demoservant interface.

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;

try {
    // generate Call object
    Call call = service.createCall();
    call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
    call.setOperationName( new QName(nskey, "remove"));
    // generate Call object
    Object[] param = new Object[0];
    // generate Call object
    call.invoke( param );
} catch( javax.xml.rpc.ServiceException e ) {
    
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
    
} catch( javax.xml.rpc.JAXRPCException e ) {
    
} catch( java.rmi.RemoteException e ) {
    
} catch( java.lang.Exception e ) {
    
}
```

IDL Definition Available to Factory Interface linkage

When a CORBA application that uses the Factory interface linking with an RPC-based client application is used, be aware of the following precautions regarding the IDL definition:

- Define the Factory interface having a method for generating an object reference and the Servant interface generated by the Factory interface.
- `<Factory interface>`
  - As a method for generating an object reference, define a single method without parameters whose return value is user-defined interface type (Servant interface type).
  - The name of the method for generating an object reference is "createXXXX." "XXXX" is an arbitrary character that is available in IDL definition.
  - Avoid using the other interfaces.
• <Servant interface>
  – As a method for releasing the Servant interface, define a single method without return value or parameters.
  – The name of the method used for releasing the Servant interface is "remove" or "destroy."

**Linkage with CORBA Server Application that uses TypeCode Type**

Before executing a remote procedure call in an RPC-based client application, the following processing is required to remotely call a method that uses the TypeCode type of CORBA as a parameter or a return value:

• Generation of the com.fujitsu.interstage.soap.util.TypeCode class and setting of transfer data.

**Example**

The following shows the coding example assumed when the op method of a CORBA server application is generated from the IDL definition below:

```java
module ODsample {
  interface TypeCodetest {
    TypeCode op( in TypeCode arg1 );
  }
};
```

• Instance variable of org.omg.CORBA.TypeCode class: corba_typeCode1

• Name space name of TypeCode class group: urn:xmlns-fujitsu-com:soap-corba-type2002-02

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;

try {
  //generate Call object
  Call _call = service.createCall();
  //setting of return value (generate QName of TypeCode)
  String _ns = "urn:xmlns-fujitsu-com:soap-corba-type2002-02";
  QName _qName = new QName( _ns , "TypeCode" );
  _call.setReturnType( _qName );
  //Construction of parameter
  com.fujitsu.interstage.soap.util.TypeCode _p1 = 
      new com.fujitsu.interstage.soap.util.TypeCode( corba_typeCode1 );
  _call.addParameter( "arg1" , _qName , ParameterMode.IN );
  java.lang.Object[] _iparams = new java.lang.Object[]{ _p1 };
  //call method and obtain return value
  com.fujitsu.interstage.soap.util.TypeCode _res =
      ( com.fujitsu.interstage.soap.util.TypeCode )_call.invoke( _iparams );
```
org.omg.CORBA.TypeCode _ret = _res.getCORBAType();
}
catch( javax.xml.rpc.ServiceException e ) {
:
}
catch( javax.xml.rpc.soap.SOAPFaultException e ) {
:
}
catch( javax.xml.rpc.JAXRPCException e ) {
:
}
catch( java.rmi.RemoteException e ) {
:
}
catch( java.lang.Exception e ) {
:
:

**Linkage with CORBA Server Application that Uses any Type**

Before executing a remote procedure call in an RPC-based client application, to remotely call a method (that uses the any type as a parameter or a return value), the following processing is required:

- Generation of the com.fujitsu.interstage.soap.util.Any class and setting of transfer data

**Example**

The following shows a coding example when the op method of a CORBA server application is generated from the IDL definition below:

```java
module ODsample {
    interface anytest {
        any op( in any arg1 );
    };
};
```

- Instance variable of org.omg.CORBA.Any class: corba_any1
- Name space name of Any/TypeCode class group: urn:xmlns-fujitsu-com:soap-corba-type2002-02

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;

try {
    Call _call = service.createCall();
    // AnyQName
    String _ns = "urn:xmlns-fujitsu-com:soap-corba-type2002-02";
    QName _qName = new QName( _ns , "Any" );
    _call.setReturnType( _qName );
```
com.fujitsu.interstage.soap.util.Any _p1 =
    new com.fujitsu.interstage.soap.util.Any( corba_any1 );
    _call.addParameter( "arg1" , _qName , ParameterMode.IN );
    java.lang.Object[] _iparams = new java.lang.Object[]{ _p1 };

    //call method and obtain return value
    com.fujitsu.interstage.soap.util.Any _res =
        ( com.fujitsu.interstage.soap.util.Any )_call.invoke( _iparams );
    org.omg.CORBA.Any _ret = _res.getCORBAType();

} catch( javax.xml.rpc.ServiceException e ){
    :
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
    :
} catch( javax.xml.rpc.JAXRPCException e ) {
    :
} catch( java.rmi.RemoteException e ) {
    :
} catch( java.lang.Exception e ) {
    :
}:

**Other Precautions**

The implementation repository ID for the CORBA/SOAP client gateway is as follows. Use the ID for development of connection information (such as object reference) as required.

IDL:com/fujitsu/interstage/soap/gateway/client:1.0
Precautions on Linkage with EJB Application

This section describes precautions for generating an RPC-based client application that links with a CORBA application.

Provision of Parameters to be Sent to Method of EJB Server Application

Change of Parameter Name in EJB Server Application

Parameters of a method defined in the remote interface definition of an EJB server application are renamed "arg1, arg2,..." in the order they are defined.

Thus, "arg1, arg2,..." must be specified to the parameter names to be specified in generating send parameters.

Example

In this case the remote interface having the following method is defined:

```java
package pack1;
import java.rmi.*;
import javax.ejb.*;

public class calcBean implements javax.ejb.SessionBean {
    public int add(int  a,  int  b) {
        ...
    }
}
```

The parameter name is changed as shown below in a Web service information file:

<table>
<thead>
<tr>
<th>Definition on remote interface</th>
<th>Definition on Web service information</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int add (int a, int b)</td>
<td>int add (int arg1, int arg2)</td>
</tr>
</tbody>
</table>

Generation of send parameters is as follows:

```java
// import statement declaration
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

//generate Call object
```
Chapter 6: Installing a CORBA/SOAP Gateway

```java
Call call = service.createCall();
Integer arg1 = new Integer(10);
Integer arg2 = new Integer(20);
call.addParameter("arg1", XMLType.XSD_INT, ParameterMode.IN);
call.addParameter("arg2", XMLType.XSD_INT, ParameterMode.IN);
java.lang.Object[] _iparamsInt = new Object[] { arg1, arg2 };
```

**Change of Parameters where Methods of the Same Name as Defined**

In the remote interface of an EJB server application, if methods of the same name are found, the end of the parameter names are changed to serial numbers. Also, the parameter names of the methods having the same name are changed as shown below:

**Example**

If a remote interface having methods of the same name is defined:

```java
package pack1;
import java.rmi.*;
import javax.ejb.*;
public class calcBean implements javax.ejb.SessionBean {
    public int add(int arg0, int arg1) {
    }
    public float add(float arg0, float arg1) {
    }
}
```

The parameter name is changed as shown below in a Web service information file:

<table>
<thead>
<tr>
<th>Definition on remote interface</th>
<th>Definition on Web service information</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int add (int arg0, int arg1)</td>
<td>int add (int arg1, int arg2)</td>
</tr>
<tr>
<td>public float add (float arg0, float arg1)</td>
<td>float add (float arg3, float arg4)</td>
</tr>
</tbody>
</table>

**Generation of send parameters is as follows:**

```java
// import statement declaration
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

//generate Call object
```
Call callint = service.createCall();
Integer intArg1 = new Integer( 10 );
Integer intArg2 = new Integer( 20 );
callint.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN );
callint.addParameter( "arg2", XMLType.XSD_INT, ParameterMode.IN );
callint.setReturnType(XMLType.XSD_INT);
java.lang.Object[] _iparamsInt = new Object[] { intArg1 , intArg2 };

Call callfloat = service.createCall();
Float floatArg1 = new Float( 2.1 );
Float floatArg2 = new Float( 2.0 );
callfloat.addParameter( "arg3", XMLType.XSD_FLOAT, ParameterMode.IN );
callfloat.addParameter( "arg4", XMLType.XSD_FLOAT, ParameterMode.IN );
callfloat.setReturnType(XMLType.XSD_FLOAT);
java.lang.Object[] _iparamsFloat = new Object[] { floatArg1 , floatArg2 };

Association of Conversion Mapping for Conversion from Java Values to XML Values and Vice Versa (EJB)

An RPC-based client application (that links with an EJB server) performs association of the conversion mapping of an IDL-defined enumeration type, structure type and exception type.

Note: The CORBA server application includes a component/transaction server application. Also, the XML data type structure is mapped to the XML and EJB Data Types.

- Name space name: urn:Fujitsu-Soap-Service-Data
- Name: package name--Java class name SOAPGWTy pe ("a-b-cSOAPGWType" if the Java class name is "a.b.c")

package fujitsu.sample; // package name
public class Human
    implements java.io.Serializable {
    
};

Association of the mapping of the Java class defined in the remote interface definition is shown below.

// import statement declaration
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;

try {
    service = ServiceFactory.newInstance().createService(new QName("","");
} catch( javax.xml.rpc.ServiceException e ) {
    
} 

service = ServiceFactory.newInstance().createService(null);
TypeMappingRegistry tmr = service.getTypeMappingRegistry();
TypeMapping tm = tmr.createTypeMapping();
// structure register mapping of Human
QName _qname =
    new QName( "urn:Fujitsu-Soap-Service-Data", "fujitsu-sample-
HumanSOAPGWType" );
tm.register( fujitsu-sample-HumanSOAPGWType.class, _qname,
    new com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory( fujitsu.sample.HumanSOAPGWType.class, _qname ),
    new com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory( fujitsu.sample.HumanSOAPGWType.class, _qname ) );
tmr.register( "http://schemas.xmlsoap.org/soap/encoding/", tm );

RPC-based Client Application that Links With STATEFUL SessionBean

An EJB application has two interfaces: a Home interface for generating EJB objects, and a Remote interface for calling user's business methods.

To generate an RPC-based client application that links with STATEFUL SessionBean, it is necessary to call methods of an EJB server application for STATEFUL SessionBean in the order shown below.

- Initialization of application
- Obtaining URI of Remote interface
- Application-inherent processing
- Calling the remote method of Remote interface
- Application-inherent processing
- Remote interface release processing

Figure 6-3 Order of Calling Methods of EJB Server Application (for STATEFUL SessionBean)
The following example shows an RPC-based client application that links with an EJB application (generated from the Home/Remote interface definition below):

Example

```
[HOME interface]
package pack1;

import java.rmi.*;
import javax.ejb.*;

public interface calcBeanHome extends javax.ejb.EJBHome {
    public calcBeanRemote create() throws java.rmi.RemoteException,
    javax.ejb.CreateException;
}

[REMOTE interface]
package pack1;

import java.rmi.*;
import javax.ejb.*;

public interface calcBeanRemote extends javax.ejb.EJBObject {
    public int addIntegers(int val1, int val2) throws 
    java.rmi.RemoteException;
}
```

Obtaining URI of Remote Interface

To obtain the URI of the Remote interface, call an operation for generating the Remote interface defined in the HOME interface. This processing can be omitted.

Example

To obtain the URI of the calcBeanRemote interface, call the create operation of the calcBeanHome interface. Store the obtained URI in the interval variable: nskey.

```
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

String nskey = null;
try {
    // import statement declaration
    Service service = ServiceFactory.newInstance().createService(new
    QName("",""));
    // import statement declaration
    Call call = service.createCall();
    call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
    call.setOperationName( new QName(" calcBean", "create" ));
    call.setReturnType(XMLType.XSD_STRING);
    Object[] param = new Object[0];
    String tmp1 = (String)call.invoke( param );
```
Calling the Remote Method of Remote Interface

Use the URI of the Remote interface (obtained by calling the operation for generating the Remote interface of the Home interface) to call the remote method of Remote interface. This may be repeated until the Remote interface is released.

If the processing for obtaining the URI of the Remote interface is omitted, after calling the remote method, obtain the URI to be used in the next call from the operation name information.

Example

If the URI of the Remote interface has been obtained:

To call the addIntegers method for the calcBeanRemote interface, use the URI of the calcBeanRemote interface set in the internal variable: nskey.

```
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

try {
    //generate Call object
    Call call = service.createCall();

call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
call.setOperationName( new QName(nskey, "addIntegers") );
//register parameter information
call.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN );
call.addParameter( "arg2", XMLType.XSD_INT, ParameterMode.IN );
call.setReturnType(XMLType.XSD_INT);
//generate parameter
Object[] param = new Object[] { new java.lang.Integer(1), new java.lang.Integer(2) };
//call Web service
    Integer result_tmp = (Integer)call.invoke( param );
    int result = result_tmp.intValue();
}
```
Precautions on Linkage with EJB Application

If the URI of the Remote interface has not been obtained:

To obtain the URI to be used in the next call, Use calcBean to call the addIntegers method of the calcBeanRemote interface.

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.XMLType;

String nskey = "calcBean"; //generate URI
try {
    //generate Call object
    Call call = service.createCall();
    call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
    call.setOperationName( new QName(nskey, "addIntegers") );
    //register parameter information
    call.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN );
    call.addParameter( "arg2", XMLType.XSD_INT, ParameterMode.IN );
    call.setReturnType(XMLType.XSD_INT);
    //generate parameter
    Object[] param = new Object[] { new java.lang.Integer(1), new java.lang.Integer(2) };
    //Call Web service
    Integer result_tmp = (Integer)call.invoke( param );
    nskey = call.getOperationName().getNamespaceURI(); //obtain URI to be used in the next call
    int result = result_tmp.intValue();
} catch( javax.xml.rpc.ServiceException e ) {
    :
} catch( javax.xml.rpc.soap.SOAPFaultException e ) {
    :
} catch( javax.xml.rpc.JAXRPCException e ) {
    :
} catch( java.rmi.RemoteException e ) {
    :
} catch( java.lang.Exception e ) {
    :
}
```
Remote Interface Release Processing

When all remote methods for the Remote interface have been called, release the Remote interface generated. To release the Remote interface, call the release method defined in the Remote interface.

Example

To call the remove method as the release method of the calcBeanRemote interface, use the URI of the calcBeanRemote interface set in the internal variable: nskey.

```java
// import statement declaration
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;

try {
    //generate Call object
    Call call = service.createCall();
    call.setTargetEndpointAddress("http://interstage/soap/servlet/WSContainer");
    call.setOperationName( new QName(nskey, "remove");
    //generate parameter
    Object[] param = new Object[0];
    //call Web service
    call.invoke( param );
} catch( javax.xml.rpc.ServiceException e ) {} catch( javax.xml.rpc.soap.SOAPFaultException e ) {} catch( javax.xml.rpc.JAXRPCException e ) {} catch( java.rmi.RemoteException e ) {} catch( java.lang.Exception e ) {};
```
Obtaining the Exception Class from Fault Details Item

An exception reported by an EJB server application has information set in the Fault details item. To obtain the Fault details item and obtain the exception class information, specify the following.

- Name space name: urn:Fujitsu-Soap-Service-Data
- Local name: package name--Java class name SOAPGWTy pe ("a-b-cSOAPGWTy pe" if the Java class name is "a.b.c")
- An RPC-based client application that links with an EJB server application requires the above processing for obtaining the Fault details items.
- In the exception class, conversion mapping must be developed before calling a method.

Example

The following code shows how to obtain Fault details items when the remote interface shown below is defined and the sub method is defined to throw the exception class pack1.SubException. Two methods can be used:

Method 1: Obtaining each piece of member information of an exception class and developing an exception class; and

Method2: Using an extended API provided by the Interstage SOAP service to obtain an exception class.

To define more than one member in an exception class, using method (2) simplifies the process.

```java
//remote interface definition
package pack1;
import java.rmi.*;
import javax.ejb.*;
public class calcBean implements javax.ejb.SessionBean {
    public int sub(int a, int b) throws pack1.SubException {
    }
}

//exception class definition
public class SubException extends java.lang.Exception {
    public String detailMessage;
    public int detailCode;
    public SubException() { }
}
```
• Name space name of exception class: urn:Fujitsu-Soap-Service-Data
• Local name of exception class: pack1-SubExceptionSOAPGWType
• Java class name of exception class: pack1.SubException

```java
// import statement declaration
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.soap.SOAPFaultException;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;
import javax.xml.soap.Detail;
import javax.xml.soap.DetailEntry;
import javax.xml.namespace.QName;
import java.util.Iterator;
import javax.xml.soap.SOAPElement;

try {
    Service service = null;
    service = ServiceFactory.newInstance().createService(new QName("",""));
    TypeMappingRegistry tmr = service.getTypeMappingRegistry();
    TypeMapping tm = tmr.createTypeMapping();
    //generate exception class, QName
    QName _qname = new QName("urn:Fujitsu-Soap-Service-Data","pack1-SubExceptionSOAPGWType");
    //register parameter information
    tm.register(pack1.SubException.class, _qname,
            new com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory(
            pack1.SubException.class, _qname),
            new com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory(
            pack1.SubException.class, _qname ) );

    // register mapping information to mapping registry
    tmr.register( "http://schemas.xmlsoap.org/soap/encoding/", tm );

    //method calling
    ..... = call.invoke( ... );
}
} catch( javax.xml.rpc.ServiceException e ) { }
} catch( SOAPFaultException e ) {
    //obtain Fault information
    QName _faultCode = e.getFaultCode();
    String _faultString = e.getFaultString();
    String _faultActor = e.getFaultActor();
    Detail _detail = e.getDetail();
    if( _detail != null ){
        //obtain Fault details item
        String _detailMessage = null;
        int _detailCode = 0;
        Iterator _it = _detail.getChildElements();
        while(_it.hasNext()) {
```
SOAPElement _elm = (SOAPElement)_it.next();
String _elmName = _elm.getElementName().getLocalName();
//decision of exception class information
if( _elmName.equals("pack1-SubExceptionSOAPGWTy")) {
    //method for obtaining each piece of member information of exception
class to develop exception class
    pack1.SubException excep;
    _it = _elm.getChildElements();
    //obtain member information of exception class
    //obtain member information by the member name defined in exception
class
    while(_it.hasNext()) {
        _elm = (SOAPElement)_it.next();
        _elmName = _elm.getElementName().getLocalName();
        if( _elmName.equals("detailMessage")) {
            _detailMessage = _elm.getValue();
        } else if( _elmName.equals("detailCode")) {
            _detailCode = (new Integer((String)_elm.getValue())).intValue();
        }
    }
    //generate exception class
    excep = new pack1.SubException();
    //set exception class member information
    excep.detailMessage = _detailMessage;
    excep.detailCode = _detailCode;
    //throw exception class
    throw excep;
}
//method for using extended API provided by Interstage SOAP service to
obtain exception class
QName ExcepQName = new QName( "urn:Fujitsu-Soap-Service-Data", "pack1-
SubExceptionSOAPGWTy" );
java.lang.Object excep;
//obtain exception class
try {
    excep = ((com.fujitsu.interstage.soapx.message.SOAPElementImpl)_elm)
        .getValueAsType(ExcepQName);
} catch( java.lang.Exception ee ) {
    // describe exception processing assumed
}
//throw exception class
throw (pack1.SubException)excep;
}
Chapter 7

How to use the Reliable Messaging Function

The Reliable messaging function supports two types of models:

- PUSH model where a server system receives SOAP messages
- PULL model where a client system receives SOAP messages.

This chapter describes how to implement each model.

**Note**
Session management is not available in the reliable messaging function.

To find details about how use a proxy in the reliable messaging function, refer to Chapter 3 Configuring an Environment. To perform basic authentication, refer to Authentication of Reliable Messaging Function.
Agreement between Client System and Server System

To use the Reliable messaging function, first determine the model to be used (PUSH or PULL). Then, determine the necessary agreements (agreed-upon items) necessary for the client system and server system to use Reliable messaging.

For details about deployment of receiver application and deployment of sender application in the following description, refer to Chapter 16 How to Use Reliable Messaging Function for Web Services (SOAP) in the Security System Guide.

Message Type ID

Message type ID is used as an identifier to identify the type of a SOAP message. The type of a SOAP message is used to identify a receiver application for receiving SOAP messages. It can be set independently of the format of a SOAP message.

The syntax of the message type ID must conform to NMTOKEN of XML, and must contain characters usable as a file name and directory name. Message type ID can be set independently of a machine name or IP address.

The same message type ID must be used to set a sender application (for performing transmission) and a receiver application (for performing reception) of SOAP messages of the same type.

- At the deployment time of the sender application, the message type ID of a SOAP message must be specified under Deployment of Sender application form. From the sender application, a SOAP message is sent with the message type ID specified in the argument of the JAXM API method.
- At the deployment time of the receiver application, the class name and message type ID of a receiver application must be specified under Deployment of Receiver application.
Sender ID, Receiver ID

Sender ID and Receiver ID used as identifiers to identify the sender/receiver server system or client system of a SOAP message.

- Sender ID refers to sending the client ID of a PUSH model or sending the server ID of a PULL model.
- Receiver ID refers to receiving the client ID of a PULL model or receiving server ID of a PUSH model.

The syntax of both Sender ID and Receiver ID must conform to NMTOKEN of XML and contain characters usable as a file name and directory name. The sender/receiver ID can be set independently of a machine name or IP address.

The sender/receiver ID must be agreed upon between the sender and the receiver of a same SOAP message type. Both the sender and the receiver must set the Sender ID and Receiver ID that have been agreed upon in setting the Web service information of a sender application and a receiver application.

- The sending party of a SOAP message sets the Sender ID and Receiver ID that have been agreed on in a single message type ID under Deployment of Sender application. From the sender application, a SOAP message is sent with the receiver ID specified as the destination of the SOAP message in the JAXM API method.
- The receiving party of a SOAP message provides a receiver application and sets the agreed-upon Sender ID and Receiver ID as well as the class name of the receiver application under Deployment of Receiver application.

Sender IDs or Receiver IDs must be unique in a single server system or client system. That is:

- The same ID must be used for the receiver server ID of all PUSH models and the sender server ID of all PULL models set on a single server system.
- The same ID must be used for the sender client ID of all PUSH models and the Receiver client ID of all PULL models set on a single client system.
URL of Receiver Server/Sender Server

The URL of a receiver server or sender server is determined after the Reliable messaging function is set on the receiver server of the PUSH model or sender server of the PULL model.

For setting the Reliable messaging function of a server, refer to Chapter 16 How to use Reliable Messaging function for Web Services(SOAP) in the Security System Guide.

For details of how to determine URL, refer to Chapter 9 Managing Web Service Information

The URL of a Web service container is specified when the Reliable messaging function is set on a client system (which can be a sender client of the PUSH model, or a receiver client of the PULL model.)

For this operation, specify the URLs of a receiver server and a sender server to perform communication of SOAP messages for the same message type.

SOAP Signature Verification ID

To use a SOAP digital signature for preventing repudiation of sending or receiving a message, specify a SOAP signature verification ID for each party (sender and receiver) of a SOAP message. If the SOAP signature verification ID is not specified, only the reliable transmission of the SOAP message is supported. Repudiation of sending or receiving a message cannot be prevented.

Specify the alias of a public key (obtained from the communications partner of a SOAP message in the SOAP signature verification ID.) For a description of the preparation of a key pair and required public-key, refer to the Security System Guide.

Specify the SOAP signature verification ID (alias of public-key) for each communications partner of a SOAP message. The ID must be unique within a single server system or client system.
PUSH Model where Server System Receives Messages

In the PUSH model, an application on the client system sends a SOAP message to an application on the server system.

Agreement in PUSH Model

The following table contains items that must be agreed upon in advance, to set up applications on the client system and server system of the PUSH model.

### Table 7-1 Items to be Agreed on in Advance

<table>
<thead>
<tr>
<th>Item set by PUSH model sending party (client system)</th>
<th>Item set by PUSH model receiving party (server system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message type ID</td>
<td>Message type ID</td>
</tr>
<tr>
<td>Receiver server ID</td>
<td>Receiver server ID</td>
</tr>
<tr>
<td>Sender client ID</td>
<td>Sender client ID</td>
</tr>
<tr>
<td>Key pair of sender client</td>
<td>Public-key obtained from sender client</td>
</tr>
<tr>
<td>Public-key obtained from Receiver server</td>
<td>Key pair of Receiver server</td>
</tr>
<tr>
<td>URL registering PUSH model Receiver server</td>
<td>URL registering PUSH model Receiver server</td>
</tr>
</tbody>
</table>

Set each item in turn, when setting up the Reliable messaging function. To use the non-repudiation (signature option) function, a public-key (used in verification of a SOAP digital signature) needs to be exchanged in advance.

Generate the sender application on the sender client and the receiver application on the receiver server separately. Then make the necessary adjustments for each of them. Also, note the following procedures.
Preparation of Receiver Application on PUSH Model Receiver Server

In a PUSH model (Receiver) server, a receiver application is generated in the following sequence:
1. Preparation of key pair and public-key
2. Preparation of receiver application
3. Allocation of receiver application
4. Setting of the Reliable messaging function.

Preparation of Key Pair and Public Key
This procedure is used for the non-repudiation (signature option) function. When this function is not used, the procedure starts with the next step, Preparation of Receiver Application.
For more details, refer to Preparation of Key Pair and Public-Key Used by Receiver Server in the Security System Guide.

Preparation of Receiver Application
Prepare a receiver application as a Java class implementing the OnewayListener interface of JAXM API.

Note: the ReqRespListener interface of JAXM API cannot be used.
The following is an implementation example of a receiver application:

```java
public class ReliableSampleServer implements OnewayListener {
    public void onMessage(SOAPMessage message) {
        SOAPEnvelope env = message.getSOAPPart().getEnvelope();
        :
    }
}
```

For details about how to implement the OnewayListener interface, refer to Chapter 4 Installing Applications that use the Messaging Method Application.
For the structure of the SOAPMessage class, refer to SOAP Message processing by SAAJ-API.
A receiver application is called after a PUSH model (receiver) server has received a SOAP message and acknowledgment has been made.
Even after a SOAP message is received, if the SOAP message is illegal (say, for example, the signature is erroneous) the receiver application will not be called. In this case, the failed attempt to receive a SOAP message is output to a log file for the Web service container.
Information specific to the Reliable messaging function (such as the message type and sender client ID) can be obtained by casting the SOAP message obtained (see below) to the SOAPReliableMessage class.

```java
SOAPReliableMessage relMessage = (SOAPReliableMessage)message;
String messageType = relMessage.getMessageType();
```

For more information, refer to the SOAPReliableMessage class of JavaDOC attached.

### Allocation of Receiver Application

After the class of the receiver application (Receive a SOAP message) has been compiled, allocate the Class file on the path set in the environment variable CLASSPATH of the server system environment. Unless a JAR file used for the high reliability Web service function is set in the environment variable CLASSPATH, environment development is required separately. For information about how JAR files are added, refer to Chapter 3, Configuring an Environment.

### Setting of the Reliable Messaging Function

Use the Web service information edit tool to edit the Reliable messaging function settings. For more information, refer to the Server Security System Guide.

Exchanged messages are stored in files by the message storage function of the Reliable messaging function. To obtain or delete a message after a communications session, refer to the API for Message Management. For details, refer to Management of Stored Messages.

### Preparation of Sender Application on PUSH Model Sender Client

In a PUSH model (sender) client, a sender application could be prepared in the following sequence:

1. Preparation of key information
2. Preparation of sender application
3. Setting of Reliable messaging function
4. Executing the sender application.

**Note**

Operating a sender application in more than one process may trigger unnecessary re-transmissions and message congestion during storage of messages, which can lower operation speed. Use of multiple threads is recommended to run processes concurrently.

### Preparation of Key Information

This procedure is used for the non-repudiation (signature option) function. When this function is not used, the procedure starts with the next step, Preparation of Sender Application.

For details of this procedure, refer to the Security System Guide.
Preparation of Sender Application

The sender application of the Reliable messaging function uses the ProviderConnection class of JAXM API to send a SOAP message.

The sender application could send a SOAP message through a virtual connection to the receiver. At this point, an actual request does not take place.

```java
ProviderConnectionFactory conFactory = ProviderConnectionFactory.newInstance();
ProviderConnection con = conFactory.createConnection();
```

Next, obtain the MessageFactory interface for generating a SOAP message. To generate a reliable message at some specific time, specify SOAPReliable in the argument of the createMessageFactory method.

A SOAP message other than "SOAPReliable" cannot be generated in the Reliable messaging function of the Interstage SOAP service.

```java
MessageFactory msgFactory = con.createMessageFactory("SOAPReliable");
```

Next, provide a SOAP message to be sent.

Set the target SOAP message to the SOAPMessage class obtained from the MessageFactory interface. Develop the target SOAP message using the same API as the one used in messaging in the SOAPMessage class.

For details about how to develop a SOAP message, refer to SOAP Message Processing by SAAJ-API.

```java
SOAPMessage message = msgFactory.createMessage();
```

Setting the destination and the message type to the SOAPMessage class specifies the receiver server ID and message type ID (that were agreed on with the receiver server).

When using the specified API, cast the SOAPMessage class to the SOAPReliableMessage class as shown in the following example. For the detailed API of the SOAPReliableMessage class, refer to the JavaDOC attached.

```java
SOAPReliableMessage relMessage = (SOAPReliableMessage)message;
//register destination
relMessage.setReceiver("server");
//register message type
relMessage.setMessageType("furikomi");
```
Finally, instruct the sending of the SOAP message. As an argument, specify the SOAPMessage class that has been generated from the MessageFactory interface, and to which the SOAP message is set using JAXM API.

```java
//send message
con.send(message);
```

This stores the SOAP message generated into a message storage location and starts sending of the SOAP message.

For the PUSH model, the SOAP message is sent to the receiver server when the SOAP message is stored. If an attempt has failed to send the message to the receiver server, re-transmission attempts are automatically repeated until timing out. It is possible to send a message from the sender application, irrespective of whether the receiver server is ready.

If correct acknowledgment is not received (despite re-transmission attempts of a specified re-transmission count), sending of the SOAP message fails. A record of the failed attempt to send the SOAP message will be created in the log file, specified in Chapter 12, Customizing the Web Service Container.

If sending of a reliable message has failed (and the server package of Interstage SOAP service is installed), an error message is recorded in the system log.

Whenever a sender application is active, it can detect successful sending of a message by providing a class implementing the SOAPReliableSendMessageListener interface.

The following provides an example of implementation in this class.

```java
SOAPReliableSendMessageListener listener = new
SOAPReliableSendMessageListener()
{
    public void messageSent(SOAPReliableMessageEvent event) {
        //processing on success
        String messageId = event.getMessageId();
        :
    }
};
```

Register the class instance generated as shown below.

```java
((SOAPReliableConnection)con).addSendMessageListener(listener);
```

For details of the event class passed to the sender application, refer to the JavaDoc attached.

It is also possible for the sender application to detect failure of message sending by providing a class implementing the SOAPReliableSendMessageErrorListener interface.

Following is an example of implementation in this class:

```java
SOAPReliableSendMessageErrorListener listener =
    new SOAPReliableSendMessageErrorListener() {
    public void messageSendError(SOAPReliableMessageErrorEvent event) {
        //processing on failure
        String messageId = event.getMessageId();
        int code = event.getErrorCode();
        :
    }
};
```
Chapter 7: How to use the Reliable Messaging Function

The created instance is registered as follows:

```java
((SOAPReliableConnection)con).addSendMessageErrorListener(listener);
```

For details of the event class passed to the sender application, refer to the JavaDoc attached.

In this method, after the sender application has restarted, notice is not sent to the sender application. For details about the failure or success of the sender application after the restart, refer to the log files.

**Setting of Reliable Messaging Function**

Use the Web service information edit tool to set the Reliable messaging function.

For details, refer to the Security System Guide.

To perform communications via proxy, proxy information must be set in advance. For details, refer to Defining the Client System Environment.

**Execution of Sender Application**

When the Reliable messaging function has been set, execute applications to send SOAP messages provided under Preparation of Sender Application using respective methods.

The Reliable messaging function must be separately started. Describe the following program source code in the main function of a client application. The execution timing is either before or after sending of a SOAP message. Having multiple calling times does not cause an error. Without this code, re-transmission of a SOAP message does not take place after restarting the client.

```java
SOAPReliable.init();
```
Structure of Application using PUSH Model

The PUSH model is effective in an environment that requires reliable messaging and non-repudiation and a reliable messaging receiver can be provided a web.

For example, if a SOAP message is sent from the terminal of retailer A to wholesaler B, the PUSH model is used.

![Diagram of Application Structure using PUSH Model](image)

**Figure 7-1 Application Structure using PUSH Model**

In the above figure, when an application such as the GUI application on the terminal at retailer A is started and a sales slip is filled out, the sender application on the PUSH model (sender) client is executed. Also, a reliable SOAP message is sent. On the PUSH model (Receiver) server at wholesaler B, when the SOAP message is received and processed by the server, a receiver application (provided in advance) is called.
PULL Model where Client System Receives Messages

In the PULL model, an application on the server system sends a SOAP message to an application on the client system.

Agreement Necessary in PULL Model

The following table contains items that must be agreed upon in advance, to set up applications on the (receiver) client and (sender) server of the PULL model:

<table>
<thead>
<tr>
<th>Item set by PUSH model sending party (client system)</th>
<th>Item set by PUSH model receiving party (server system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message type ID</td>
<td>Message type ID</td>
</tr>
<tr>
<td>Receiver client ID</td>
<td>Receiver client ID</td>
</tr>
<tr>
<td>Sender server ID</td>
<td>Sender server ID</td>
</tr>
<tr>
<td>Key pair of sender server</td>
<td>Public-key obtained from sender server</td>
</tr>
<tr>
<td>Public-key obtained from receiver client</td>
<td>Key pair of receiver client</td>
</tr>
<tr>
<td>URL registering PULL model sender server</td>
<td>URL registering PULL model sender server</td>
</tr>
</tbody>
</table>

Set each item in turn, when setting up the Reliable messaging function. To use the non-repudiation (signature option) function, a public-key (used in verification of a SOAP digital signature) needs to be exchanged in advance.

Generate the receiver application on the receiver client, and the sender application on the sender server separately, then make the necessary adjustments for each of them. Also, note the following procedures.
Preparation of Sender application on PULL Model Sender Server

In a PULL model (sender) server, a receiver application is generated in the following sequence:

1. Preparation of key pair and public-key
2. Preparation of sender application
3. Setting of the Reliable messaging function
4. Allocation of sender application.

Note
Operating a sender application on more than one Web service container may trigger message congestion during storage of messages, lowering the operation speed. It is recommended to operate a send operation on a single Web service container.

Preparation of Key Information

This procedure is used for the non-repudiation (signature option) function. When this function is not used, the procedure starts with the next step, Preparation of Sender Application.

For more details, refer to Preparation of Key Pair and Public-Key Used by Sender Server in the Security System Guide.

Preparation of Sender Application

To send a SOAP message, the sender application of the Reliable messaging function uses the ProviderConnection class of JAXM API.

For information about how to enable a user application to send messages, refer to Preparation of Sender Application (as the program code used is common.)

For the PULL model, only the SOAP message storage processing is made when the sender application is executed. When the Reliable messaging function of the Receiver client is started, the SOAP message stored is sent to the Receiver client.

Setting of Reliable Messaging Function

Use the Web service information edit tool to edit the Reliable messaging function settings.

For more information, refer to the Security System Guide.

Allocation of Sender Application

Execute a sender application for the PULL model (sender) server on a sender server machine. To operate a sender application on the Servlet container, which is also used in the SOAP service, execute the sender application from the Servlet/JSP operating with HTTP POST from the user.

SOAP messages sent from a sender application are stored in a file by way of the Reliable messaging function. This is so that the sender application can be executed with a timing independent of polling from the PULL model Receiver client. For a specific allocation example of a sender application on the PULL model sender server, refer to Structure of Application Using PULL Model.
Preparation of Sender Application on PULL Model Receiver Client

In a PULL model (Receiver) client, a receiver application is generated in the following sequence:

1. Preparation of key information
2. Preparation of receiver application
3. Allocation of receiver application
4. Setting of Reliable messaging function
5. Start of Reliable messaging function.

Note
Operating a receiver application in more than one process may trigger unnecessary re-transmissions and message congestion during storage of messages, which can lower operation speed. Use of multiple threads is recommended to run processes concurrently.

Preparation of Key Information
This procedure is used for the non-repudiation (signature option) function. When this function is not used, the procedure starts with the next step, Preparation of Receiver Application.
For details of this procedure, refer to the Security System Guide.

Preparation of Receiver Application
Provide a receiver application as a Java class in which the OnewayListener of JAXM API is implemented. Note that the ReqRespListener interface of JAXM API cannot be used.

The following is an implementation example of a receiver application:

```java
//receiver application processing section
public class ReliableSampleServer implements OnewayListener {
    public void onMessage(SOAPMessage message) {
        SOAPEnvelope env = message.getSOAPPart().getEnvelope();
        :
    }
}
```

For details of how to implement the OnewayListener interface, refer to Chapter 4 Installing Applications that use the Messaging Method.

For the structure of the SOAPMessage class, refer to SOAP Message Processing by SAAJ-API in Chapter 4.

A receiver application is called after a PULL model (Receiver) client has received a SOAP message and acknowledgment has been made.

Even when a SOAP message is received, if the SOAP message is illegal (say, for example, the signature is erroneous) the receiver application will not be called. In this case, the failed attempt to receive a SOAP message is output to a log file under Setting the Logs of the Client Function in Chapter 12, Customizing the Web Service Container.

If sender a reliable message fails (and the server package of Interstage SOAP service is installed) an error message is output to the system log.
Information specific to the Reliable messaging function (such as the message type and sender server ID) can be obtained by casting the SOAP message obtained (see below) to the SOAPReliableMessage class.

```java
//fetch receive information
SOAPReliableMessage relMessage = (SOAPReliableMessage)message;
String messageType = relMessage.getMessageType();
```

For the information that can be obtained, refer to the SOAPReliableMessage class of JavaDOC attached.

On a client system allocating a PULL model receiver application, the Reliable messaging function must be started. The following shows an example of description of the initialization method of the SOAPReliable class that starts the Reliable messaging function:

```java
//implement receiver application
public static void main(String[] args) {
    //Call initialization method of SOAPReliable class for starting reliable messaging
    SOAPReliable.init();
}
```

A method for starting the Reliable messaging function can be called without causing an error as many times as desired at any time. The method can be described on a Java class separately from that of the receiver application. Call an initialization method of the SOAPReliable class for starting the Reliable messaging function on one or more processes.

For more information, refer to JavaDOC attached.

**Allocation of Receiver Application**

After the class of the receiver application (receiving a SOAP message) has been compiled, allocate the Class file on the path set in the environment variable CLASSPATH of the client system environment.

Unless a JAR file used for the high reliability Web service function is set in the environment variable CLASSPATH, environment development is required separately. For information about how JAR files are added, refer to Chapter 2, Configuring an Environment.

**Setting of the Reliable Messaging Function**

Use the Web service information edit tool to set the Reliable messaging function settings.

For more information, refer to the Security System Guide.

To perform communications via proxy, proxy information must be set in advance. For details, refer to Defining the Client System Environment in Chapter 3.

A receiver application of the PULL model cannot receive a SOAP message. Be sure to perform the following processing to start the Reliable messaging function.
Starting Reliable Messaging Function

The Reliable messaging function is started by calling the initialization method for the SOAPReliable class from a receiver application.

The Reliable messaging function that has started, calls the receiver application class prepared under Preparation of Receiver Application. Thus, it is necessary to allocate the receiver application class in advance to the path set in the environment variable CLASSPATH.

Exchanged SOAP messages are stored in files by the message storage function of the reliable messaging as required. For information about the function to manage messages, refer to Management of Stored Messages.

Structure of Application Using PULL Model

The PULL model is effective in an environment where a Web server cannot be provided by the receiving party of SOAP messages (that require reliable messaging and non-repudiation between two specific parties.) However, the Web server used for reliable messaging can be provided by the sending party.

For example, if a SOAP message is sent from the Web server of wholesaler B to the client system of retailer A, the PUSH model is used.

Figure 7-2 Application Structure using PULL Model

In the above figure, a Web application is executed operating on the Web server (such as Servlet/JSP at wholesaler B). From this Web application, the sender application on the PULL model server is executed, and a reliable SOAP message is sent to retailer A. At this point, the send message is stored on the send server as in reliable messaging.

After that, when the Reliable messaging function on the Receiver client is started, a SOAP message is sent to the receive client as reliable messaging. Consequently, a receiver application set in the Receiver client is executed and the SOAP message is received.
Structure of Application Using a Combination of PUSH and PULL Models

A model using a combination of PUSH and PULL models is effective if either party cannot provide a Web server to receive SOAP messages (that require reliable messaging), as well as non-repudiation in communications of these SOAP messages between two specific parties, say, due to a firewall.

For example, consider the case of a message flow where a SOAP message input on retailer A is sent to wholesaler B and the processing result is sent to retailer A (both using the Reliable messaging function.) In this case, a model using a combination of PUSH and PULL models is used.

In the above figure, when the application on the terminal at retailer A is started and a sales slip is filled out, the sender application for the PUSH model (sender) client is executed. In addition, a reliable SOAP message is sent to wholesaler B.

For the Reliable messaging function at wholesaler B, a PUSH model receiver application (provided in advance) is called (when the PUSH model SOAP message is received), and reception processing is executed. During receive processing, the PUSH model sender application is executed, and a SOAP message (destined for retailer A of the PULL model receiver client) is generated. At this point, the SOAP message is stored on wholesaler B of the PULL model sender server.

Subsequently, from retailer A of the PULL model Receiver client, polling of the PULL model sender server for SOAP message automatically starts. The SOAP message stored is sent to retailer A of the PULL model-receiving client. Consequently, the PULL model receives an application set in retailer A and is executed and the SOAP message is received.
Note
Although the PUSH sender application and PULL receiver applications operate on retailer A, the Reliable messaging function does not associate SOAP messages passed to these two applications. If association is necessary, associated IDs must be positioned in the SOAP messages so as to provide an association between send messages and receive messages in the application.
Authentication of Reliable Messaging Function

This section describes how to provide authentication settings to allow only specific PUSH model sender clients and receiver clients to perform reliable messaging based on the message type ID.

Setting of Server System

Ensure that setup satisfies the following during the setting of the PUSH model receiver server and PULL model sender server.

- Separately set a message type ID for each PUSH model (sender) client and PULL model (receiver) client that needs to be discriminated from the other type.
- Separately set a Web service identifier for each message type ID.

After setting the Reliable messaging function using this procedure, set HTTP basic authentication for each receiver server URL and sensing server URL that have been determined. Set the basic authentication for each URL on the IJServer WorkUnit.

For details, refer to the J2EE User's Guide.

Setting of Client System

After setting the PUSH model (sender) client and PULL model (receiver) client, use the `soapsecrelconf` command to input information for basic authentication.

```
soapsecrelconf -add from -messageid message1 -partnerid partner1 -serviceid service -listener listenerclass -username userid -password password
```

Management of Stored Messages

The client system and server system stores transmitted reliable messages as files. This section describes how to handle stored messages.

Destination of Reliable Messages

Reliable messages are stored in the following directories:

**Windows**

C:\Interstage\F3FMsoap\etc\reliableMessage\n
**Solaris OE  Linux**

/opt/FJSVsoap/etc/reliableMessage/

**Solaris OE**

For multi-system operation, the following directory is used as a default destination.

/opt/FJSVsoap/MI/SystemName/etc/reliableMessage/

Actual messages are stored in the message destination. Messages are stored until they are deleted by a user application. Change the message destination as required, determined by the storage period and number of messages.

Message destination can be changed using either of the two methods below.

- Use the symbolic link of the directory

  **Solaris OE  Linux**

  Change /opt/FJSVsoap/etc/reliableMessage

  or

  /opt/FJSVsoap/MI/system name/etc/reliableMessage

  to the symbolic link to a larger-capacity directory

  Before performing this operation, it is recommended to ensure that message communications are not in progress, or to temporarily stop the IJServer WorkUnit.
• Use the `soapsecretconf` command to change the destination directory.
  Use the -storedir option of the `soapsecretconf` command to specify an arbitrary directory as a message destination for each message type.
  For details of this command, refer to the Reference Manual (Command Edition).

This message storage file uses the Java object format so that normal file access cannot obtain the file contents. To obtain the file contents, use the getMessage method of `MessageStore` class.

### Listing of Reliable Message IDs

Listing of reliable message IDs uses the `MessageStoreQuery` class. The following shows an example of the descriptions to obtain messages:

```java
// obtain list of IDs of unsent messages
String[] messageIds = MessageStoreQuery.getAllUnsentMessages();
```

```java
// obtain list of IDs of acknowledged messages
String[] messageIds = MessageStoreQuery.getAllMessages(MessageContext.MESSAGE_CONFIRM_RECEIVED);
```

For details, refer to the `MessageStoreQuery` class of JavaDOC attached.

### Obtaining Reliable Messages

Reliable messages can be obtained using the `getMessage` method for the `MessageStore` class. The following shows an example of description to obtain messages:

```java
MessageStoreFactory factory = MessageStoreFactory.getFactory("messageType");
MessageStore store = factory.getMessageStore("partnerId");
InterstageMessageContext mesContext = 
  (InterstageMessageContext)store.getMessage("messageId");
SOAPMessage message = mesContext.getMessage();
```

Specify the IDs set in the setting of Reliable messaging function of each client and server in messageType and partnerId.

For details, refer to the `MessageStore` class and `InterstageMessageContext` class of JavaDOC attached.

If the signature option (non-repudiation) is enabled, a SOAP message with a signature can be obtained.

The current Reliable messaging function does not support verification of the signature on a SOAP message obtained.

For verification from an application, separately use the XML Signature library, etc.
Deletion of Messages

To delete a message stored in the reliable message destination file, use the removeMessage method of the MessageStore class. The following shows an example of description for deletion of messages:

```java
MessageStoreFactory factory = MessageStoreFactory.getFactory("messageType");
MessageStore store = factory.getMessageStore("partnerId");
store.removeMessage("messageId");
```

For details, refer to the MessageStore class of JavaDOC attached.
Chapter 8

Support of Data Types

This chapter details the mapping of XML and Java data types and the mapping of XML and CORBA data types that are used in an RPC client application and a CORBA/SOAP gateway.

Note
"WWW," "XXX," "YYY," and "ZZZ" used in the explanation and tables in this chapter represent the following package names:

- WWW : com.fujitsu.interstage.soap.util
- XXX : com.fujitsu.interstage.soapx.types
- YYY : javax.xml.rpc.holders
- ZZZ : com.fujitsu.interstage.soapx.holders

Within the tables that detail the following data types, the prefixes included in XML data types indicate the following:

- AAA represents name space http://www.w3.org/2000/10/XMLSchema.
- xsd represents name space "http://www.w3.org/2001/XMLSchema"
- soap and soapenc represent name spaces that depend on the SOAP version.
  - For SOAP1.1: http://schemas.xmlsoap.org/soap/encoding/
  - For SOAP1.2: http://www.w3.org/2002/06/soap-encoding
Mapping of XML and Java Data Types

This section details the mapping of Java and XML data types to be used in RPC applications (RPC server application and RPC client application).

Note

Within an RPC application, different data types are used in the out/inout parameter and in/return. This section also describes data types to be used in the in/return and out/inout parameters. In a DII-system RPC client application, the data type for in/return is used in the out/inout parameter.

Simple Types

Below shows the correspondence between XML and Java data types to be used in an RPC application.

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
<th>in or return</th>
<th>out or inout</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:int</td>
<td>int</td>
<td></td>
<td>YYY.IntHolder</td>
</tr>
<tr>
<td>xsd:unsignedInt</td>
<td>XXX.UnsignedInt</td>
<td></td>
<td>ZZZ.UnsignedIntHolder</td>
</tr>
<tr>
<td>xsd:short</td>
<td>short</td>
<td></td>
<td>YYY.ShortHolder</td>
</tr>
<tr>
<td>xsd:unsignedShort</td>
<td>XXX.UnsignedShort</td>
<td></td>
<td>ZZZ.UnsignedShortHolder</td>
</tr>
<tr>
<td>xsd:long</td>
<td>long</td>
<td></td>
<td>YYY.LongHolder</td>
</tr>
<tr>
<td>xsd:unsignedLong</td>
<td>XXX.UnsignedLong</td>
<td></td>
<td>ZZZ.UnsignedLongHolder</td>
</tr>
<tr>
<td>xsd:byte</td>
<td>byte</td>
<td></td>
<td>YYY.ByteHolder</td>
</tr>
<tr>
<td>xsd:unsignedByte</td>
<td>XXX.UnsignedByte</td>
<td></td>
<td>ZZZ.UnsignedByteHolder</td>
</tr>
<tr>
<td>xsd:float</td>
<td>float</td>
<td></td>
<td>YYY.FloatHolder</td>
</tr>
<tr>
<td>xsd:double</td>
<td>double</td>
<td></td>
<td>YYY.DoubleHolder</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>boolean</td>
<td></td>
<td>YYY.BooleanHolder</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td></td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td>xsd:decimal</td>
<td>java.math.BigDecimal</td>
<td></td>
<td>YYY.BigDecimalHolder</td>
</tr>
<tr>
<td>xsd:integer</td>
<td>java.math.BigInteger</td>
<td></td>
<td>YYY.BigIntegerHolder</td>
</tr>
<tr>
<td>soap:base64</td>
<td>byte[]</td>
<td></td>
<td>YYY.ByteArrayHolder</td>
</tr>
<tr>
<td>xsd:base64Binary</td>
<td>byte[]</td>
<td></td>
<td>YYY.ByteArrayHolder</td>
</tr>
<tr>
<td>xsd:hexBinary</td>
<td>byte[]</td>
<td></td>
<td>YYY.ByteArrayHolder</td>
</tr>
<tr>
<td>AAA:timeInstant</td>
<td>java.util.Calendar</td>
<td></td>
<td>YYY.CalendarHolder</td>
</tr>
<tr>
<td>xsd:dateTime</td>
<td>java.util.Calendar</td>
<td></td>
<td>YYY.CalendarHolder</td>
</tr>
</tbody>
</table>
Note

- The line feed codes ("\r\n" and "\r") included in Java String-type data are converted to the line feed code ("\n") whenever received, then passed to an RPC application. An RPC client application obtains String-type data in which the converted return code is set in return values and inout/out parameters. An RPC server application obtains in/inout parameters String-type data in which only a single type of line feed code ("\n") is used.
- The maximum length of a byte-type array of a Java data type to be mapped using the soap:base64, xsd:base64Binary, and xsd:hexBinary types is the maximum value of the int type (2,147,483,647).
- When an RPC client application sends base64-type data, the name space representing the data type may be automatically converted depending on the SOAP version.

Classes Representing Unsigned Values

Java does not have the unsigned data type. Thus, use the following classes provided by the Interstage SOAP service as classes representing the unsigned data type.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX.UnsignedByte</td>
<td>A class representing XML data type, unsignedByte.</td>
</tr>
<tr>
<td>XXX.UnsignedShort</td>
<td>A class representing XML data type, unsignedShort.</td>
</tr>
<tr>
<td>XXX.UnsignedInt</td>
<td>A class representing XML data type, unsignedInt.</td>
</tr>
<tr>
<td>XXX.UnsignedLong</td>
<td>A class representing XML data type, unsignedLong.</td>
</tr>
</tbody>
</table>

A class representing an unsigned value has the following methods:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| XXX.UnsignedByte | public UnsignedByte(long value) | A constructor
Specify a value between 0 (minimum) and 255 (maximum). Negative values cannot be specified. |
| XXX.UnsignedByte | public byte byteValue()         | A method that returns a byte value.
Returns a negative value if the instance represents an unsignedByte value of 128 or greater. |
| XXX.UnsignedByte | public short shortValue()       | A method that returns a short value. |

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| XXX.UnsignedShort | public UnsignedShort(long value) | A constructor
Specify a value between 0 (minimum) and 65,535 (maximum). Negative values cannot be specified. |
### Class name | Method name | Explanation |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>public short shortValue()</td>
<td>A method that returns a short value. Returns a negative short value if the instance represents an unsignedShort value of 32,768 or greater.</td>
</tr>
<tr>
<td></td>
<td>public int intValue()</td>
<td>A method that returns an int value.</td>
</tr>
<tr>
<td>XXX.UnsignedInt</td>
<td>public UnsignedInt(long value)</td>
<td>A constructor Specify a value between 0 (minimum) and 4,294,967,295 (maximum). Negative values cannot be specified.</td>
</tr>
<tr>
<td></td>
<td>public int intValue()</td>
<td>A method that returns an int value. Returns a negative int value if the instance represents an unsignedInt value of 2,147,483,648 or greater.</td>
</tr>
<tr>
<td></td>
<td>public long longValue()</td>
<td>A method that returns a long value. Returns a negative long value if the instance represents an unsignedLong value of 9,223,372,036,854,775,808 or greater.</td>
</tr>
<tr>
<td>XXX.UnsignedLong</td>
<td>public UnsignedLong(double value)</td>
<td>A constructor Specify a value between 0 (minimum) and 18,446,744,073,709,551,615 (maximum). Negative values can be specified.</td>
</tr>
<tr>
<td></td>
<td>public long longValue()</td>
<td>A method that returns a long value. Returns a negative long value if the instance represents an unsignedLong value of 9,223,372,036,854,775,808 or greater.</td>
</tr>
<tr>
<td></td>
<td>public BigInteger bigIntegerValue()</td>
<td>A method that returns a java.math.BigInteger value.</td>
</tr>
</tbody>
</table>

#### Example

In the following example, "-1" and "255" is displayed at the second and third lines, respectively.

```java
XXXUnsignedByte ubyte = new XXXUnsignedByte((long)255); //first line
System.out.println(ubyte.byteValue());                //second line
System.out.println(ubyte.shortValue());              //third line
```

### out/inout

To use an out/inout parameter, an RPC server application uses a Holder class that retains data. These are the following Holder classes:

YYY.IntHolder, ZZZ.UnsignedIntHolder, YYY.ShortHolder, ZZZ.UnsignedShortHolder, ZZZ.DateHolder, YYY.LongHolder, ZZZ.UnsignedLongHolder, YYY.ByteHolder, ZZZ.UnsignedByteHolder, YYY.FloatHolder, YYY.DoubleHolder, YYY.BooleanHolder, YYY.StringHolder, YYY.ByteArrayHolder, YYY.BigDecimalHolder, YYY.BigIntegerHolder, ZZZ.ArrayOfIntHolder, ZZZ.ArrayOfUnsignedIntHolder, ZZZ.ArrayOfShortHolder,
Each of these classes only has value fields that represent retained data and constructors.

The following table explains the fields and constructors for the `YYY.IntHolder` class that retains `int`. Likewise, other `YYY.data-type-nameHolder` and `ZZZ.data-type-nameHolder` classes have value fields of retained types, default constructors, and constructors that specify initial values.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>YYY.IntHolder</code></td>
<td>public int value</td>
<td>A field. An int value that the instance currently retains.</td>
</tr>
<tr>
<td></td>
<td>public void IntHolder()</td>
<td>A default constructor. Constructs an instance with 0 in the value field.</td>
</tr>
<tr>
<td></td>
<td>public void IntHolder(int initial)</td>
<td>A constructor. Constructs an instance with initial in the value field.</td>
</tr>
</tbody>
</table>

A method of an RPC server application receives an instance of the Holder class in an out/inout parameter. It updates the value field of the received instance to send a new value in an out/inout parameter to an RPC client application.
Example
Usage sample of Holder Class

```java
// RPC server method
public int server_method(int in_para, IntHolder inout_para, StringHolder out_para) {
    int number = inout_para.value;
    // Assign a value to the value field of an out/inout parameter (which will be
    // sent to the client when returned).
    inout_para.value = 10;
    out_para.value = "abc";
}
```

A stub-system RPC client application uses an instance of the Holder class in an out/inout parameter. After
an RPC server application is called, a value returned by the RPC server application will be set in the value
field of the instance of the Holder class.

Following is an example of obtaining the value:

```java
// A stub-system RPC client application
import javax.xml.rpc.holders.IntHolder;

// Have parameters ready (arg3 is the out parameter).
int arg1 = 100;
int arg2 = 200;
IntHolder arg3 = new IntHolder();
// Call server methods with parameters int(in), int(in), and int(out)
//(The target is a stub object).
int result = target.opInt( arg1, arg2, arg3 );

// Obtain the value of the out parameter.
System.out.println( "arg3 = " + arg3.value );
```

A DII-system RPC client application does not refer to the Holder class. To obtain an out/inout parameter,
call an RCP server application, then obtain it using the get method (in which arguments are parameter
names) of the java.util.Map object, a return value of the getOutputParams method of the javax.xml.rpc.
Call class. You can obtain a value in the out or inout parameter in the same way.
Following is an example of obtaining a parameter.

// RPC-DII client application
// Obtain the value of the out/inout parameter.
java.util.Map outParams = call.getOutputParams();
Integer inout = (Integer)outParams.get("inout_para"); // parameter name is inout_para
String out = (String)outParams.get("out_para"); // parameter name is out_para

To use three-dimension or higher-dimension array data in the out/input parameter of an RPC server application, create a Holder class in the RPC server application (with the Holder array data in the value field). Following is an example of creating Holder-class data with a three-dimensional int array in the value field.

Example
An example of creating Holder-class data to be used in an RPC server application (simple array type) is shown below:

```java
import javax.xml.rpc.holders.Holder;
public class ArrayOfInt3DHolder implements Holder {
    public int[][][] value;
    public ArrayOfInt3DHolder() {} // default constructor
    public ArrayOfInt3DHolder(int[][][] initial){
        value = initial;
    }
}
```
Enumeration Type

Enumeration is a data type that can have one value in a set of user-defined values for a certain type (a simple type other than boolean).

For example, the user can arbitrarily define a string-type enumeration that can have "blue," "red," and "green" or an int-type enumeration that can have 1, 2, 3, 5, and 7.

Table 8-5  Enumeration Types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration</td>
<td>Arbitrary class defined as an enumeration</td>
</tr>
</tbody>
</table>

The following shows a WSDL definition of the XML enumeration type in which string types "blue," "red," and "green" are defined as a set of enumeration-type values.

```
<simpleType name="Color">
  <restriction base="xsd:string">
    <enumeration value="blue"/>
    <enumeration value="red"/>
    <enumeration value="green"/>
  </restriction>
</simpleType>
```

in/return

Define a class representing an enumeration to be used in an RPC application as the public class having the following methods. In this table, the "base type" represents a data type on which an enumeration should be based and the "enumeration type" represents a data type itself to be defined as an enumeration.

Table 8-6  In/return Methods Classes

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration type (arbitrary class defined as an enumeration)</td>
<td>protected enumeration-type (base-type value)</td>
<td>This constructor initializes a class representing an enumeration and returns an enumeration-type instance, from a base-type value on which an enumeration type should be based. Use this constructor to initialize the fields in a class. Normally, use fromValue() instead of this constructor.</td>
</tr>
<tr>
<td></td>
<td>public base-type getValue()</td>
<td>This method returns a value represented by this enumeration-type instance, as a base-type value on which an enumeration should be based.</td>
</tr>
<tr>
<td>Class name</td>
<td>Method name</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>public static <code>enumeration-type</code> fromValue(<code>base-type value</code>)</td>
<td>This method returns the value as this enumeration-type instance, when a base-type value on which an enumeration should be based is specified. If the specified base-type value does not match a value included in the enumeration, <code>java.lang.IllegalArgumentException</code> will be thrown.</td>
</tr>
<tr>
<td></td>
<td>public static <code>enumeration-type</code> fromString(<code>String value</code>)</td>
<td>This method returns the value as an enumeration-type instance, when the character-string representation of an enumeration-type value is specified. If the specified character-string representation does not match a value included in the enumeration, <code>java.lang.IllegalArgumentException</code> will be thrown.</td>
</tr>
<tr>
<td></td>
<td>public <code>String</code> toString()</td>
<td>This method returns the character-string representation of a data-type value retained in an instance.</td>
</tr>
<tr>
<td></td>
<td>public <code>boolean</code> equals(<code>Object obj</code>)</td>
<td>This method checks whether the characteristics of an obj is identical to those of the specified obj. If the specified obj has the same type as the class representing an enumeration and has the same data-type value as the one retained in the instance, the return value will be true. Otherwise, it will be false.</td>
</tr>
<tr>
<td></td>
<td>public static final <code>base-type_definition-name</code></td>
<td>This field retains each of the values included in an enumeration as a base-type value with a definition name added.</td>
</tr>
</tbody>
</table>

**Example**

The enumeration type consisting of three `String-type definition values` "blue," "red," and "green" can be defined as follows:

```java
public static final String _blue = "blue";
public static final String _red = "red";
public static final String _green = "green";
```
### Chapter 8: Support of Data Types

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
</table>
|            | public static final enumeration-type definition-name | This field retains each of the values included in an enumeration as an instance of this enumeration type with a definition name added. **Example**

The enumeration type consisting of three String-type definition values “blue,” “red,” and “green” can be defined as follows:

- public static final enumeration-type blue = new enumeration-type (_blue);
- public static final enumeration-type red = new enumeration-type (_red);
- public static final enumeration-type green = new enumeration-type (_green);

#### Example

Usage sample of enumeration class (enumeration of strings)

```java
Color col; // New declaration

// Retrieve a value: Using the fromValue() method
try {
    col = Color.fromValue("blue"); // Set a value.
} catch (IllegalArgumentException E) {
    Error handling if "blue" is a value not defined in class "Color"
}

// Retrieve a value: Using the fromString() method
try {
    col = Color.fromString("blue"); // Set a value.
} catch (IllegalArgumentException E) {
    Error handling if "blue" is a value not defined in class "Color"
}

// Retrieve a value: Using the fromString() method
col = Color.blue;
: System.out.println(col.toString());
```

Note that null cannot be specified as a value that a class representing an enumeration can have (When an object is constructed, java.lang.IllegalArgumentException will be thrown).
Example

```java
package fujitsu.sample; // package name

public final class Color {
    private String _value;
    public static final String _blue = "blue";
    public static final String _red = "red";
    public static final String _green = "green";

    public static final Color blue = new Color(_blue);
    public static final Color red = new Color(_red);
    public static final Color green = new Color(_green);

    public static Color fromValue(String value) {
        if (value.equals(_red)) return red;
        if (value.equals(_green)) return green;
        if (value.equals(_blue)) return blue;
        else
            throw new IllegalArgumentException(value + "is not a member of Color");
    }

    public static Color fromString(String value) {
        if (value.equals(_red)) return red;
        if (value.equals(_green)) return green;
        if (value.equals(_blue)) return blue;
        else
            throw new IllegalArgumentException(value + "is not string expression of any
member of Color");
    }

    protected Color(String value) {
        if (value == null ||
            !value.equals(_red) && !value.equals(_green) && !value.equals(_blue))
            throw new IllegalArgumentException(value + "is not a member of Color");
        _value = value;
    }

    public String getValue() {
        return _value;
    }

    public String toString() {
        return _value;
    }

    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (!(obj instanceof Color)) return false;
        return _value.equals(((Color) obj)._value);
    }
}
```

Note

To use an enumeration as a parameter of a method of an RPC server application, use the same class in in/return.
To use the enumeration type in an RPC client application, register the associated item using the register method of the javax.xml.rpc.encoding.TypeMappingRegistry class.

For information on the registration method, see Correspondence of Conversion Mapping Required to Convert between Java and XML Values.

**inout/out**

In this table, the "enumeration type" represents a data type to be defined as an enumeration and the "enumeration-type Holder" represents a Holder class that retains an enumeration-type value.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Field name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration-type Holder</td>
<td>public enumeration-type value</td>
<td>An instance field.</td>
</tr>
<tr>
<td>(Holder of an arbitrary class</td>
<td></td>
<td>An enumeration-type value currently retained in the instance.</td>
</tr>
<tr>
<td>defined as an enumeration)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void enumeration-type</td>
<td>A constructor.</td>
</tr>
<tr>
<td>Holder (enumeration-type initial)</td>
<td>Creates an instance with initial in the value field.</td>
</tr>
</tbody>
</table>

To use the enumeration type as an inout/out parameter of a method of an RPC server application, use the enumeration-type Holder as the data type for the parameter. The method of the RPC server application updates the value field of an enumeration-type Holder instance received as a parameter. Thus, the updated enumeration-type value will be sent to an RPC client application.

Following is an example of creating a Holder class having enumeration type Color in the value field:

**Example**

Example of creating a Holder class to be used in an RPC server application (enumeration type)

```java
import javax.xml.rpc.holders.Holder;
public class ColorHolder implements Holder {
    public Color value;
    public ColorHolder() {} // default constructor
    public ColorHolder(Color initial) {
        value = initial;
    }
}
```

**Example**

Following is an example of using enumeration as an inout parameter of an RPC server application:

```java
// Method of an RPC server application
// Register the enumeration type as parameter information to the Web service manager.
public void server_method(ColorHolder col) {
    // Setting of a value retained in col: Using the fromValue() method
    col.value = Color.fromValue("red");
}
```
// Setting of a value retained in col: Using the fromString() method
col.value = Color.fromString("red");
// Setting of a value retained in col: Directly using a public member
col.value = Color.red;
}

In the example above, the server method retains a Color object in the first parameter, col, and sends the value of the Color object as the out parameter to the RPC client application. Three methods are used to obtain the Color object representing "red" and retains it in col.

To use an enumeration-type array in the out/inout parameter of the RPC server application, create a Holder class. The following shows an example of creating a Holder class in order to use an enumeration-type one-dimensional array in the out/inout parameter of the RPC server application.

Example
Following is an example of creating a Holder class used in an RPC server application (enumeration array type).

```java
import javax.xml.rpc.holders.Holder;
public class ArrayOfColorHolder implements Holder {
  public Color[] value;
  public ArrayOfColorHolder() {} // default constructor
  public ArrayOfColorHolder(Color[] initial){
    value = initial;
  }
}
```

Note
- To use an enumeration in an RPC server application, register the associated item between a Java class representing an enumeration and an XML data type. For this purpose, register com.fujitsu.interstage.soapx.encoding.ser.EnumSerializerFactory and com.fujitsu.interstage.soapx.encoding.ser.EnumDeserializerFactory as a serializer (that converts a Java data type to an XML data type) and a deserializer (that converts an XML data type to a Java data type).

  For information on the registration method, see typeMapping Tag in XML Tags Used to Write Web Service Information.
- Null cannot be used as a definition value to be included in an enumeration.
Structure Type

The structure type is a data type that can have any number of data items (called members). The user can freely define the number of data items and their types.

Table 8-8  Structure Type and Data Types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure type</td>
<td>Any class defined as a structure</td>
</tr>
<tr>
<td>in or return</td>
<td>out or inout</td>
</tr>
</tbody>
</table>

The following shows a WSDL definition of the structure type in XML that has the String-type name and the int-type age as its members.

```xml
<complexType name="Human">
  <sequence>
    <element name="name" type="xsd:string ">
    <element name="age" type="xsd:int ">
  </sequence>
</complexType>
```

Define a public class having each of the members as a public field and use it as a structure as shown:

```java
package name
public classs StructureName
public memberType  memberName
public memberType  memberName
:
//public default constructor
public StructureName();
}
```

Example

Following is an example of defining a structure:

```java
package fujitsu.sample;  //package name
public class Human { //package name
    public String name;  // declares to each of the members
    public int age;
    public Human(){ // public default constructor
        name="nobody"; age=0;
    }
    public Human(String myname, int myage){ // public default constructor(Abbreviation)
        name = myname; age = myage;
    }
}
```
Note

- Any Java data type that can be used as an in parameter or return value can be used as a structure member.
- Only a public field can be used as a structure member. Such fields as private cannot be used as a structure member.
- To use a structure in an RPC client application, register the associated item using the register method of the javax.xml.rpc.encoding.TypeMappingRegistry class.

For information about the registration method, refer to Correspondence of Conversion Mapping Required for Conversion Between Java and XML Values.

To refer to (or assign) the value of a structure member, refer to (or assign) the field of an instance of the class used as a structure.

Example

Example of using a structure

```java
Human person = new Human(); // New declaration
    person.name = "John"; // Set a value.
    person.age = 25;
    String nameOfCustomer = person.name; // Obtain a value
    int ageOfCustomer = person.age;
```

out/inout

In the following table, the "structure type" represents an any class defined as a structure and the "structure Holder" represents a Holder class that retains a structure-type value.

**Table 8-9 Structure-type Holder**

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure-type Holder</td>
<td>public structure-type value</td>
<td>An instance field</td>
</tr>
<tr>
<td>(Holder of an any class defined as a structure)</td>
<td>public structure-type Holder (structure-type initial)</td>
<td>An structure value currently retained in the instance.</td>
</tr>
<tr>
<td></td>
<td>A constructor</td>
<td>Creates an instance with initial in the value field.</td>
</tr>
</tbody>
</table>

To use a structure as an inout/out parameter of a method of an RPC server application, use the structure-type Holder as the data type of the parameter. The method of the RPC server application updates the value field of a structure-type Holder instance received as a parameter. Thus, the updated structure-type value is sent to an RPC client application.

Following is an example of creating a Holder class with structure type Human in the value field.
Example
Example of creating a Holder class to be used in an RPC server application (structure type)

```java
package fujitsu.sample;  // package name
import javax.xml.rpc.holders.Holder;
public class HumanHolder implements Holder {
    public Human value;
    public HumanHolder() {} // default constructor
    public HumanHolder(Human initial){
        value = initial;
    }
}
```

Example
Example of using a structure as an inout parameter of an RPC server application

```java
// Method of an RPC server application
// Register a structure type as parameter information to the Web service manager.
public void server_method(HumanHolder person) {
    person.value.age = person.value.age+1; // When directly updating a received
    structure member
    person.value = new Human(person.value.name, (person.value.age+1)); // When
    creating a new structure
}
```

In the example above, the server method (when called) retains Human-type structure data sent by an RPC
client application in person (an inout parameter). After processing the method, the Human-type structure
data retained in person is sent to the RPC client application. This example shows two ways in which the
server method returns to the RPC client application the Human-type structure data with 1 added to age, a
member of data received (that is, 2 is added).

To use a structure-type array in an out/inout parameter of an RPC server application, create a Holder class.
The following shows an example of creating a Holder class to use a structure-type one-dimensional array
in an out/inout parameter of the RPC server application.

Example
Following is an example of creating a Holder class used in an RPC server application (structure array
type):

```java
package fujitsu.sample;  // package name
import javax.xml.rpc.holders.Holder;
public class ArrayOfHumanHolder implements Holder {
    public Human[] value;
    public ArrayOfHumanHolder() {} // default constructor
    public ArrayOfHumanHolder(Human[] initial){
        value = initial;
    }
}
```
Note
To use a structure type in an RPC server application, register the association of a Java class representing a structure and an XML data type. Also, register com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory and com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory as a serializer (that converts a Java data type to an XML data type) and a deserializer (that converts an XML data type to a Java data type.)

For information about the registration method, refer to typeMapping Tag in XML Tags Used to Write Web Service Information.

Bean Type

The Bean type is a data type similar to a structure, having an arbitrary number of data items (called members) as properties. An application developer can freely define the possible number of data items and their types.

In the Bean type, a member is represented using the set/get method. In the same way as for a structure, a member can also be represented as a public instance field.

Table 8-10  Bean Data Type

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in or return</td>
</tr>
<tr>
<td>Structure</td>
<td>Any class defined as the Bean type</td>
</tr>
<tr>
<td></td>
<td>out or inout</td>
</tr>
<tr>
<td></td>
<td>Any class defined as the Bean type</td>
</tr>
</tbody>
</table>

Following is a WSDL definition of the structure in XML, which has the string-type name and the int-type age as its members.

```xml
<complexType name="Human">
  <sequence>
    <element name="name" type="xsd:string "/>
    <element name="age" type="xsd:int ">
  </sequence>
</complexType>
```

in/return

Define a public class with each of the members as a public field of the set/get method and use it as the Bean type as shown below:

```java
package name
public classs BeanTypeName
  //public default constructor
  public BeanTypeName ()

  // Property (represented as an instance field)
  public dataTypeOfProperty propertyName 

  // Property (represented as the set/get method)
  public void setPropertyName(dataTypeOfProperty, value){ } 
```
Example:

Example of defining the Bean type

```java
package fujitsu.sample;  //package name
public class HumanBean {
    public HumanBean(){ // public default constructor
        public HumanBean() { name="nobody"; _age = 0; }
        public HumanBean(String name, int age){ //
            this.name=name;
            _age=age;
        }

        // StringType name Property (represented as an instance field)
        public String name;
        // StringType age Property (represented as an instance field)
        public void setAge(int age) { _age = age; }
        public int getAge() { return _age; }
        private int _age; // Not directly referred to as a member because this is private

    }
}
```

Note:

- Any Java data type that can be used as an in parameter or return value can be used as a property of the Bean type.
- To use the Bean type in an RPC client application, you need to register the associated item using the register method of the javax.xml.rpc.encoding.TypeMappingRegistry class.
  
  For information about the registration method, refer to the Correspondence of Conversion Mapping Required for Conversion Between Java and XML Values.

To refer to (or assign) the value of a Bean-type member, refer to (or assign) the field of an instance of the class used as a structure.

Example

An example of using a structure is shown on the following page.

```java
HumanBean person = new HumanBean(); // New declaration :
   person.name = "John"; // Set a value.
   person.setAge(25);
   :
   String nameOfCustomer = person.name; // Obtain a value.
   int ageOfCustomer = person.getAge(); // Obtain a value.
```
**out/inout**

In the following table, the "Bean type" represents an arbitrary class defined as the Bean type and a "Bean-type Holder" represents a Holder class that retains a Bean-type value.

**Table 8-11 Bean-type Holder**

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bean-type Holder (Holder of an arbitrary class defined as the Bean type)</td>
<td>public Bean-type value</td>
<td>An instance field</td>
</tr>
<tr>
<td></td>
<td>A structure value currently retained in the instance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>public Bean-type Holder (Bean-type initial)</td>
<td>A constructor</td>
</tr>
<tr>
<td></td>
<td>Creates an instance with initial in the value field.</td>
<td></td>
</tr>
</tbody>
</table>

To use the Bean type as an inout/out parameter of a method of an RPC server application, use the Bean-type Holder as the data type of the parameter. The method of the RPC server application updates the value field of a Bean-type Holder instance received as a parameter. Thus, the updated Bean-type value is sent to an RPC client application.

**Example**

Example of using the Bean type as an inout parameter of an RPC server application.

```java
// Method of an RPC server application
public void server_method(HumanBeanHolder person) // Register the Bean type as parameter information to the Web service
person.value.setAge() = person.value.getAge()+1; // When directly updating a received Bean-type member
person.value = new HumanBean(person.value.name, (person.value.getAge()+1)); // When creating a new Bean type
```

In the example above, the server method (when called) retains HumanBean-type data sent by an RPC client application in person (an inout parameter). After processing the method, the HumanBean-type data retained in person is sent to the RPC client application. This example shows two ways in which the server method returns to the RPC client application the HumanBean-type data with 1 added to age, a member of data received (that is, 2 is added).

**Note:**

To use the Bean type in an RPC server application, register the association of a Java class representing the Bean type and the XML data type. Register com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory and com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory as a serializer (that converts the Java data type to the XML data type) and a deserializer (that converts the XML data type to the Java data type).

For more information about the registration method, refer to Chapter 9, Managing Web Service Information.
Array Type

The following table shows the correspondence between the XML array type and the Java data type used in an RPC application.

Table 8-12  Array Type

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>soapenc:Array</td>
<td>Array ZZZ.ArrayOf data-type Holder</td>
</tr>
</tbody>
</table>

The following example shows a WSDL definition for a String-type array.

```xml
<complexType name='ArrayOfString'>
    <complexContent>
        <restriction base='soapenc:Array'>
            <attribute ref='soapenc:arrayType' wsdl:arrayType='xsd:string[]'/>
        </restriction>
    </complexContent>
</complexType>
```

Note

- An RPC application can handle an array type having elements that are "any Java data type that can be used as an in parameter or return value, except for the java.util.Vector class."
  
  For information on the data type mapping for elements, see "Simple Type," "Enumeration Type," "Structure Type," and "Bean Type."

- When an RPC client application sends the array type, a name space representing the data type is automatically converted according to the SOAP version.

in/return

A Java array will be used.

Example of an int-type one-dimensional array

```java
int[] value;
```

Example of an UnsignedInt-type two-dimensional array

```java
com.fujitsu.interstage.soapx.types.UnsignedInt[][] value;
```

Example of a structure-type three-dimensional array

```java
any-class-defined-as-structure-type [][][] value;
```

Example of an enumeration-type one-dimensional array

```java
any-class-defined-as-enumeration-type [] value;
```
Example of an enumeration-type three-dimensional array

```java
any-class-defined-as-enumeration-type [][][] value;
```

Example of a Bean-type two-dimensional array

```java
any-class-defined-as-Bean-type [][] value;
```

### out/inout

[One-dimensional Holder]

- `ZZZ.ArrayOfIntHolder`, `ZZZ.ArrayOfUnsignedIntHolder`
- `ZZZ.ArrayOfShortHolder`, `ZZZ.ArrayOfUnsignedShortHolder`
- `ZZZ.ArrayOfDateHolder`, `ZZZ.ArrayOfUnsignedLongHolder`
- `ZZZ.ArrayOfByteHolder`, `ZZZ.ArrayOfUnsignedByteHolder`
- `ZZZ.ArrayOfFloatHolder`, `ZZZ.ArrayOfDoubleHolder`
- `ZZZ.ArrayOfStringHolder`, `ZZZ.ArrayOfBigIntegerHolder`
- `ZZZ.ArrayOfImageHolder`, `ZZZ.ArrayOfSourceHolder`
- `ZZZ.ArrayOfMimeMultipartHolder`, `ZZZ.ArrayOfDataHandlerHolder`

[Two-dimensional Holder]

- `ZZZ.ArrayOfInt2DHolder`, `ZZZ.ArrayOfUnsignedInt2DHolder`
- `ZZZ.ArrayOfShort2DHolder`, `ZZZ.ArrayOfUnsignedShort2DHolder`
- `ZZZ.ArrayOfDate2DHolder`, `ZZZ.ArrayOfUnsignedLong2DHolder`
- `ZZZ.ArrayOfByte2DHolder`, `ZZZ.ArrayOfUnsignedByte2DHolder`
- `ZZZ.ArrayOfFloat2DHolder`, `ZZZ.ArrayOfDouble2DHolder`
- `ZZZ.ArrayOfString2DHolder`, `ZZZ.ArrayOfBigInteger2DHolder`
- `ZZZ.ArrayOfImage2DHolder`, `ZZZ.ArrayOfSource2DHolder`
- `ZZZ.ArrayOfMimeMultipart2DHolder`, `ZZZ.ArrayOfDataHandler2Dholder`

To use an array as an out/inout parameter, use the RPC server application in the `ZZZ.ArrayOf` data-type Holder class. This class has a field that retains an array and a constructor, as shown in the following table:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ZZZ.ArrayOf data-type Holder</code></td>
<td>public data-type-array value</td>
<td>An instance field</td>
</tr>
<tr>
<td></td>
<td>public void ArrayOf data-type Holder (data-type-array initial)</td>
<td>A constructor. Creates an instance with initial in the value field.</td>
</tr>
</tbody>
</table>

**Note**

Caution: `ZZZ.ArrayOf data-type 2Dholder` will be used for a two-dimensional array.
To use an array as an inout/out parameter of a method of an RPC server application, use the ZZZ.ArrayOf data-type Holder as the data type for the parameter. The method of the RPC server application updates the value field of a ZZZ.ArrayOf data-type Holder class instance received as a parameter. Thus, the updated array is sent as an out/inout parameter to an RPC client application.

To use a simple-type array with three or more dimensions as an inout/out parameter in an RPC server application, create a Holder class with the array data type in the inout/out parameter specified in value.

Example

Example of creating a three-dimensional Holder class to be used in an RPC server application

```java
import javax.xml.rpc.holders.Holder;
class ArrayOfString3DHolder implements Holder {
    public String[][][] value;
    public ArrayOfString3DHolder() {}  // default constructor
    public ArrayOfString3DHolder( String[][][] initial ) {
        this.value = initial;
    }
}
```

Example

Example of using an int-type one-dimensional array in an RPC client application

```java
//RPC client application
package client;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.encoding.XMLType;
import java.util.Map;
class arrayClient {
    public static void main(String[] args){
        Service service = ServiceFactory.newInstance().createService(new QName("",""));
        Call call = service.createCall();
        call.addParameter( "iArrayHolder", XMLType.SOAP_ARRAY, int[].class, ParameterMode.INOUT );
        call.setReturnType(XMLType.SOAP_ARRAY,int[].class);
        // configure the parameter
        int[] inArray = new int[10];
        for(int i=0; i<inArray.length; i++)
        {  
inArray[i] = i;
        }
        Object[] params = new Object[]{ inArray };
        // Call the Web service and obtain the return value
        int[] _ret = (int[])call.invoke(params);
    }
}
```
Example of using an int-type one-dimensional array as an inout parameter for an RPC server application:

```java
package server;
import com.fujitsu.interstage.soapx.holders.ArrayOfIntHolder;
public class arrayServer{
    public int[] server_method(ArrayOfIntHolder iArrayHolder){
        int[] iArray = iArrayHolder.value;  // Obtain an array retained in ArrayOfIntHolder.
        int[] newArray = new int[iArray.length];
        for(int i=0;i<iArray.length;i++){
            newArray[i] = iArray[i] * 2;
        }
        iArrayHolder.value = newArray;  // Re-configure an array retained in ArrayOfIntHolder.
        return newArray;
    }
}
```

In the example above, the server method retrieves an array retained in the ArrayOfIntHolder object of the first parameter, doubles the value retained in each element, then sends the resultant value as an inout parameter to an RPC client application.

Example of coding in a Web service information file

```xml
<service name="" provider="java:RPC">
    <parameter name="className" value="server.arrayServer.server_method"/>
    <parameter name="allowedMethods" value="*"/>
    <operation name="server_method">
        <parameter name="iArrayHolder" type="soapenc:Array"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" mode="INOUT"/>
    </operation>
</service>
```
java.util.Vector Type

An RPC application can use the java.util.Vector class to send or receive an array containing two or more arbitrary data types described in this section.

Table 8-14 java.util.Vector class

<table>
<thead>
<tr>
<th>Data type to be registered as an XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>issoap:Vector</td>
<td>java.util.Vector ZZZ.VectorHolder</td>
</tr>
</tbody>
</table>

"issoap" in the above table represents name space "urn:xmlns-fujitsu-com:xml-soap-2002-10."

Note

- An RPC application cannot handle an array having elements of the java.util.Vector class.
  - In addition, it cannot send or receive an instance of the java.util.Vector class having itself as an element (looping) as follows:
    ```java
    Vector vec = new Vector();
    vec.addElement(vec);      // The instance itself is an element.
    ```
  - The java.util.Vector-type data will be sent and received as an XML anyType-type array.

in/return

Use the java.util.Vector class.

The Java data type that can be used as a Vector element is the same as a Java data type that can be used as an in parameter or return value.

Note that, if a Vector element is null, the information on the type of this element will not be retained.

Example

```java
Vector vec = new Vector();
String str = null;
vec.addElement(str); // The information that indicates that the element is the /java.lang.String type will not be retained.
```

out/inout

To use the java.util.Vector class as an inout/out parameter of a method of an RPC server application, use the ZZZ.VectorHolder class as the data type of the parameter. The method of the RPC server application will update the value field of a ZZZ.VectorHolder class instance received as a parameter. Thus, the modified java.util.Vector in the value field of VectorHolder will be sent to an RPC client application.

Example

Example of using this class as an inout parameter of an RPC server application is shown below:

```java
// Method of an RPC server application
// Register vec as an inout parameter to the Web service manager.
```
public void server_method(VectorHolder vec) {
    Object[] objects = vec.value.toArray(); // Obtain a Vector element as an array of Object.
    for(int i = 0; i<objects.length; i++) {
        String longString = (String)objects[i] + "00";
        vec.value.set(i,longString); // Set a Vector
    }
}

In the example above, the server method adds character string "00" to the end of all the elements (all assumed as the java.lang.String type) in the value field of an instance of the ZZZ.VectorHolder class as an inout parameter and sends the resultant value to a client.

### Attachment Types

Use the following data types as parameters of an RPC application to exchange SOAP messages with attached files ("attachments") without being concerned with the handling of these attachments.

**Table 8-15  Attachment Types**

<table>
<thead>
<tr>
<th>Data type to be registered as an XML data type</th>
<th>Parameter of RPC application (Java data type class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>issoap:Image</td>
<td>java.awt.Image</td>
</tr>
<tr>
<td>issoap:Source</td>
<td>javax.xml.transform.Source</td>
</tr>
<tr>
<td>issoap:Multipart</td>
<td>javax.mail.internet.MimeMultipart</td>
</tr>
<tr>
<td>issoap:DataHandler</td>
<td>javax.activation.DataHandler</td>
</tr>
<tr>
<td>issoap:PlainText</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>out or inout</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZZ.ImageHolder</td>
</tr>
<tr>
<td>ZZZ.SourceHolder</td>
</tr>
<tr>
<td>ZZZ.MimeMultipartHolder</td>
</tr>
<tr>
<td>ZZZ.DataHandlerHolder</td>
</tr>
<tr>
<td>YYY.StringHolder</td>
</tr>
</tbody>
</table>
The following constants are defined as data type names to be registered as XML data types.

### Table 8-16 Package Name: com.fujitsu.interstage.soapx

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants</td>
<td>MIME_IMAGE</td>
<td>A javax.xml.namespace.Qname-type constant representing a name with a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name space of attachment type &quot;issoap:Image&quot;</td>
</tr>
<tr>
<td></td>
<td>MIME_SOURCE</td>
<td>A javax.xml.namespace.Qname-type constant representing a name with a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name space of attachment type &quot;issoap:Source&quot;</td>
</tr>
<tr>
<td></td>
<td>MIME_MULTIPART</td>
<td>A javax.xml.namespace.Qname-type constant representing a name with a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name space of attachment type &quot;issoap:Multipart&quot;</td>
</tr>
<tr>
<td></td>
<td>MIME_DATA_HANDLER</td>
<td>A javax.xml.namespace.Qname-type constant representing a name with a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name space of attachment type &quot;issoap:DataHandler&quot;</td>
</tr>
<tr>
<td></td>
<td>MIME_PLAINTEXT</td>
<td>A javax.xml.namespace.Qname-type constant representing a name with a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name space of attachment type &quot;issoap:PlainText&quot;</td>
</tr>
</tbody>
</table>

"issoap" in the above table represents name space "urn:xmlns-fujitsu-com:xml-soap-2002-10."

If a class other than javax.activation.DataHandler is used as the attachment type, data will be converted according to the used class and thus the received attachment may have different data than the source attachment.

If you need to receive the same data as the source (avoid data conversion), use the javax.activation.DataHandler class.

If, for example, the java.lang.String class is used, the data in the attachment will be converted to the Java character code regardless of the source character code.

**Note**

javax.mail.internet.MimeMultipart and javax.activation.DataHandler are the API classes of JavaMail (TM) and Java Activation Framework (JAF). See the supplied JavaDOC.

Solaris OE Linux

To use an issoap:Image-type attachment, you must have an environment in which an application in the X Window system can run. Alternatively, use a java.awt.Image object for an issoap:DataHandler or issoap:Multipart-type attachment (that is, use MIME types image/jpeg and image/gif).
As for an ordinary data type, use an object of the "in or return" class (see table, above) as a parameter.

Example

The following shows an example of using an object of this class in a DII-system client application.

```java
Source source = new StreamSource(new FileInputStream("sample.xml"));
call.addParameter( "v", Constants.MIME_SOURCE, ParameterMode.IN );
Object resp = call.invoke( new Object[]{ source } );
```

In the same way as for an ordinary data type, use an object of the "out or inout" class (see the above table) as a parameter. Each of these classes has only a field representing the retained data and a constructor.

The following table explains the field and constructor for the ZZZ.SourceHolder class that retains javax.xml.transform.Source. Likewise, the YYY.data-type-name.Holder and ZZZ.data-type-nameHolder classes also have the value field for the retained type, default constructor, and constructor that specifies the initial value.

### Table 8-17 Field and Constructor for the ZZZ.SourceHolder Class

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method name/Field name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceHolder</td>
<td>public Source value</td>
<td>An instance field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A Source value currently retained in the instance.</td>
</tr>
<tr>
<td></td>
<td>public SourceHolder (Source initial)</td>
<td>A constructor. Creates an instance with initial in the value field.</td>
</tr>
</tbody>
</table>

Example

Following is an example of using this class as an inout parameter of an RPC server application.

```java
// Method of an RPC server application
// Register sh as an inout parameter to the Web service manager.
public void serverMethod(SourceHolder sh) {
    Source inSource = sh.value;
    : (Processing of creating a source to be returned as outSource (omitted))
    sh.value = outSource;
}
```

In the example above, the server method replaces the contents of an instance of the ZZZ.SourceHolder class received as an inout parameter and sends the resultant value to a client.
Multiple Reference

Multiple reference means referring to one instance from multiple variables.

If an RPC application sends multiple variables that refer to the same instance in the class to be used as the structure, Bean, or enumeration type or the java.util.Vector class, the destination RPC application will also be in a status in which they refer to the same instance.

Example

Variables, person_a and person_b refer to the same Human-class instance. If both the RPC client application and RPC server application send both person_a and person_b, these two variables refer to the same Human-class instance also at the destination.

```java
Human person_a = new Human(); // Create a Human-class instance.
Human person_b = person_a;    // Both person_a and person_b refer to the same instance.
```

Single reference (creating an instance for every variable value) will be used on a data type other than the class to be used as the structure, Bean, or enumeration type or the java.util.Vector class.

Correspondence of Conversion Mapping Required for Conversion Between Java and XML Values

To use the structure, Bean, or enumeration type in a DII-system RPC client application, use the register method of the javax.xml.rpc.encoding.TypeMapping class to define the correspondence of conversion mapping by specifying:

- A defined class of the user-defined type,
- An XML data type name with a name space, and
- An object of the serializer or deserializer factory that depends on the data type.

The following table describes the classes of the serializer and deserializer factories.
### Serializer and Deserializer Factories Used to Register the Structure and Bean types

<table>
<thead>
<tr>
<th>Class name</th>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeanSerializerFactory implements javax.xml.rpc.encoding.SerializerFactory</td>
<td>public BeanSerializerFactory ( Class beanClass, javax.xml.namespace.QName typeName )</td>
<td>Constructs a structure/Bean-type serializer factory that associates user-defined class &quot;beanClass&quot; to a name-space-attached XML data type named as &quot;typeName.&quot;</td>
</tr>
<tr>
<td>BeanDeserializerFactory implements javax.xml.rpc.encoding.DeserializerFactory</td>
<td>public BeanDeserializerFactory ( Class beanClass, javax.xml.namespace.QName typeName )</td>
<td>Constructs a structure/Bean-type deserializer factory that associates a name-space-attached XML data type named as &quot;typeName&quot; to user-defined class &quot;beanClass.&quot;</td>
</tr>
</tbody>
</table>

To use the user-defined structure or Bean type, use the above constructor to create an instance, then use the register method of the javax.xml.rpc.encoding.TypeMapping class to register the instance. An application does not need to use a method of the serializer or deserializer fact.

### Serializer and Deserializer Factories Used to Register the Enumeration Type

<table>
<thead>
<tr>
<th>Class name</th>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnumSerializerFactory implements javax.xml.rpc.encoding.SerializerFactory</td>
<td>public EnumSerializerFactory ( Class enumClass, javax.xml.namespace.QName typeName )</td>
<td>Constructs an enumeration-type serializer factory that associates user-defined class &quot;enumClass&quot; to a name-space-attached XML data type named as &quot;typeName.&quot;</td>
</tr>
<tr>
<td>EnumDeserializerFactory implements javax.xml.rpc.encoding.DeserializerFactory</td>
<td>Public EnumDeserializerFactory ( Class enumClass, javax.xml.namespace.QName typeName )</td>
<td>Constructs an enumeration-type deserializer factory that associates a name-space-attached XML data type named as &quot;typeName&quot; to user-defined class &quot;enumClass.&quot;</td>
</tr>
</tbody>
</table>

To use the user-defined enumeration type, use the above constructor to create an instance, then use the register method of the javax.xml.rpc.encoding.TypeMapping class to register the instance. An application does not need to use a method of the serializer or deserializer factory.
Example of Association of Conversion Mapping

The following shows an example of association of conversion mapping for a structure.

- Name space: urn:Fujitsu-Soap-Service-Data
- Name: package-name - java-class-name ("a-b-c" if the Java class name is "a.b.c")

```java
package fujitsu.sample; // package name
public class Human
    implements java.io.Serializable {
    :
}
```

The association of mapping of Java classes defined in this remote interface definition is as follows:

```java
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
import javax.xml.rpc.encoding.TypeMappingRegistry;
import javax.xml.rpc.encoding.TypeMapping;

service = ServiceFactory.newInstance().createService(new QName("",""));
TypeMappingRegistry tmr = service.getTypeMappingRegistry();
TypeMapping tm = tmr.createTypeMapping();
// structure: Register mapping of structure, Human.
QName _qname = new QName( "urn:Fujitsu-Soap-Service-Data",
"fujitsu-sample-Human" );
tm.register( fujitsu.sample.Human.class, _qname,
    new com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory(
        fujitsu.sample.Human.class, _qname ),
    new com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory(
        fujitsu.sample.Human.class, _qname ) );
tmr.register( "http://schemas.xmlsoap.org/soap/encoding/", tm ); // For SOAP1.1
tmr.register( "http://www.w3.org/2002/06/soap-encoding", tm ); // For SOAP1.2
```
Mapping of XML and CORBA Data Types

This section details the mapping of XML and CORBA data types.

Use the data types explained in this section to create an RPC application that links with a CORBA application.

**Note**
- In an RPC client application, use a Java data type in in/return value (return) and a Holder class with a value of a Java data type in inout/out. See the correspondence table in Basic Data Types.
- For information on handling an XML data type in an RPC application, refer to Mapping of XML and Java Data Types.

Basic Data Types

The following table shows the correspondence between the CORBA and XML basic data types.

Use a "Java data type in an RPC application" if the RPC application links with a CORBA application that uses a "CORBA data type."

**Table 8-20 Correspondence Between the CORBA and XML Basic Data Types**

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/return</td>
<td>out/inout</td>
<td></td>
</tr>
<tr>
<td>xsd:int</td>
<td>int</td>
<td>long</td>
</tr>
<tr>
<td>xsd:unsignedInt</td>
<td>XXX.UnsignedInt</td>
<td>ZZZ.UnsignedIntHolder</td>
</tr>
<tr>
<td>xsd:short</td>
<td>short</td>
<td>YYY.ShortHolder</td>
</tr>
<tr>
<td>xsd:unsignedShort</td>
<td>XXX.UnsignedShort</td>
<td>ZZZ.UnsignedShortHolder</td>
</tr>
<tr>
<td>xsd:long</td>
<td>long</td>
<td>YYY.LongHolder</td>
</tr>
<tr>
<td>xsd:float</td>
<td>float</td>
<td>YYY.FloatHolder</td>
</tr>
<tr>
<td>xsd:double</td>
<td>double</td>
<td>YYY.DoubleHolder</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>boolean</td>
<td>YYY.BooleanHolder</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td>xsd:unsignedByte</td>
<td>XXX.UnsignedByte</td>
<td>ZZZ.UnsignedByteHolder</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
</tbody>
</table>
Enumeration Type

The following table shows the correspondence between CORBA enumeration (enum) and XML data type. Within an RPC application, the Java data type has a data name SOAPEnumImpl class in the in parameter/return value (return) and data name SOAPEnumImplHolder in the out parameter/inout parameter. In an RPC application, use the "data-name SOAPEnumImpl" and "data-name SOAPEnumImplholder" class for the enum type of a CORBA application that is linked.

Table 8-21  Correspondence Between CORBA Enumeration (enum) and XML Data Type

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration</td>
<td>data-name SOAPEnumImpl</td>
<td>enum</td>
</tr>
<tr>
<td></td>
<td>data-name SOAPEnumImplHolder</td>
<td></td>
</tr>
</tbody>
</table>

Note

To use the enumeration type in an RPC client application, you need to provide conversion mapping of Java and XML data types. For more information, refer to Chapter 6, Installing a CORBA/SOAP Gateway.

Example

IDL compilation of the following IDL definition

```idl
module fujitsu {
    module sample {
        enum Color {blue, red, green};
    };
};
```

The following enum name + SOAPEnumImpl class will be created.

```java
generate fujitsu.sample;
public class ColorSOAPEnumImpl {
    private String _value;
    public static final String _blue = "blue";
    public static final String _red = "red";
    public static final String _green = "green";
    public static final ColorSOAPEnumImpl blue = new ColorSOAPEnumImpl(_blue);
    public static final ColorSOAPEnumImpl red = new ColorSOAPEnumImpl(_red);
    public static final ColorSOAPEnumImpl green = new ColorSOAPEnumImpl(_green);

    private static final String[] member = new String[] {
        "blue",
        "red",
        "green" }
    protected ColorSOAPEnumImpl() {
    }
    protected ColorSOAPEnumImpl( String value ) {
        _value = value;
    }
```
public String getValue()
{
    return _value;
}

public static ColorSOAPEnumImpl fromValue( String value )
{
    if( value.equals(_blue)) return blue;
    else if( value.equals(_red)) return red;
    else if( value.equals(_green)) return green;
    else
        throw new java.lang.IllegalArgumentException("Invalid Enum value: " + value);
}

public static ColorSOAPEnumImpl fromString( String value )
{
    if( value.equals(_blue)) return blue;
    else if( value.equals(_red)) return red;
    else if( value.equals(_green)) return green;
    else
        throw new java.lang.IllegalArgumentException("Invalid Enum string: " + value);
}

public boolean equals( Object obj )
{
    if( obj instanceof fujitsu.sample.ColorSOAPEnumImpl )
    {
        if(((fujitsu.sample.ColorSOAPEnumImpl)obj).getValue() == getValue())
            return(true);
    }
    return(false);
}

public int hashCode()
{
    return(getValue().hashCode());
}

public String toString()
{
    return (_value);
}

public int getIndex()
{
    if( _value == null ){
        throw new java.lang.IllegalArgumentException("Mapping fail, non initialize.");
    }
    if( member == null ){
        throw new java.lang.IllegalArgumentException("Enumm member non exsit.");
    }
    for(int index = 0 ; index < member.length; index++)
    {
        if(_value.equals(member[index])){
            return(index);
        }
    }
}
throw new java.lang.IllegalArgumentException("Mapping fail, invalid Enum :
"+ _value);
}
public void setByIndex(int index)
{
    if(0<=index && index<member.length){
        _value = member[index];
        return;
    }
    throw new java.lang.IllegalArgumentException("Invalid Enum index: "+index);
}

### Structure and Exception Types

The following table shows the correspondence between the CORBA structure (struct) and exception (exception) types and the XML data types.

In an RPC application, the Java data type will be the "data-name (struct-type-name or exception-type-name) SOAPGWType" class created by the IDL compiler in the in parameter/return value (return) and the "data-name (struct-type-name) SOAPGWTypeHolder" class created by the IDL compiler in the out parameter/inout parameter.

To link the struct and exception types of a CORBA application in an RPC application, use the "data-name SOAP SOAPGWTypel" class in the in parameter/return value (return) and "data-name (struct-type-name) SOAPGWTypeHolder" class created by the IDL compiler.

#### Table 8-22 Correspondence Between CORBA structure (struct) and exception (exception) types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>data-name SOAPGWType</td>
<td>data-name SOAPGWTypeHolder</td>
</tr>
<tr>
<td>Structure</td>
<td>data-name SOAPGWType</td>
<td>--</td>
</tr>
</tbody>
</table>

**Note**

- To use a structure or exception type in an RPC client application, prepare conversion mapping of Java and XML data types.
- For information about creating an RPC application that uses the structure type, refer to Structure Type. For information about conversion mapping of Java and XML data types, refer to Chapter 6, Installing a CORBA/SOAP Gateway.
- For information about creating an RPC application that uses the exception type, refer to Chapter 6, Installing a CORBA/SOAP Gateway.
Example

IDL compilation of the following IDL definition

```idl
module fujitsu {
    module sample {
        struct Human {
            string name;
            long age;
        };
    };
};
```

The following struct name + SOAPGWType class will be created.

```java
package fujitsu.sample;
final public class HumanSOAPGWType implements java.io.Serializable {
    public java.lang.String name;
    public int age;
    public HumanSOAPGWType() {}
    public HumanSOAPGWType( org.omg.CORBA.ORB _orb, fujitsu.sample.Human Human_corba )
    {
        this.setCORBAType( _orb, Human_corba );
    }
    public void setCORBAType( org.omg.CORBA.ORB _orb, fujitsu.sample.Human Human_corba )
    {
        this.name = Human_corba.name;
        this.age = Human_corba.age;
        return;
    }
    public fujitsu.sample.Human getCORBAType( org.omg.CORBA.ORB _orb )
    {
        fujitsu.sample.Human _ret = new fujitsu.sample.Human();
        _ret.name = this.name;
        _ret.age = this.age;
        return _ret;
    }
    public java.lang.String getName(){
        return( this.name );
    }
    public void setName(java.lang.String _name){
        this.name = _name;
    }
    public int getAge(){
        return( this.age );
    }
    public void setAge(int _age){
        this.age = _age;
    }
}
```
Array/Sequence Type

The following table shows the correspondence between the CORBA array (array) and sequence (sequence) types and the XML data types.

For the array and sequence types of a linkage CORBA application, use an array in the in parameter/return value (return) and a Holder class in the out parameter/inout parameter.

The Holder class to be used depends on the type of an array element and the number of dimensions of an array.

### Table 8-23 Correspondence Between CORBA Array (array) and Sequence (sequence) Types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>soapenc:Array</td>
<td>Array</td>
<td>Array</td>
</tr>
<tr>
<td>soapenc:Array</td>
<td>Array</td>
<td>Holder class</td>
</tr>
</tbody>
</table>

The following table shows the relationship between the type of an array element and a Holder class in terms of the number of dimensions.

### Table 8-24 Relationship Between Type of an Array Element and a Holder Class

<table>
<thead>
<tr>
<th>XML data type of array element</th>
<th>Holder class name</th>
<th>CORBA data type of array element</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:int</td>
<td>AAA.ArrayOfIntHolder</td>
<td>AAA.ArrayOfIntnDHolder</td>
</tr>
<tr>
<td>xsd:unsignedInt</td>
<td>AAA.ArrayOfUnsignedIntHolder</td>
<td>AAA.ArrayOfUnsignedIntnDHolder</td>
</tr>
<tr>
<td>xsd:short</td>
<td>AAA.ArrayOfShortHolder</td>
<td>AAA.ArrayOfShortnDHolder</td>
</tr>
<tr>
<td>xsd:unsignedShort</td>
<td>AAA.ArrayOfUnsignedShortHolder</td>
<td>AAA.ArrayOfUnsignedShortnDHolder</td>
</tr>
<tr>
<td>xsd:long</td>
<td>AAA.ArrayOfLongHolder</td>
<td>AAA.ArrayOfLongnDHolder</td>
</tr>
<tr>
<td>xsd:float</td>
<td>AAA.ArrayOfFloatHolder</td>
<td>AAA.ArrayOfFloatnDHolder</td>
</tr>
<tr>
<td>xsd:double</td>
<td>AAA.ArrayOfDoubleHolder</td>
<td>AAA.ArrayOfDoublenDHolder</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>AAA.ArrayOfBooleanHolder</td>
<td>AAA.ArrayOfBooleannDHolder</td>
</tr>
<tr>
<td>xsd:string(1)</td>
<td>AAA.ArrayOfStringHolder</td>
<td>AAA.ArrayOfStringnDHolder</td>
</tr>
<tr>
<td>xsd:unsignedByte</td>
<td>AAA.ArrayOfUnsignedByteHolder</td>
<td>AAA.ArrayOfUnsignedByteDHolder</td>
</tr>
<tr>
<td>Structure (TypeCode type)</td>
<td>AAA.ArrayOfTypeCodeHolder</td>
<td>AAA.ArrayOfTypeCodeDHolder</td>
</tr>
<tr>
<td>Structure (Any type)</td>
<td>AAA.ArrayOfTypeCodeHolder</td>
<td>AAA.ArrayOfTypeCodeDHolder</td>
</tr>
</tbody>
</table>
### Mapping of XML and CORBA Data Types

<table>
<thead>
<tr>
<th>XML data type of array element</th>
<th>Holder class name</th>
<th>CORBA data type of array element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One-dimensional</td>
<td>n-dimensional</td>
</tr>
<tr>
<td>Structure</td>
<td>BBB.ArrayOfdataName SOAPGWTTypeHolder</td>
<td>BBB.ArrayOfdataName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOAPGWTTypeEnumDHolder</td>
</tr>
<tr>
<td>Enumeration</td>
<td>BBB.ArrayOfdataName SOAPEnumImplHolder</td>
<td>BBB.ArrayOfdataName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOAPEnumImplInDHolder</td>
</tr>
</tbody>
</table>

In a Holder class with an n-dimensional array, n represents 2, 3, 4, etc. depending on the number of dimensions.

**Example**

For a two-dimensional array of xsd:int, the Holder class name is AAA.ArrayOfInt2Dholder.

The supplied AAA package of the Holder class of the XML data type (other than structure or enumeration) is the same package as the Web service (a class in which the method is implemented).

In the following IDL definition, the package of the ArrayOfStringHolder class is fujitsu.sample.

```idl
module fujitsu {
    interface sample {
        typedef sequence <string> seq_string;
        seq_string op( in seq_string arg1 );
    };
}
```

The BBB package of the structure/ enumeration Holder class is the same as the package of the data-name SOAPGWTType class/data-name SOAPEnumImpl class.

**Note**

An RPC application that links with a CORBA application can handle array-type elements that are Java data types indicated as "basic data type," "enumeration type," "structure/exception type," "CORBA Object/ interface type," "TypeCode type," and "any type" but not the exception type.

### CORBA Object/Interface Type

The following table shows the correspondence between the CORBA Object and interface (interface) types and the XML data types.

For the Object and interface types of a linkage CORBA application, use java.lang.String in the in parameter/return value (return) and a YYY.StringHolder class in the out parameter/inout parameter.

**Table 8-25 Correspondence Between the CORBA Object and Interface (interface) Types**

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in/return</td>
<td>out/inout</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td></td>
<td>object</td>
<td>Object</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>YYY.StringHolder</td>
</tr>
<tr>
<td></td>
<td>interface</td>
<td>interface</td>
</tr>
</tbody>
</table>
**TypeCode Type**

The following table shows the correspondence between the CORBA TypeCode type and the XML data type.

For the TypeCode type of a linkage CORBA application, use WWW.TypeCode in the in parameter/return value (return) and a ZZZ.TypeCodeHolder class in the out parameter/inout parameter.

**Table 8-26 Correspondence Between the CORBA TypeCode Type and the XML Data Type**

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>corba:TypeCode</td>
<td><a href="http://WWW.TypeCode">WWW.TypeCode</a></td>
<td>ZZZ.TypeCodeHolder</td>
</tr>
</tbody>
</table>

"corba" in the above table represents name space "urn:xmlns-fujitsu-com:soap-corba-type2002-02."

**Any Type**

The following table shows the correspondence between any CORBA data type and the XML data type.

For any type of a linkage CORBA application, use WWW.Any in the in parameter/return value (return) and a ZZZ.AnyHolder class in the out parameter/inout parameter.

**Table 8-27 Correspondence Between any CORBA Data Type and the XML Data Type**

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corba:Any</td>
<td><a href="http://WWW.Any">WWW.Any</a></td>
<td>ZZZ.AnyHolder</td>
</tr>
</tbody>
</table>

"corba" in the above table represents name space "urn:xmlns-fujitsu-com:soap-corba-type2002-02."
Mapping of XML and EJB Data Types

This section details the mapping of Java data types to be used in RPC applications (RPC server and client applications) and XML data types.

Use the data types explained in this section to create an RPC application that links with an EJB application.

Note

- In an EJB application, parameters with the out/inout attribute cannot be used.
- For information on handling an XML data type in an RPC application, refer to Mapping of XML and Java Data Types.

Primitive Types

The following table shows the correspondence between the Java primitive types to be used in an EJB application and the XML data types.

Use a "Java data type in an RPC application" if the RPC application links with an EJB application that uses an "EJB data type."

Table 8-28  Correspondence Between EJB Java primitive types and the XML data types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>xsd:short</td>
<td>short</td>
<td>short</td>
</tr>
<tr>
<td>xsd:long</td>
<td>long</td>
<td>long</td>
</tr>
<tr>
<td>xsd:float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>xsd:double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>java.lang.String</td>
<td>string</td>
</tr>
<tr>
<td>xsd:unsignedByte</td>
<td>XXX.UnsignedByte</td>
<td>byte</td>
</tr>
</tbody>
</table>
Class and Exception Types

The following table shows the correspondence between the class and exception types to be used in an EJB application and the XML data types.

In an RPC application, use the "class-name SOAPGWType" class or "exception-name SOAPGWType" class for a class and exception of an EJB application to which a link is required.

### Table 8-29 Correspondence Between Class, EJB Exception Types and XML data types

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>class-name SOAPGWType</td>
<td>Class</td>
</tr>
<tr>
<td>Structure</td>
<td>exception-name SOAPGWType</td>
<td>Exception</td>
</tr>
</tbody>
</table>

**Note**
- For an EJB data-type class, only a member declared as public is mapped to a structure of an XML data type. No member with the same type name specified as the class type name is mapped. Only a class with a mappable member can be used.
- Only an EJB-data-type class with a default constructor can be used.
- To use a class type in an RPC client application, you need to provide conversion mapping of Java and XML data types.
- For information about creating an RPC application that uses the structure type, refer to Structure Type.
- For information about correspondence of conversion mapping of Java and XML data types, refer to Chapter 6, Installing a CORBA/SOAP Gateway.
- For information about creating an RPC application that uses the exception type, refer to Chapter 6, Installing a CORBA/SOAP Gateway.

**Example**

For EJB applications that define business methods, execute the J2EE Deployment tool with the following classes in the parameter

```java
package fujitsu.sample; // package name
public class Human
    implements java.io.Serializable {
    
};
```
The following class name + SOAPGWType class will be created.

```java
package fujitsu.sample;   // package name
public class HumanSOAPGWType {
    public String name;
    public int age;
}
```

**Array Type**

The following table shows the correspondence between the Java array type to be used in an EJB application and the XML data type.

The Java data type in a Web service will be the array type in the in parameter/return value (return).

<table>
<thead>
<tr>
<th>XML data type</th>
<th>Java data type in RPC application</th>
<th>CORBA data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>soapenc:Array</td>
<td>Array</td>
<td>Array</td>
</tr>
</tbody>
</table>

**Note**

An RPC application (that links with an EJB application) can handle array-type elements that are Java data types (indicated as “primitive type” and “class/exception type”, but not the exception type).
Chapter 9

Managing Web Service Information

This chapter describes the method of managing Web service information by Web Service Manager. Web service information can be managed in either of the following two methods:

- Using the Web service information edit tool (GUI)
- Using the `soapmodifydd` command.

Web service information is registered information on Web service that is referenced by the Interstage SOAP service when it calls Web service and the information is identified by the “Web Service Identifier” that is unique in the system.

Web service information is registered to the server and client systems each as required.

- Web service information to be registered in the server system
  
  Web service management information provided by the server system.
  
  Information to be registered includes the parameter name, data type, and attribute of the Web service and information for using the high reliability Web service function.

- Web service information to be registered in the client system
  
  Management information to be registered when communicating with the Web service of the server system by using the high-reliability Web service function.
  
  When the high reliability Web service function is not to be used, it need not be registered.
Chapter 9: Managing Web Service Information

Management Web Service Information by the GUI

Starting the Web Service Information Edit Tool

From the Interstage Management Console, select System | Tool | Web Service Information Edit Tool to display the top screen of the Web Service Information Edit Tool.

Top Screen of the WebService Information Edit Tool

Click the Server Function or Client Function button to perform registration selection.
Deployment in the Server System Environment

To register the Web service information, click the left button of Top. (To register Web service information of the RPC server application, click the **RPC Service** button.) In the following explanation, the remote procedure calling server application screen is used as an example.

For Reliable messaging (PUSH) and Reliable messaging (PULL) refer to Chapter 16, How to Use Reliable Messaging Function for Web Services (SOAP) in the Security System Guide.

Registering Web Service Information

Click the **RPC service** button. The screen on the following page appears.

Enter the class name, class file name, or JAR file name of an RPC server application and click the **Display WSDD** button. Web service information appears.
Remote Procedure Call Service Information Input Screen

- **Web Service Identifier**
  
  In the text box, type the Web Service Identifier with characters permitted in the URL (case sensitive). This name must be unique in this system.
When the **Display WSDD** button is clicked, a character string consisting of character string "urn:" followed by the Java class name of the RPC server application is displayed as the Web Service Identifier in the text box.

- **Processed HeaderElement**
  Set whether the processed SOAP header elements are to be deleted or not after a request/response message is received.
  The default is Do Not Delete.
  - Do not delete
  - Delete

- **Web service role (actor) name**
  Specify the Web service role (actor) name(s) in URI format.
  To describe two or more role (actor) names, delimit each name with a comma.
  The role (actor) name is a role name of the Web service in the transfer route when SOAP performs the specified transfer. Normally, it does not need to be specified.

- **Web service application class name**
  In the text box, type the Java class name of the RPC server application that generates Web service information. The class name must specify a class name with a package name.

- **Web service application file name**
  In the text box, type the full path of the class file or JAR file of the RPC server application. When specifying two or more file paths, delimit them with a comma (,). If the file path is not specified, the file path specified in the environment variable CLASSPATH is used.

- **Web service application: Display WSDD button**
  Click this button to initialize Web service information to the state of default information based on the specified Java class name, class file name, or JAR file name of the RPC server application.
  The methods displayed are only public and static methods. For the name space name and XML data type name, which are user-defined information, are represented by their default values.

  Clicking the **Display WSDD** button updates the following types of information.

  **Note**
  Security information is updated to the state where it is not specified.
  - Names of methods that are allowed to use basic service information (allowedMethods)
  - Whole method information
  - Whole user-defined information
  - While security information

- **Web service application life cycle**
  Select the life cycle of the RPC server application from the following three items.
  - Request
  - Session
  - Application
For details, see “scope attribute” in “parameter tag” in Appendix A.

- **Provider type**
  For the PRC server application, select the following provider type.
  - Remote Procedure Calling method Web service
  For details, see “provider attribute” in “service tag” in Appendix A.

- **Allowed method name (allowedMethods)**
  Enter the method name(s) that can be used as an RPC server application.
  To describe two or more method names, delimit each name by a comma.
  The default is an asterisk “*” (All methods can be used).
  When the **Display WSDD** button is clicked, the public and static methods belonging to the Java class of the RPC server application are displayed.

- **Add Method** button
  In the initial state, there is one method input field.
  To register two or more methods for one service, click the **Add Method** button. Each time this button is clicked, one method input field is added.

- **Method name**
  Enter the name of the method of the RPC server application that is called remotely.

- **Add Parameter** button
  In the initial state, there are three parameter input fields.
  Each time you click the **Add Parameter** button, you add one parameter input field.

- **Parameter name**
  Enter the parameter name of the method. The names that are initially displayed by the Display WSDD button are assigned serial numbers beginning from arg1.

- **Argument type**
  Select the parameter data type from the list.
  Select the data type with the local name of the data type of XML mapped to the parameter.
  If the parameter is an array type, select the local name of the data type of the array.

- **Array the number of dimensions**
  If the parameter is an array type, enter its number of dimensions.
  If the parameter is not an array type, specify zero.
  Note that the vector type cannot be specified as an array type. If “Vector” is specified as the argument type and a number other than zero is specified as the number of dimensions of the array, the number of dimensions is automatically set to zero after a warning is output.

- **Available User-defined type number**
  If the parameter is of an enumeration type or User-defined type, select the number of the User-defined type specified in the User-defined type information.
  For the User-defined type number, specify the number displayed in User-defined type Information.
• Parameter attribute
  Select the parameter attribute (in/out/inout). When the Display WSDD button is clicked, each attribute is displayed with either "in" or "inout" selected.

User-Defined Type Information

• Add Type button
  To add the User-defined type, enter the User-defined type class name and click Add Type button.
  The User-defined type is added to the list of the User-defined type information. The new type can now be used as a User-defined type by specifying the number displayed in User-defined type number used.

• Type
  Set the type of the User-defined type to be registered: enumeration type, structure or Bean type.

• User-defined type class name
  In the text box, type the class name of the user-defined type to be registered. The class name must specify a class name with a package name.
  The Java class, including the user-defined type, must be specified in the environment variable CLASSPATH. Alternatively, it must be specified in "Classpath" displayed at [WorkUnit setting] on the Interstage Management Console. To display [WorkUnit setting] on the Interstage Management Console, select [System] > [WorkUnits] > [MyIJSserver] (IJServer WorkUnit name), and then the [Settings] tab.

• Name space name
  Specify a name space name for the Qualified Name (XML data type name) of the User-defined type that you are registering. The default is “urn:Fujitsu-Soap-Service-Data”.

• XML data type name (user-defined XML data type name)
  Specify the Qualified Name (XML data type name) of the User-defined type.
  The name is the User-defined type class name with its period changed to a hyphen.

• Reset button
  Click the Reset button to delete User-defined type information. If the User-defined type to be deleted is used in a parameter, a warning is issued and deletion is canceled.

Security Information

Make settings for SOAP digital signature, and XML encryption (original assurance function).
For details, see Chapter 15, How to use XML Digital Signature and XML Encryption for Web Services(SOAP) in the Security System Guide.

Note
(1) User authentication of Security information cannot be used. Do not set it excluding "Invalid".
(2) Approval roll name of Security information cannot be used. Do not set it excluding "*".
(3) Entry of a return value is not required because it is automatically analyzed by the Web service container from the method of the RPC server application.
Entered Data Warning Screen

Click the Modify button to re-display the Web Service Information Edit Tool screen.

Web service identifier not specified.
No.1 method has no name so it will be ignored.
A method is not defined. Please define one method at least.

Since No.1 Candidate for encryption is not specified, a setup of No.1 detailed encryption information is ignored.
Since No.2 Candidate for encryption is not specified, a setup of No.2 detailed encryption information is ignored.
Since No.3 Candidate for encryption is not specified, a setup of No.3 detailed encryption information is ignored.

Since No.1 SOAP signature target is not specified, a setup of No.1 detailed signature information is ignored.
Since No.2 SOAP signature target is not specified, a setup of No.2 detailed signature information is ignored.
Since No.3 SOAP signature target is not specified, a setup of No.3 detailed signature information is ignored.
Web Service Information Input Screen.

On this screen the input fields are the ones with red frames.

Check input details. Click the Check button. The confirm registration of Web service information screen appears.
Confirm Registration of Web service Information Screen

To correct entry, click the **Modify** button (bottom right-hand corner). The Remote procedure call Web service input screen appears.

If there is no problem in the displayed information, click the **Deploy** button. The Web service information is registered. A registration complete screen appears.
Web Service Information Registration Complete Screen

Note: Click the Back button to return to the Top screen.
Displaying the Web Service Identifier

Click the **RPC Service List** button to display a list of the Web Service Identifiers for Remote procedure call service. The following Web Service Identifier list appears.

**Web Service Identifier Display Screen**

- **urn.sample-returnArray**
- **urn.sample-returnMonthly**
- **urn.sample-stringLength**
- **urn.sample-simpleheader**
- **urn.sample-countError**
- **urn.sample-convertName**
- **urn.sample-case**
- **urn.sample-yellowPerson**

Click the **Top** button at the left of the screen to return to the Top screen.
Displaying the Web Service Information

To display the Web service information of the Remote procedure call service already registered, click the Web Service Identifier of the desired Web service information from Web Service Identifier display screen.

Web Service Information Registered Content Confirmation Screen

The contents of the registered Web service information are displayed.

Click the Back button (lower left) to return the display to the Web Service Identifier display screen.
Updating Web Service Information

To update the Web service information of the already registered Remote procedure call service, display Web Service Information Registered Content Confirmation screen (from Web Service Identifier display screen). Click the **Update** button.

The Web Service Information Update screen appears. This screen works in the same way as the screen registering Web service information.
Web Service Information Update Screen

Note
(1) User authentication of Security information cannot be used. Do not set it excluding "Invalid".
(2) Approval roll name of Security information cannot be used. Do not set it excluding "***".
(3) Changing a registered Web Service Identifier is regarded as updating it; but as a new registration if the name to be changed is not yet registered.

**Deleting Web Service Information**

To delete the Web service information of the already-registered Remote procedure call service:

1. Go to Web Service Identifier display screen.
2. Display Web service information registered content confirmation screen.
3. Click the **Undeploy** button (bottom left).

The Web service information deletion screen is displayed.

**Web Service Information Deletion Confirmation Screen**

Click the **Back** button. The Web Service Information Registered Data Confirmation screen appears.

Click the **Undeploy** button to delete the Web service information. The Web Service Information Deletion Complete screen appears.
Web Service Information Deletion Complete Screen

Click the **Back** button to return the display to Web Service Identifier display screen.
Exiting the Editing of Web Service Information

To exit the editing of Web service information, be sure to click the **Select Servlet(Quit)** button on the left of Top screen. Clicking the **Select Servlet(Quit)** button ends editing and closes the Web Service Information Edit Tool.

Deployment in the Client System Environment

Select **Start | Program | Interstage | SOAP service | Web service information edit tool** to activate the Web Service Information Edit Tool.

![Interstage Web Service Information Edit Tool](image)

In the client system, make settings concerning the high-reliability Web service function. For details, refer to Chapter 15 SOAP Digital Signature and XML Encryption for Web Services (SOAP) and Chapter 16 How to use Reliable Messaging function for Web Services (SOAP) in the Security System Guide.

**Note**

The display of the Windows Start menu may slightly differ from system to system.
Managing Web Service Information with soapmodifydd


For details about the methods used to describe Web service information, refer to Appendix A.
Web Service Identifier and URL

Web Service Identifier and URL Specified in the Deployment of the Server System

Contents of the Web Service Identifier Specified in the Deployment of the Server System

The Web Service Identifier is a name for identifying the Web service information referenced by the Interstage SOAP service when the Web service is called. It must be unique in the server system. Describe the Web Service Identifier as shown below.

- Use characters that can be used for the path of the URL.
- The name must be unique in the server system (case sensitive).
- There is no limitation in the Web Service Identifier length.

URL of the Web Service Registered in Deployment

The URL of the Web service specified for server system deployment is specified as shown below. The URL will depend on the SOAP message exchanging method (Messaging or RPC method) employed. It also depends on whether XML encryption is used or not.

- Web service in Messaging method, or Web service in RPC method using XML encryption:
  The URL includes the Web Service Identifier specified in server system deployment.
  http://host-name/Web-application-name/services/Web-service-identifier
  - Example
  - Host name: MyHost
  - Web application name: soap
  - Web service identifier: myService
  - http://MyHost/soap/services/myService

- Web service in RPC method that does not use XML encryption

  Note
  The URL specified below is the servlet address, regardless of the Web service identifier specified in deployment of the server system.
  http://host-name/Web-application-name/servlet/servlet-name-of-Web-service-container
  - Example
  - Host-name: MyHost
  - Web-application-name: soap
  - servlet-name-of-Web-service-container: WSContainer
Web Service Identifier and URL

- http://MyHost/soap/servlet/WSContainer

**Web Service Identifier Specified by the Deployment of the Client System**

**Contents of the Web Service Identifier Specified in the Deployment of the Client System**

The Web Service Identifier is a name for identifying the Web service information referenced by the Interstage SOAP service when the application calls the Web service. It must be unique in the client system.

Describe the Web Service Identifier as shown below:
- The name must be unique in the client system (case sensitive).
- There is no limitation in the Web Service Identifier length.

**Web Service Identifier Described in the Messaging Method Application**

Specify the name space name specified in the child element of the SOAP body of the SOAP message to be transmitted as a Web Service Identifier.

**Web Service Identifier Described in the PRC Method Application**

Specify the name space name of the operation name that is the remote procedure call destination. The operation name is the value specified in the parameter of the setOperationName method of the cjavax.rpc.Call class.

The name space name of the operation name is the value of the “namespace” attribute of the “body” SOAP binding extension element of the “input” child element of the “binding” element of WSDL (urn:sample-rpc-service in the following example of WSDL).

- WSDL example

```xml
<bringing name="sampleBinding" type="sample">  
<operation name="opInt">  
<input>  
<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded namespace="urn:sample-rpc-service"/>  
</input>  
<output>  
<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded namespace="urn:sample-rpc-service"/>  
</output>  
</operation>  
</binding>
```
Chapter 10

Operating a Web Service

This chapter describes the operation procedure for the Web service container and the CORBA/SOAP gateway.
Web Service Operation Procedure

Use the Interstage Management Console to start or stop the Web service container.

From the console, select [System] > [WorkUnits] > [MyIJServed] (IJServed WorkUnit name) to display the Operation screen.

On this screen, click the Start button to open the IJServed WorkUnit in which the Web service container is deployed.

The Stop button closes the IJServed WorkUnit.
Operating the CORBA/SOAP Gateway

This section details operation of the CORBA/SOAP Gateway.

CORBA/SOAP Server Gateway

The CORBA/SOAP server gateway is an RPC server application. This section describes the procedure for registering the gateway with the Web Service Manager and deleting the CORBA/SOAP server from the Web Service Manager. It also describes the procedure used to display the information registered in the Web Service Manager.

Registering the CORBA/SOAP Server Gateway

Register the XML file of Web service information in the Web service information management by using the soapmodifydd command. The XML file is generated by the IDL compiler or from [EJB application setting] on the Interstage Management Console.

To display [EJB application setting], select [System] > [WorkUnit] > [MyIJServer] (IJServer WorkUnit name) > [Deployment] tab, and then [Detailed setting].

When the Interstage Management Console is used to generate the XML file of Web service information, the file is generated in the following directory:

- **Windows**
  
  %IS_HOME%\J2EE\var\deployment\ijserver\IJServer-workunit-name\soapgw

- **Solaris OE**
  
  /opt/FJSVj2ee/var/deployment/ijserver/ijServer-workunit-name/soapgw

- **Linux**

  /opt/FJSVj2ee/var/deployment/ijserver/ijServer-workunit-name/soapgw

**Example**

Web service information with the Web Service Identifier "ODsample::POAsample1" added is registered with the URL for the Web service container:  http://localhost:12000/soap_admin/servlet/WSContainer/

**Windows**

The file name of the Web service information is assumed to be "_intfSOAPGW_dd.xml" and the IJServer WorkUnit name is assumed to be "MyIJServer."

```
soapmodifydd -deploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ODsample::POAsample1
```

```
C:\Interstage\J2EE\var\deployment\ijserver\MyIJServer\soapgw\_intfSOAPGW_dd.xml
```
Chapter 10: Operating a Web Service

Solaris OE Linux

The file name of the Web service information is assumed to be "_intfSOAPGW_dd.xml" and it is located under the /tmp/ODsample directory.

```
soapmodifydd -deploy -url
http://localhost:12000/soap_admin/servlet/WSContainer
-urn ODsample::POAsample1
/opt/FJSVj2ee/var/deployment/ijserver/MyIJServer/soapgw/_intfSOAPGW_dd.xml
```

Note

- Before the registration of the CORBA/SOAP server gateway is effective, the IJServer WorkUnit in which the Web service container is deployed must be closed and restarted.
- Set the Web Service Identifier registered in the Web service information management (as shown below) depending on the application to be linked.
  - When linking with the CORBA or component transaction (TD) server application
    Set the same name as the naming service object name of the naming service object, in which the CORBA or component transaction (TD) server application is registered.
  - When linking with the EJB server application
    Specify the Enterprise Bean name as the Web Service Identifier.
    Deploy the EJB application in an IJServer WorkUnit whose type is other than "IJServer(Web+EJB[1VM])."
- If the naming service object name of the server application to be linked is placed under the naming context, specify it by delimiting the layers of the naming context with a slash "/".

Example

When linking with ObjectA that is registered under the naming context NamingContextA
-urn NamingContextA/ObjectA

- When the CORBA/SOAP server gateway and the CORBA/SOAP client gateway reference the same naming service, it is necessary to avoid duplication with the naming service object registered by the CORBA/SOAP client gateway.
  Take one of the following measures.
  - Change the naming service object name of the CORBA server application and register the changed name to Web Service Manager as the Web Service Identifier.
  - Change the naming service object name used by the CORBA client application and register the data conversion servant with the CORBA/SOAP client gateway by using the changed name.
Deleting the CORBA/SOAP Server Gateway

The registered information of the CORBA/SOAP server gateway can be deleted using the `soapmodifydd` command.

**Example**

The Web service information of the CORBA/SOAP server gateway having the service identification name "ODsample::POAsample1" that is registered with the URL of the Web service container http://localhost:12000/soap_admin/servlet/WSContainer" is deleted.

```
soapmodifydd -undeploy -url
http://localhost:12000/soap_admin/servlet/WSContainer
 -urn ODsample::POAsample1
```

Displaying the CORBA/SOAP Server Gateway

Information regarding the CORBA/SOAP server gateway registered with the Web service information management can be displayed using the `soapmodifydd` command.

**Example**

The `soapmodifydd` command first identifies the URL of the Web service container where the CORBA/SOAP server gateway is registered, and then its service identification name ("ODsample::POAsample1") is specified

```
soapmodifydd -query -url
http://localhost:12000/soap_admin/servlet/WSContainer
 -urn ODsample::POAsample1
```

The following Web service information for the CORBA/SOAP server gateway is displayed.

```
<ns2:service name="urn:ODsample::POAsample1" provider="ns1:RPC"
 xmlns:ns1="urn:xmlns-fujitsu-com:soapx-wsdd-providers-java-2002-09"
 xmlns:ns2="urn:xmlns-fujitsu-com:soapx-wsdd-2002-09">
 <ns2:operation name="op">
  <ns2:parameter name="arg1" type="tns:string"
   xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
  <ns2:parameter name="arg2" mode="inout" type="tns:string"
   xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
  <ns2:parameter name="arg3" mode="out" type="tns:string"
   xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
 </ns2:operation>
 <ns2:parameter name="allowedMethods" value="*"/>
 <ns2:parameter name="className" value="svStringArrayImpl"/>
 <ns2:parameter name="sendXsiTypes" value="true"/>
 <ns2:parameter name="sendMultiRefs" value="true"/>
 <ns2:parameter name="sendXMLDeclaration" value="true"/>
</ns2:service>
```
CORBA/SOAP Client Gateway

The CORBA/SOAP client gateway is a server application running on CORBA service.

This section describes the procedure for starting and stopping the CORBA/SOAP client gateway. It also describes the procedure for operating the data conversion servant (including registration with the CORBA/SOAP client gateway, deleting from it, and displaying the registered information).

Note
Start the CORBA service before starting the operation of the CORBA/SOAP client gateway.

Start the CORBA Service Before Starting the Operation of the CORBA/SOAP Client Gateway.

**Windows**

The procedure for starting and stopping the CORBA/SOAP client gateway is shown below.

- **Start**: Start the CORBA/SOAP client gateway in any of the following methods with the CORBA service started.
  - `soapgwstartclgw` command
  - `net` command
    - `net start "CORBA/SOAP ClientGW"`
  - From the service screen ([Control panel] - [Service]), select the following service and set the service state to “Start”:
    - SOAP service: CORBA/SOAP ClientGW service
- **Stop**
  - `soapgwstopclgw` command
  - `net` command
    - `net start "CORBA/SOAP ClientGW"`
  - Select the following service from the service screen ([Control panel] - [Service]) and set the service state to “Stop”:
    - SOAP service: CORBA/SOAP ClientGW service

**Solaris OE** **Linux**

To start and stop the CORBA/SOAP client gateway, execute the following commands under the “/opt/FJSVsoap/bin” directory.

- **Start**: `soapgwstartclgw` command
- **Stop**: `soapgwstopclgw` command
Operating the Data Conversion Servant

Operate the data conversion servant using the following commands.

- Register: `soapgwaddclgw` command
- Delete: `soapgwdelclgw` command
- Display: `soapgwlistclgw` command

Registering and Activating the Data Conversion Servant

The `soapgwaddclgw` command registers the conversion servant with the CORBA/SOAP client data and activates it.

This operation enables the linkage of the CORBA client application that uses the IDL definition to generate the data conversion servant and the RPC server application.

By using the `-replace` option of the `soapgwaddclgw` command, the data conversion servant registered with the same interface repository ID attached can be re-registered (registered information can be changed).

**Example**

The data conversion servant for the interface repository ID “IDL:ODsample/stringtest:1.0” is registered with the CORBA/SOAP client gateway and activated.

```
soapgwaddclgw  -id IDL:ODsample/stringtest:1.0  -n ODsample::stringtest
             -url http://host_name:80/soap_dev/servlet/WSContainer  -urn urn:ODsample::stringtest
```

Deleting the Data Conversion Servant

The `soapgwdelclgw` command inactivates the conversion servant and deletes it from the CORBA/SOAP client data. The data conversion servant is deleted after processing of all requests received from the CORBA client application is completed.

**Example**

The data conversion servant of the interface repository ID “IDL:ODsample/stringtest:1.0” is deleted from the CORBA/SOAP client gateway.

```
soapgwdelclgw  -id IDL:ODsample/stringtest:1.0
```
Displaying the Data Conversion Servant

The information of the data conversion servant registered with the CORBA/SOAP client gateway is displayed.

Example

The information for the data conversion servant of the interface repository ID “IDL:ODsample/stringtest:1.0” is displayed.

```
soapgwlistclgw
--------------------------------------------------
RepositoryID : IDL:ODsample/stringtest:1.0
ClassName    : ODsample._stringtestSOAPGWServant
--------------------------------------------------
```

Note

- Start the CORBA/SOAP client gateway before operating the data conversion servant (registration, deletion, and display).
- All Java class files (excluding the CORBA/SOAP server gateway) output by IDL compilation are required to be located in specific directories. If the directories are not specified with the environment variable (CLASSPATH), they are required to be in the following directories:

  **Windows**

  `%IS_HOME%\F3FMsoap\classes`

  **Solaris OE**

  `/opt/FJSVsoap/classes`

  **Linux**

  `/opt/FJSVsoap/classes`

- In order to add the directory that contains the data conversion servant for the environmental variable CLASSPATH, you need to specify it using the full path name and re-start the system.
- When adding the data conversion servant, the naming service registration name of the CORBA server application in operation (or to be operated) and the naming service name of the data conversion servant must not be the same.
With the Interstage SOAP service, the following types of data can be logged:

- 1000 to 1499: Web service container
- 1500 to 1999: Web Service Manager
- 2000 to 2999: CORBA/SOAP server gateway
- 3000 to 3999: CORBA/SOAP client gateway
- 6000 to 6999: High reliability Web service function related sources.

**Note:** The source of logged data can be classified by the message ID included in the log.

**Server Function Log**

The logs related to the Web service container, CORBA/SOAP server gateway, and high-reliability Web service can be referenced using the Interstage Management Console.

From the Interstage Management Console, select [System] > [WorkUnits] > [MyJServer] (JServer WorkUnit name) > [View Log] tab, then select the target log file at "Select log file." Information such as exceptions that occurred during operation can be referenced.

**Log Output Format**

**Windows**

[Date]: label: log type: message number: message contents
exception stack trace

**Example**


**Solaris OE**

[Date]: UX: label: log type: message number: message contents
exception stack trace

**Linux**

[Date]: UX: label: log type: message number: message contents
exception stack trace

**Example**


**Note**

There are messages that are also output to the system logs.
There are messages to which no stack trace is output.
Log Type

Table 10-1  Log Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Information</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error</td>
</tr>
</tbody>
</table>

Logs of the CORBA/SOAP Client Gateway

Logs of the CORBA/SOAP client gateway are always output to the log file. This output information includes operation logs and information about exceptions that have occurred.

Log Output Destination

**Windows**
- C:\Interstage\F3FMsoap\log\log.sv
- C:\Interstage\F3FMsoap\log\log.cl

**Solaris OE**  **Linux**
- /opt/FJSVsoap/log/log.sv
- /opt/FJSVsoap/log/log.cl

"log.sv" is a log file for the CORBA/SOAP client gateway and data conversion servant. In addition, "log.cl" is a log file of the client gateway operation command.

If the size of the log file exceeds 1M bytes, make a backup copy by adding the extension ".bak" to the file name. A backup file up to the previous generation can be created.

Log Output Format

**Windows**

[Date]: label: log type: message number: message contents
exception stack trace

Example

[2003/01/19 02:32:01:192]: ISSOAP: INFO: issoap3018: Client Gateway is start.

**Solaris OE**  **Linux**

[Date]: UX: label: log type: message number: message contents
exception stack trace
Example
[2003/01/19 02:32:01:192]: UX: ISSOAP: INFO: issoap3018: Client Gateway is start.

Note
There are messages that are also output to the system logs.
There are messages to which no stack trace is output.

Log Type

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Information</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error</td>
</tr>
</tbody>
</table>

Message Numbers
Logs output for the CORBA/SOAP client gateway can be sorted by message number. This is shown below.

Windows
- 3000 to 3099: CORBA/SOAP ClientGW service and soapgwstartclgw command
- 3100 to 3199: soapgwaddclgw command
- 3200 to 3299: soapgwdelclgw command
- 3300 to 3399: soapgwlistclgw command
- 3400 to 3499: the data conversion servant
- 3500 to 3599: soapgwstopclgw command

Solaris OE  Linux
- 3000 to 3099: soapgwstartclgw command
- 3100 to 3199: soapgwaddclgw command
- 3200 to 3299: soapgwdelclgw command
- 3300 to 3399: soapgwlistclgw command
- 3400 to 3499: the data conversion servant
- 3500 to 3599: soapgwstopclgw command
Logs of the Client Function

Though logs of the client function are not output in ordinary circumstances, they can be output by customizing the Web service container.

For further details, refer to the Setting the Logs of the Client Function in Customizing the Web Service Container.

Log Output Format

**Windows**

```
[Date]: label: log type: message number: message contents
exception stack trace
```

**Example**

An example of a message that is output to the log file (message.log) is shown below.

```
[2003/01/19 02:32:01:800]: ISSOAP: INFO: issoap3018: Client Gateway is start.
```

**Solaris OE  Linux**

```
[Date]: UX: label: log type: message number: message contents
exception stack trace
```

**Example**

```
[2003/01/19 02:32:800]: UX: ISSOAP: INFO: issoap3018: Client Gateway is start.
```

**Note**

There are messages that are also output to the system logs.

There are messages to which no stack trace is output.

Log Type

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Information</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error</td>
</tr>
</tbody>
</table>
Chapter 11

Sample Program Configuration
Sample Program Configuration

Sample programs provided by the Interstage SOAP service are placed in the following directories:

**Windows**

%IS_HOME%\F3FMsoap\examples

**Solaris OE  Linux**

/opt/FJSVsoap/examples

The overview of the sample programs is shown below.

<table>
<thead>
<tr>
<th>Table 11-1  RPC Method Sample Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
</tbody>
</table>
| Simple type (in)  
(Calc)  | int-type data is passed with the “in” parameters and return values. | Calc |
| Structure type (inout)  
(ConvertName)  | Structure type data is passed with “in” and “inout” parameters and return values. | ConvertName |
| Simple type (return)  
(CountRequest)  | The number of times the server program is called by the RPC client and the application is passed with return values by using session management. | CountRequest |
| Simple type one-dimensional array  
(ReturnArray)  | Int-type one-dimensional array data is passed with “in” parameters and return values. | ReturnArray |
| Enumeration type  
(ReturnMonth)  | Enumeration type data is passed with “in” and “inout” parameters and return values. | ReturnMonth |
| Basic data type  
(SimpleInout)  | int-type data is passed with “inout” and “out” parameters and return values. | SimpleInout |
| Simple type two-dimensional array  
(StringLength)  | A java.lang.String class and int-type two-dimensional data is passed with “in” parameters and return values. | StringLength |
| Structure type one-dimensional array  
(Young3Persons)  | Structure type one-dimensional array data is passed with “inout” parameters. | Young3Persons |
### Table 11-2  Sample Programs that Extend J2EE Component Models to Web Services

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Directory name</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJB class type (in)</td>
<td>EJB class type data is passed with “in” parameters and return values.</td>
<td>SampleEJB</td>
</tr>
<tr>
<td>(ConvertNameEJB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATEFUL EJB-SOAP linkage</td>
<td>A java.lang.String class is passed with “in” parameters and return values.</td>
<td>ConvertStrBean</td>
</tr>
<tr>
<td>(ConvertStrBean)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 11-3  Messaging Type Sample Programs

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Directory name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable Messaging</td>
<td>This function guarantees the arrival of the SOAP message.</td>
<td>Reliable</td>
</tr>
<tr>
<td>(Reliable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 11: Sample Program Configuration

Before Using Sample Programs

This section describes the necessary environment settings to use for sample programs. The explanation of the sample program execution procedure assumes that the Interstage SOAP service has already been running. The explanation also assumes that a sample program and a Web service container are running on the same machine.

Preparation

- Web server port number: 80
- Interstage Management Console port number: 12000
- Servlet name: WSContainer
- Interstage SOAP service installation directories:
  - Windows
    - C:\Interstage\F3FMsoap
  - Solaris OE, Linux
    - /opt/FJSVsoap
- WorkUnit name: MyIJServer
- WorkUnit type: Operation of Web application and EJB application

Windows

Web application name of Web service container: /soap_dev (File "C:\Interstage\F3FMsoap\war\soap_dev.war" is deployed in WorkUnit "MyIJServer")

Solaris OE, Linux

Web application name of Web service container: /soap_dev (File "/opt/FJSVsoap/war/soap_dev.war" is deployed in WorkUnit "MyIJServer")

- On the Interstage Management Console, select [System] > [WorkUnits] > [MyIJServer] > [MyIJServer] (UServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. For "Classpath" at [WorkUnit Settings], specify the current directory ("."), and the following directories, then start IJServer WorkUnit "MyIJServer."

Note

The same "Classpath" must be specified to the environment variable CLASSPATH in the client system.
Before Using Sample Programs

Windows
C:\Interstage\F3FMsoap\examples

Linux  Solaris OE
/opt/FJSVsoap/examples

- URL of the Web service container: http://localhost:80/soap_dev/servlet/WSContainer
  http://localhost:12000/soap_admin/servlet/WSContainer
- SSL communication function: Unused
- High reliability Web service function: Unused (except some sample programs).
RPC Method Sample Programs

This section describes the procedure for executing RPC method sample programs. It provides details about the description of the URL specified as the argument of the client program when XML encryption is used. When not used, refer to Web Service Identifier and URL Specified in the Deployment of the Server System.

Sample Program Calc

The sample program Calc (sample Calc) performs addition by using simple-type (int) “in” parameters and a return value (return) and displays the result.

Process Overview

- Sample Calc server program
  This program has one method.
  - int add(int arg1, int arg2)
    This method receives two int-type “in” parameters (arg1 and arg2), and returns its sum with a return value (return).

- Sample Calc client program
  With the two values entered from the standard input as “in” parameters, this program calls the “add” method of the sample Calc server program.
  It receives the process result of the sample Calc server program (the sum of the two values) as a return value and displays it on the standard output.

Sample Calc Execution Procedure

1. Move to the following directory.

   Windows
   C:\Interstage\F3FMsoap\examples\Calc\server

   Solaris OE  Linux
   /opt/FJSVsoap/examples/Calc/server

2. Register the Web service information of the sample Calc to Web Service Manager.

   soapmodifydd -set -url
   http://localhost:12000/soap_admin/servlet/WSContainer
   DeploymentDescriptor.xml
3. Move to the following directory.

   **Windows**
   
   C:\Interstage\F3FMsoap\examples\Calc\client

   **Solaris OE**  **Linux**
   
   /opt/FJSVsoap/examples/Calc/client

4. Execute the client program of the sample Calc.

   ```
   java CalcClient http://localhost:80/soap_dev/servlet/WSContainer
   1. In response to the message “Please input first number,” enter the first number.
   2. In response to the message “Please input second number,” enter the second number.
   ```

### Sample Calc Execution Example

```java
java CalcClient http://localhost:80/soap_dev/servlet/WSContainer
Please input first number: 10
Please input second number: 20
10 + 20 = 30
```

### Sample Program ConvertName

The sample program ConvertName (sample ConvertName) converts the entered character strings (the names of persons) into reversed character strings or upper-case letter character strings using the structure for “in” and “inout” parameters and a return value (return).

The sample program ConvertName defines the following class as a structure.

- **ConvertName.data.HumanName**
  - The ConvertName.data.HumanName class has two java.lang.String class fields representing the names firstName and secondName.

### Process Overview

- **Sample ConvertName server program**

  This program has two methods.
  - **ConvertName.data.HumanName reverse(HumanName name)**
    
    This method receives the instance (HumanName object) of the structure type ConvertName.data.HumanName as the “in” parameter “name.”
    
    It creates a HumanName object that reverses the character strings of the firstName and secondName fields of the HumanName object and returns it as a return value.
Example
Received structure "name" (HumanName object)
firstName field: taro
secondName field: fujitsu
is converted into
Returned structure (HumanName object)
firstName field: orat
secondName field: ustijuf

- void toUpperCase(ConvertName.data.HumanName name)
  This method receives the instance (HumanName object) of the structure type
  ConvertName.data.HumanName as the "inout" parameter "name."
  It converts the character strings of the firstName and secondName fields of the HumanName
  object into character strings written in upper-case letters.
Example
Received structure "name" (HumanName object)
firstName field: taro is converted into TARO.
secondName field: fujitsu is converted into FUJITSU.

- Sample ConvertName client program

  1. This program creates a HumanName object from the character strings (first and second names)
     entered from the standard input and calls the "reverse" method of the sample ConvertName server
     program with the "in" parameter structure "name."
     It receives the process result of the sample ConvertName server program (character strings of
     reversed names) as a return value and displays it on the standard output.

  2. It calls the toUpperCase method of the sample ConvertName server program with the same
     parameters as shown above.
     It receives the process result for the sample ConvertName server program (name character strings
     converted into upper-case letter character strings) with the "inout" parameter structure "name" and
     displays it on the standard output.

Sample ConvertName Execution Procedure

  1. Move to the following directory.

  Windows
  C:\Interstage\F3FMsoap\examples\ConvertName\server

  Solaris OE Linux
  /opt/FJSVsoap/examples/ConvertName/server
2. Register the Web service information of the sample ConvertName to Web Service Manager.

```
soapmodifydd -set -url
http://localhost:12000/soap_admin/servlet/WSContainer
DeploymentDescriptor.xml
```

3. Move to the following directory.

- **Windows**
  
  `C:\Interstage\F3FMsoap\examples\ConvertName\client`

- **Solaris OE**
  
  `/opt/FJSVsoap/examples/ConvertName/client`

4. Execute the client program of the sample ConvertName.

```
java ConvertNameClient http://localhost:80/soap_dev/servlet/WSContainer
```

**Sample ConvertName Execution Example**

```
java ConvertNameClient http://localhost:80/soap_dev/servlet/WSContainer
Please input first name: taro
Please input second name: fujitsu
reverse = orat ustijuf
to upper case = TARO FUJITSU
```
Sample Program CountRequest

The sample program CountRequest (sample CountRequest) displays the cumulative value for the number of times the server program is called during the use of the session management function using a simple type (long) return value (return).

Process Overview

- Sample CountRequest server program
  This program has one method.
  - public long count()
    This method returns the cumulative value of the number of times the server program was called with a long-type return value (return).

- Sample CountRequest client program
  This program calls the “count” method of the sample CountRequest server program ten times. It receives the process result of the sample CountRequest server program (the cumulative value of the number of times the server program was called) and displays it on the standard output.

Sample CountRequest Execution Procedure

1. Move to the following directory.
   - Windows
     C:\Interstage\F3FMsoap\examples\CountRequest\server
   - Solaris OE, Linux
     /opt/FJSVsoap/examples/CountRequest/server

2. Register the Web service information of the sample CountRequest to Web Service Manager.
   soapmodifydd -set -url
   http://localhost:12000/soap_admin/servlet/WSContainer
   DeploymentDescriptor.xml

3. Move to the following directory.
   - Windows
     C:\Interstage\F3FMsoap\examples\CountRequest\client
Sample Program ReturnArray

The sample program ReturnArray (sample ReturnArray) performs multiplication using simple (int) type array “in” parameters and return values (return) and displays the result.

Process Overview

- Sample ReturnArray server program
  
  This program has one method.
  
  - int[] returnArray(int[] intArray)

  This method receives an “int” type array with the “in” parameter intArray.

  It doubles the value of each element of intArray and returns the result as return values (return).

- Sample ReturnArray client program.

  This program assigns the values entered from the standard input to the “int” type array intArray (there are three elements in the array) and calls the returnArray method of the sample ReturnArray server program.

  It receives the process result of the sample ReturnArray server program (the doubled values of the elements in the array) as return values and displays the result on the standard output.

Sample ReturnArray Execution Procedure

1. Move to the following directory.

   Windows

   C:\Interstage\F3FMsoap\examples\ReturnArray\server
Chapter 11: Sample Program Configuration

1. In response to the message “Please input a number-0:”, enter the first number.
2. In response to the message “Please input a number-1:”, enter the second number.
3. In response to the message “Please input a number-2:”, enter the third number.

Sample ReturnArray Execution Example

```
java ReturnArrayClient http://localhost:80/soap_dev/servlet/WSContainer
```

```
Please input a number-0: 10
Please input a number-1: 20
Please input a number-2: 30
10 * 2 = 20
20 * 2 = 40
30 * 2 = 60
```
Sample Program ReturnMonth

The sample program ReturnMonth (sample ReturnMonth) displays the name of the month (January, etc.) corresponding to the entered number (1 to 12) or the number or name of the next month. It uses the Enumeration type “in” and “inout” parameters and return values (return).

The following two classes are defined as classes for indicating enumeration.

- **ReturnMonth.data.MonthName**
  This is an enumeration class that indicates “the name of a month.”
  The class can have 12 java.lang.String values that represent months such as “January,” “February,” … , “December.”

- **ReturnMonth.data.MonthNumber**
  This is an enumeration class that indicates the “number of the month”.
  The class can have 12 int values that represent months such as 1, 2, … 12.

Process Overview

- **Sample ReturnMonth server program**

  This program has two methods.

  - ReturnMonth.data.MonthName returnMonthName(ReturnMonth.data.MonthNumber number)
    This method receives the MonthNumber object as the “in” parameter “number” and returns the corresponding MonthName object as a return value.

  - void nextMonthNumber(ReturnMonth.data.MonthNumber number)
    This method receives the MonthNumber object as the “inout” parameter “number” and converts the parameter “number” into the MonthName object that indicates the next month.

- **Sample ReturnMonth client program**

  1. This program sets the int value between 1 and 12 entered from the standard input to the instance of the ReturnMonth.data.MonthNumber class (MonthNumber object). With this value as an “in” parameter, it calls the returnMonthName method of the sample ReturnMonth server program.

     It receives the process result of the sample ReturnMonth server program (the corresponding MonthName object) as a return value and displays it on the standard output.

  2. This program calls the nextMonthNumber method of the sample ReturnMonth server program with the same parameter as Step 1 above.

     It receives the process result for the sample ReturnMonth server program (the MonthNumber object indicating the next month) and displays it on the standard output.

  3. With the MonthNumber object received in Step 2 above as an “in” parameter, this program calls the returnMonthName method of the sample ReturnMonth server program.

     It receives the process result of the sample ReturnMonth server program (the corresponding MonthName object) as a return value and displays it on the standard output.
Sample ReturnMonth Execution Procedure

1. Move to the following directory.

   - **Windows**
     
     \(\text{C:}\text{:Interstage}\text{:F3FMsoap}\text{:examples}\text{:ReturnMonth}\text{:server}\)
   
   - **Solaris OE**  
     
     \(/\text{opt}\text{:FJSVsoap}\text{:examples}\text{:ReturnMonth}\text{:server}\)

2. Register the Web service information of the sample ReturnMonth to Web Service Manager.

   ```
   soapmodifydd -set -url
   http://localhost:12000/soap_admin/servlet/WSContainer
   DeploymentDescriptor.xml
   ```

3. Move to the following directory.

   - **Windows**
     
     \(\text{C:}\text{:Interstage}\text{:F3FMsoap}\text{:examples}\text{:ReturnMonth}\text{:client}\)
   
   - **Solaris OE**  
     
     \(/\text{opt}\text{:FJSVsoap}\text{:examples}\text{:ReturnMonth}\text{:client}\)

4. Execute the client program of the sample ReturnMonth

   ```
   java ReturnMonthClient http://localhost:80/soap_dev/servlet/WSContainer
   ```

   1. In response to the message “Please input a number (1-12):”, enter a number between 1 and 12.

Sample ReturnArray Execution Example

   ```
   java ReturnMonthClient http://localhost:80/soap_dev/servlet/WSContainer
   Please input a number(1-12): 5
   Name of the month = May
   Number of the next month = 6
   Name of the month = June
   ```
Sample Program SimpleInout

The sample program SimpleInout (sample SimpleInout) performs multiplication by using simple type (int) “inout” and “out” parameters and return values (return) and displays the result.

Process Overview

- Sample SimpleInout server program
  This program has one method.
  - int calculate(IntHolder inout_int, IntHolder out_int)
    This method takes out a value from the first “inout” parameter “inout_int,” multiplies it by two and sets it again, multiplies it by three and sets it to the second “out” parameter “out_int.”. It then returns a value multiplied by -1 as a return value.

- Sample SimpleInout client program
  This program sets the value entered from the standard input to the “inout” parameter “inout_int” and calls the “calculate” method of the sample SimpleInout server program.
  It receives the process result of the sample SimpleInout server program (a value multiplied by -1 as a return value, a value multiplied by two as the “inout” parameter “inout_int,” and a value multiplied by three as the “out” parameter “out_int”) and displays the result on the standard output.

Sample SimpleInout Execution Procedure

1. Move to the following directory.

   **Windows**
   C:\Interstage\F3FMsoap\examples\SimpleInout\server

   **Solaris OE  Linux**
   /opt/FJSVsoap/examples/SimpleInout/server

2. Register the Web service information of the sample SimpleInout to Web Service Manager.

   soapmodifydd -set -url
   http://localhost:12000/soap_dev/servlet/WSContainer
   DeploymentDescriptor.xml

3. Move to the following directory.

   **Windows**
   C:\Interstage\F3FMsoap\examples\SimpleInout\client
Chapter 11: Sample Program Configuration

4. Execute the client program of the sample SimpleInout

```java
java SimpleInoutClient http://localhost:80/soap_dev/servlet/WSContainer
```

In response to the message "Please input a number:", enter a number.

Sample ReturnArray Execution Example

```java
java SimpleInoutClient http://localhost:80/soap_dev/servlet/WSContainer
Please input a number: 10
10 x (-1) = -10
10 x 2 = 20
10 x 3 = 30
```

Sample Program StringLength

The sample program StringLength (sample StringLength) displays the lengths of the individual strings of the array elements using a java.lang.String class two-dimensional array “in” parameter. In addition, it uses a simple (int) type two-dimensional array return value (return).

Process Overview

- Sample StringLength server program
  This program has one method.
  - `public int[][] returnLength(java.lang.String[][] stringArray)`
    This method receives a java.lang.String class two-dimensional array as the “in” parameter stringArray.
    It returns an “int” type two-dimensional array having stringArray elements and their character string lengths as elements.

- Sample StringLength client program
  Before calling the returnLength method of the sample StringLength server program, this program creates a java.lang.String class two-dimensional array to which character strings are set. It then displays the contents on the standard output.

  The program sets the above java.lang.String two-dimensional array to the “in” parameter stringArray and calls the returnLength method of the sample StringLength server program.

  It then receives the process result of the sample StringLength server program (an array indicating the lengths of character strings) and displays it on the standard output.
**Sample StringLength Execution Procedure**

1. Move to the following directory.
   - **Windows**
     
     C:\Interstage\F3FMsoap\examples\StringLength\server
   - **Solaris OE**
     
     /opt/FJSVsoap/examples/StringLength/server
   - **Linux**
     
     /opt/FJSVsoap/examples/StringLength/server

2. Register the Web service information of the sample StringLength to Web Service Manager.

   ```sh
   soapmodifydd -set -url
   http://localhost:12000/soap_admin/servlet/WSContainer
   DeploymentDescriptor.xml
   ```

3. Move to the following directory.
   - **Windows**
     
     C:\Interstage\F3FMsoap\examples\StringLength\client
   - **Solaris OE**
     
     /opt/FJSVsoap/examples/StringLength/client
   - **Linux**
     
     /opt/FJSVsoap/examples/StringLength/client

4. Execute the client program of the sample StringLength

   ```sh
   java StringLengthClient http://localhost:80/soap_dev/servlet/WSContainer
   ```

**Sample StringLength Execution Example**

```sh
java StringLengthClient http://localhost:80/soap_dev/servlet/WSContainer
String:
blue, green, red
black, gold, silver
length:
4, 5, 3
5, 4, 6
```
Sample Program Young3Persons

The sample program Young3Persons (sample Young3Persons) displays the names and ages of the youngest three persons out of the five in ascending order of age using a structure type array as an “inout” parameter.

As a class indicating a structure, the following class is defined:

- Young3Persons.data.Person
  The Young3Persons.data.Person class has two fields: a java.lang.String type field indicating names named “name” and a com.fujitsu.interstage.soapx.types.UnsignedInt type field indicating ages named “age.”

Process Overview

- Sample Young3Persons server program
  This program has one method.
  - public void getYoung3Persons(ArrayHolder personsHolder)
    This method receives a one-dimensional array of instances (Person objects) of the Young3Persons.data.Person class as the “inout” parameter personsHolder.
    It rearranges the elements, Person objects, in ascending order in the “age” field and sets the array that has the top three Person objects as its elements to the “inout” parameter personsHolder again.

- Sample Young3Persons client program
  This program creates an array of Person objects (length: 5) from the values entered from the standard input (names and ages of five persons).
  It sets the array of Person objects to the “inout” parameter personsHolder and calls the getYoung3Persons method for the sample SimpleInout server program.
  It also receives the process result of the sample Young3Persons server program (the top three Person objects that are arranged in ascending order in the “age” field) and displays it on the standard output.

Sample Young3Persons Execution Procedure

1. Move to the following directory.

   Windows
   C:\Interstage\F3FMsoap\examples\Young3Persons\server

   Solaris OE Linux
   /opt/FJSVsoap/examples/Young3Persons/server
2. Register the Web service information of the sample Young3Persons to Web Service Manager.

```
soapmodifydd -set -url
http://localhost:12000/soap_admin/servlet/WSContainer
DeploymentDescriptor.xml
```

3. Move to the following directory.

```
Windows
C:\Interstage\F3FMsoap\examples\Young3Persons\client

Solaris OE Linux
/opt/FJSVsoap/examples/Young3Persons/client
```

4. Execute the client program of the sample Young3Persons

```
java Young3PersonsClient http://localhost:80/soap_dev/servlet/WSContainer
response to the message “Please input the name of person-x:”, enter the name.
response to the message “Please input the age of person-x:”, enter the name.
Repeat the above entry (name and age) five times.
```

**Sample Young3Persons Execution Example**

```
java Young3PersonsClient http://localhost:80/soap_dev/servlet/WSContainer
Please input the name of person-1: Jiro
Please input the age of person-1: 18
Please input the name of person-2: Goro
Please input the age of person-2: 12
Please input the name of person-3: Saburo
Please input the age of person-3: 17
Please input the name of person-4: Shiro
Please input the age of person-4: 15
Please input the name of person-5: Ichiro
Please input the age of person-5: 20
Young 3 persons:
[name=Goro, age=12]
[name=Shiro, age=15]
[name=Saburo, age=17]
```
Sample Program that extends PlusJ2EE Component Models to Web Services

This section describes the procedure used for executing sample programs to link with J2EE component models.

Sample Program ConvertNameEJB

The sample program ConvertNameEJB (sample ConvertNameEJB) reverses the entered character string (the name of a person) by using the EJB class type as “in” parameters and return values (return).

The sample program ConvertNameEJB defines the following class as an EJB class type.

- ConvertName.data.HumanData
  
  The ConvertName.data.HumanData class has two java.lang.String class fields representing the names firstName and secondName.

Process Overview

- EJB server program
  
  This program has one method.
  
  - ConvertName.data.HumanData reverse(HumanData name)
    
    This method receives the instance (HumanData object) in the EJB class type ConvertName.data.HumanData as the “in” parameter “name.”
    
    It creates a HumanData object in which the character strings of the firstName and secondName fields of the HumanData object are reversed and returns them as return values.

Example

- Received EJB class type “name” (HumanData object)
  
  firstName: taro
  
  secondName field: fujitsu
  
  is converted into
  
  Returned EJB class type (HumanData object)
  
  firstName field: orat
  
  secondName field: ustijuf

- Sample ConvertNameEJB client program
  
  This program creates a HumanData object from the character strings entered from the standard input (first and second names) and calls the “reverse” method of the EJB server application with the “in” parameter EJB class type “name.”
  
  It receives the process result of the EJB server application (reversed name character strings) as return values and displays them on the standard output.
Sample Program that extends PlusJ2EE Component Models to Web Services

Sample ConvertNameEJB Execution Procedure

**Windows**

- Developing the EJP application
  1. On the Interstage Management Console, select [System] > [WorkUnits], and [Create a new WorkUnit].
  2. Specify SOAPSAMPLE for "WorkUnit Name" and specify "EJB Applications Only" for "IJSERVER Type" then click the Create button (creating an IJSERVER WorkUnit).
  3. Select [WorkUnits] > [SOAPSAMPLE] (IJSERVER WorkUnit name), and [Deployment] tab.
  4. Specify "C:\Interstage\F3FMsoap\examples\SampleEJB\SampleEJB.jar" for "Deployment File."
  5. Select [Detailed Settings] and [EJB Application Settings].
  6. Select the SOAP check box and click the Deploy button.
  7. Select [WorkUnits] > [MyIJSERVER] (IJSERVER WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. At [WorkUnit Settings], add the following setting to "Java VM Options" and click the Update button:
     -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
  8. Select [MyIJSERVER] (IJSERVER WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. Add "C:\Interstage\J2EE\var\deployment\ijserver\SOAPSAMPLE\soapgw\SampleEjb" to "Classpath" in [WorkUnit Settings], and click the Update button.
  9. Select [WorkUnits] > [MyIJSERVER] (IJSERVER WorkUnit name), and [Status] tab, then click the Start button.

- Registering the CORBA/SOAP server gateway:
  1. Move to the following directory.
     C:\Interstage\J2EE\var\deployment\ijserver\SOAPSAMPLE\soapgw\SampleEjb\ConvertName\server
  2. Register the Web service information of the sample program ConvertName to Web Service Manager.

     soapmodifydd -deploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertNameServer _ConvertNameServerRemoteSOAPGW_dd.xml

- Executing the sample ConvertNameEJB client program:
  1. Move to the following directory.
     C:\Interstage\F3FMsoap\examples\SampleEJB\ConvertName\client
  2. Add the class path.
     set classpath=%classpath%;C:\Interstage\F3FMsoap\examples\SampleEJB
  3. Execute the client program of the sample ConvertNameEJB

     java ConvertNameClient http://localhost:80/soap_dev/servlet/WSContainer

In response to the message “Please input first name:”, enter the first name.
Chapter 11: Sample Program Configuration

In response to the message “Please input second name:”, enter the second name.

- Deleting the registered information (execute this operation after the EJB application is terminated):
  1. Delete the Web service information registered.

```bash
soapmodifydd -undeploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertNameServer
```

2. From the Interstage Administration Console, select [System] > [WorkUnits] > [SOAPSAMPLE] (IJServer WorkUnit name), and [Status] tab.

3. Click the Stop button of the IJServer WorkUnit "SOAPSAMPLE" to stop the IJServer WorkUnit. Then click the Delete button for deletion.

**Note**
The EJB applications registered in SOAPSAMPLE are also deleted.

**Sample ConvertNameEJB Execution Procedure**

- Developing the EJP application

  **Solaris OE**  **Linux**

1. From the Interstage Management Console, select [System] > [WorkUnits], and [Create a new WorkUnit].

2. Specify SOAPSAMPLE for "WorkUnit Name" and specify "EJB Applications Only" for "IJServer Type," then click the Create button (creating an IJServer WorkUnit).

3. Select [WorkUnits] > [Status] tab > [SOAPSAMPLE] (IJServer WorkUnit name), and [Deployment] tab.

4. Specify "/opt/FJSVsoap/examples/SampleEJB/SampleEJB.jar" for "Deployment File" or click the Browse button and select "/opt/FJSVsoap/examples/SampleEJB/SampleEJB.jar."

5. Select [Detailed Settings] and then [EJB Application Settings].

6. Select the SOAP check box at "EJB Application Settings" and click the Deploy button.

7. Select [WorkUnits] > [MyIJServer] (IJServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. At [WorkUnit Settings], add the following setting to "Java VM Option" and click the Update button:

   `-Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient`

8. Select [MyIJServer] (IJServer WorkUnit name), [Settings] tab, and [WorkUnit Settings]. Add "/opt/FJSVJ2ee/var/deployment/ijserver/SOAPSAMPLE/soapgw/SampleEJB" to "Classpath" in [WorkUnit Settings], and click the Update button.

9. Select [WorkUnits] > [MyIJServer] (IJServer WorkUnit name), and [Status] tab, then click the Start button.
• Registering the CORBA/SOAP server gateway
  (1) Move to the following directory.
  /opt/FJSVj2ee/var/deployment/ijserver/SOAPSample/soapgw/SampleEJB/ConvertName/server
  (2) Register the Web service information of the sample program ConvertName to Web Service Manager.

  soapmodifydd -deploy -url
  http://localhost:12000/soap_admin/servlet/WSContainer -urn
  ConvertNameServer _ConvertNameServerRemoteSOAPGW_dd.xml

• Executing the sample ConvertNameEJB client program:
  1. Move to the following directory.
     /opt/FJSVsoap/examples/SampleEJB/ConvertName/client
  2. Add the class path.
     CLASSPATH=$CLASSPATH:/opt/FJSVsoap/examples/SampleEJB
     export CLASSPATH
  3. Activate the sample ConvertNameEJB client program.

     java ConvertNameClient http://localhost:80/soap_dev/servlet/WSContainer

     In response to the message “Please input first name:”, enter the first name.
     In response to the message “Please input second name:”, enter the second name.

• Deleting the registered information (Execute this operation after the EJB application is terminated.)
  1. Delete the Web service information registered.

     soapmodifydd -undeploy -url
     http://localhost:12000/soap_admin/servlet/WSContainer -urn
     ConvertNameServer

  2. From the Interstage Management Console, select [System] > [WorkUnits] > [SOAPSample]
     (IJServer WorkUnit name), and [Status] tab.
  3. Click the Stop button of the IJServer WorkUnit "SOAPSample" to stop the IJServer WorkUnit. Then
     click the Delete button for deletion.

Note
The EJB applications registered in SOAPSample are also deleted.

Sample ConvertNameEJB Execution Example

  java ConvertNameClient http://localhost:80/soap_dev/servlet/WSContainer
  Please input first name: taro
  Please input second name: fujitsu
  reverse = orat ustijiuf
Sample Program ConvertStrBean

The sample program ConvertStrBean (sample ConvertStrBean) converts lower-case letters of the entered character string (the name of a person) into upper-case letters. Alternatively, it reverses the character string by using java.lang.String as an “in” parameter and a return value (return).

Process Overview

- **EJB server program**
  This program uses the following methods:
  - java.lang.String reverse(java.lang.String name)
    This method receives a “java.lang.String” type as the “in” parameter “name.”
    It reverses the character string received with “name” and returns it as a return value (return) in java.lang.String type.
  - java.lang.String toupper(java.lang.String name)
    This method receives a “java.lang.String” type as the “in” parameter “name.”
    It converts the character string received with “name” into an upper-case letter character string and returns it as a return value (return) in java.lang.String type.

**Example**

Character string of the “java.lang.String” type “name” received
secondName field: fujitsu
is converted into
Character string of the java.lang.String type “return” to be returned
firstName field: ustijuf

**Example**

Character string of the “java.lang.String” type “name” received
secondName field: fujitsu
is converted into
Character string of the java.lang.String type “return” to be returned
firstName field: FUJITSU

- **Sample ConvertStrBean client program**
  For the EJB server program, this program sets the character string received from the standard input to a java.lang.String type “in” parameter. It then calls the “reverse” and “toupper” methods of the EJB server program, and displays the respective return values.
Sample ConvertStrBean Execution Procedure

Windows

- Developing the EJB application
  1. On the Interstage Management Console, select [System] > [WorkUnits], and [Create a new Server].
  2. Specify SOAPSamp for "WorkUnit Name" and specify "EJB Application Only" for "IJServer Type," then click the Create button (creating an IJServer WorkUnit).
  3. Select [System] > [WorkUnits] > [SOAPSamp] (IJServer WorkUnit name) and [Deployment] tab.
  4. Specify "/opt/FJSVsoap/examples/ConvertStrBean/ConvertStr.jar" for "Deployment file."
  5. Select [Detailed Settings] and then [EJB Application Settings].
  6. Select the SOAP check box at "EJB Application Settings" and click the Deploy button.
  7. Select [MyIJServer] (IJServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. At [WorkUnit Settings], add the following setting to "Java VM option" and click the Update button:
     -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
  8. Select [MyIJServer] (IJServer WorkUnit name), [Settings] tab, and [WorkUnit Settings]. Add "/opt/FJSVJ2ee/var/deployment/ijserver/SOAPSamp/soapgw/ConvertStr" to "Classpath" in [WorkUnit Settings], and click the Update button.
  9. Select [WorkUnit] > [MyIJServer] (IJServer WorkUnit name), and [Status] tab, then click the Start button.

- Registering the CORBA/SOAP server gateway
  1. Move to the following directory.
     C:\Interstage\J2EE\var\deployment\ijserver\SOAPSample\soapgw\ConvertStr\ConvertStr
  2. Register the Web service information of the sample ConvertStrBean to Web Service Manager.
     soapmodifyydd -deploy -url
     http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertStrBean
     _ConvertStrBeanRemoteSOAPGW_dd.xml

- Executing the sample ConvertStrBean client program
  1. Move to the following directory:
     C:\Interstage\F3FMsoap\examples\ConvertStrBean\ConvertStrClient
  2. Add the class path.
     set classpath=%classpath%;C:\Interstage\F3FMsoap\examples\ConvertStrBean
  3. Execute the client program for the sample ConvertNameEJB
     java ConvertNameClient  java ConvertStrClient
     http://localhost:80/soap_dev/servlet/WSContainer

In response to the message "Please input soap method <reverse|toupper|remove> :", enter the method name to be called.
If “reverse” or “toupper” is entered, the message “Please input data <String|null>:” is displayed. Enter any character string.

If “remove” is entered, the sample ConvertStrBean client program is terminated.

- Deleting the registered information (Execute this operation after the EJB application is terminated.)
  1. Delete the Web service information registered.

        soapmodifydd -undeploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertStrBean

  2. From the Interstage Management Console, select [System] > [WorkUnits] > [SOAPSample] (IJServer WorkUnit name), and [Status] tab.
  3. Click the Stop button of the IJServer WorkUnit "SOAPSample" to stop the IJServer WorkUnit. Then click the Delete button for deletion.

Note
The EJB application registered with SOAPSample is also deleted.

Sample ConvertStrBean Execution Procedure

- Developing the EJP application
  1. On the Interstage Management Console, select [System] > [WorkUnits], and [Create a new WorkUnit].
  2. Specify SOAPSample for "WorkUnit Name" and specify "EJB Applications Only" for "IJServer Type," then click the Create button (creating an IJServer WorkUnit).
  3. Select [System] > [WorkUnits] > [SOAPSample] (IJServer WorkUnit name) and [Deployment] tab.
  4. Specify "/opt/FJSVsoap/examples/ConvertStrBean/ConvertStr.jar " for "Deployment File."
  5. Select [Detailed Settings] and then [EJB Application Settings].
  6. Select the SOAP check box and click the Deploy button.
  7. Select [WorkUnits] > [MyIJServer] (IJServer WorkUnit name) > [Settings] tab, and [WorkUnit Settings]. At [WorkUnit Settings], add the following setting to "Java VM Option" and click the Update button:

        -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient

        LD_LIBRARY_PATH=/opt/FSUNod/lib
        LD_LIBRARY_PATH=/opt/FJSVod/lib

  8. Select [MyIJServer] (IJServer WorkUnit name), [Settings] tab, and [WorkUnit Settings]. Add "/opt/FJSVJ2ee/var/deployment/ijserver/SOAPSample/soapgw/ConvertStr" to "Classpath" in [WorkUnit Settings], and click the Update button.
9. Select [WorkUnits] > [MyIJServer] (IJServer WorkUnit name), and [Status] tab, then click the Start button.

- Registering the CORBA/SOAP server gateway
  1. Move to the following directory
     /opt/FJSVj2ee/var/deployment/ijserver/SOAPSample/soapgw/ConvertStr/ConvertStr
  2. Register the Web service information of the sample ConvertStrBean to Web Service Manager.
     ```
     soapmodifydd -deploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertStrBean_ConvertStrBeanRemoteSOAPGW_dd.xml
     ```

- Executing the sample ConvertStrBean program
  1. Move to the following directory.
     /opt/FJSVsoap/examples/ConvertStrBean/ConvertStrClient
  2. Add the class path.
     ```
     CLASSPATH=$CLASSPATH:/opt/FJSVsoap/examples/ConvertStrBean
     export CLASSPATH
     ```
  3. Activate the sample ConvertNameEJB client program.
     ```
     java ConvertStrClient http://localhost:80/soap_dev/servlet/WSContainer
     ```
     In response to the message “Please input soap method <reverse|toupper|remove> :”, enter the method name to be called.
     If “reverse” or “toupper” is entered, the message “Please input data <String|null>.” is displayed. Enter any character string.
     If “remove” is entered, the sample ConvertStrBean client program is terminated.

- Deleting the registered information (Execute this operation after the EJB application is terminated.)
  1. Delete the Web service information registered.
     ```
     soapmodifydd -undeploy -url http://localhost:12000/soap_admin/servlet/WSContainer -urn ConvertStrBean
     ```
  2. On the Interstage Administration Console, select [System] > [WorkUnits] > [SOAPSample] (IJServer WorkUnit name), and [Status] tab.
  3. Click the Stop button of the IJServer WorkUnit “SOAPSample” to stop the IJServer WorkUnit. Then click the Delete button for deletion.

**Note**
The EJB applications registered in SOAPSample are also deleted.

**Sample ConvertStrBean Execution Example**
```java
java ConvertStrClient http://localhost:80/soap_dev/servlet/WSContainer
Please input soap method <reverse|toupper|remove> :
reverse
```
Please input data <String|null>: fujitsu
reverse = ustijuf
Please input soap method <reverse|toupper|remove> :
toupper
Please input data <String|null>:
toupper = USTIJUF
Please input soap method <reverse|toupper|remove> :
remove
remove completed.
Messaging Method Sample Programs

Sample Program Reliable

File Configuration

The role of each file is shown below.

Table 11-4 File Configuration Descriptions

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushConfig.xml</td>
<td>A setting file for the sender client system of the reliable messaging PUSH model. In this sample, a digital signature option for non-repudiation is not used.</td>
</tr>
<tr>
<td>pullConfig.xml</td>
<td>A setting file for the receiver client of the reliable messaging PUSH model.</td>
</tr>
<tr>
<td>Issoapsecconf.xml</td>
<td>A setting file for the receiver server of the reliable messaging PUSH model and for the sender server of the PULL model.</td>
</tr>
<tr>
<td>TestServer.java</td>
<td>A sample source program containing the receiver application of the PUSH model receiver server and the sender application of the PULL model sender server running on the server system.</td>
</tr>
<tr>
<td>TestPushClient.java</td>
<td>A sample source program containing the sender application running on the sender client of the PUSH model.</td>
</tr>
<tr>
<td>TestPullClient.java</td>
<td>A sample source program containing the receiver application running on the receiver client of the PULL model.</td>
</tr>
</tbody>
</table>
| startpush.bat       | Windows
startpush.sh       | Solaris OE Linux
A script file that activates the sender application of the PUSH model sender client.                                                            |
| startpull.bat       | Windows
startpull.sh         | Solaris OE Linux
A script file that activates the reliable messaging function so that the receiver application of the PULL model receiver client is called.       |
| DD-push.wsdd        | A file containing the Web service information of the PUSH model receiver server.                                                                |
| DD-pull.wsdd        | A file containing the Web service information of the PULL model sender server.                                                                    |
| deployserver.bat    | Windows
deployserver.sh      | Solaris OE Linux
A script file for deploying the above two definition files in the server system.                                                                   |
Model of Sample Program Reliable

This sample provides a client system on which the PULL model receiver application runs. In the server system, the sender application for the PULL model sender server is processed from the receiver application of the PUSH model receiver server. This sample performs indirect reliable messaging from the PUSH model sender client to the PULL model receiver client by this processing.

The type of the message transmitted in the PUSH model is “pushTest” and the type of the message transmitted in the PULL model is “pullTest.”

The ID of the PUSH model sender client is “pushClient1,” the ID of the PULL model receiver client is “pullClient1,” and the receiver server ID of the PUSH model and the sender server ID of the PULL model are “reliableServer1.”

The SOAP message to be transmitted is a simple one having one empty tag in its body.

To simplify execution, this sample assumes that two client functions and one server function run on the same machine by default.

The block diagram of the client and server functions is shown below.

![Figure 11-1 Client and Server Functions](image-url)
Process Overview

When the sample Reliable is executed, the client system having the PUSH model sender application (TestPushClient class) transmits an reliable message “pushTest” to the server system (TestServer class).

The server system (TestServer class) activates the receiver application when receiving the PUSH model message and transmits an reliable message “pullTest” to the client system having the PULL model receiver application (TestPullClient class).

The client system having the PULL model receiver application (TestPullClient class) starts polling the PULL model sender server for messages for when the reliable messaging function is activated. Then an reliable message “pullTest” is transmitted from the PULL model sender server (TestServer class) to the PULL model receiver client (TestPullClient class). The PULL model receiver client (TestPullClient class) activates the receiver application when receiving the PULL model message.

Preparing the Sample Program Reliable

Prepare for executing the program in accordance with the following procedure.

1. Move to the following directory.

   ![Windows] C:\Interstage\F3FMsoap\examples\Reliable
   ![Solaris OE] /opt/FJSVsoap/examples/Reliable
   ![Linux]

2. Compile the sample programs.

   javac TestServer.java
   javac TestPushClient.java
   javac TestPullClient.java

   If compilation fails, the Interstage SOAP service may not be installed properly. For details, refer to the Installation Guide.

3. Copy the setting files of the PUSH model receiver server and the PULL model sender server that have the reliable messaging function. If the setting file for the high reliability Web service function has already been set, make a backup copy of issoapseccconf.xml.

   ![Windows] copy issoapseccconf.xml C:\Interstage\F3FMsoap\conf\
   ![Solaris OE] cp issoapseccconf.xml /opt/FJSVsoap/conf/
   ![Linux]
Chapter 11: Sample Program Configuration

4. Select [Settings] for the IJServer WorkUnit “MyIJServer,” select [WorkUnit Settings], add the directory in step1 to “Classpath,” and restart WorkUnit “MyIJServer.”

5. Execute the following script to register the PUSH model receiver server and the PULL model sender server.

```plaintext
Windows
C:\Interstage\F3FMsoap\examples\Reliable\deployserver.bat

Solaris OE Linux
/opt/FJSVsoap/examples/Reliable/deployserver.sh
```

With the above procedure, the preparation for executing the sample program is completed.

**Executing the Sample Program Reliable**

1. Transmitting a PUSH model reliable message
   
   Execute the following script on the machine running the Interstage SOAP server.

   ```plaintext
   Windows
   C:\Interstage\F3FMsoap\examples\Reliable\startpush.bat
   
   Solaris OE Linux
   /opt/FJSVsoap/examples/Reliable/startpush.sh
   ```

   When the above script is executed, the PUSH model sender client “pushClient1” is activated and the sender application is executed. The sender application transmits a message of the message type “pushTest” for the PUSH model receiver server “reliableServer1.”

   The PUSH model receiver server “reliableServer1” receives the message from the PUSH model sender client “pushClient1” and executes the receiver application of the “pushTest” message. The receiver application executes the sender application and transmits a message of the message type “pullTest” for the PULL model receiver client “pullClient1.”

2. Receiving a PULL model reliable message

   Next, execute the following script on the machine running the Interstage SOAP server.

   ```plaintext
   Windows
   C:\Interstage\F3FMsoap\examples\Reliable\startpull.bat
   ```
When the above script is executed, the PULL model receiver client “pullClient1” starts polling the PULL model receiver server “reliableServer1” for messages. Then a reliable message created by the sender application of the PULL model receiver server “reliableServer1” is transmitted to the receiver client “pullClient1.” On the PULL model receiver client, the receiver application of the “pullTest” message is executed.

**Changing the Setting of the Sample Program Reliable**

To simplify execution, this sample assumes that both client and server systems run on the same machine. If you want to run the client system and the server system on different machines, the setting file of each client needs to be changed. In this case, on the system that is running the PUSH model sender client, change the setting by executing the soapsecrelconf command as shown below.

```bash
soapsecrelconf -add to -f pushConfig.xml -messageid pushTest -serviceid ReliableSamplePushService
    -partnerid reliableServer1 -address http://servername/soap_dev/services/ReliableSamplePushService
```

On the system that is running the PULL model receiver client, change the setting by executing the soapsecrelconf command as shown below.

```bash
soapsecrelconf -add from -f pullConfig.xml -messageid pullTest -serviceid ReliableSamplePullService
    -partnerid reliableServer1 -listener TestPullClient -address http://servername/soap_dev/services/ReliableSamplePullService
```

For either client, enter the connection destination server system name for “servername.”

**Deleting the Message File of the Sample Program Reliable**

After this sample is executed, the reliable messages are stored in files in the following directories.

**Windows**

C:\Interstage\F3FMsoap\etc\reliableMessage\store_pushTest_with_pushClient1
C:\Interstage\F3FMsoap\etc\reliableMessage\store_pushTest_with_reliableServer1
C:\Interstage\F3FMsoap\etc\reliableMessage\store_pullTest_with_pullClient1
C:\Interstage\F3FMsoap\etc\reliableMessage\store_pullTest_with_reliableServer1

**Solaris OE**

/opt/FJSVsoap/etc/reliableMessage/store_pushTest_with_pushClient1
/opt/FJSVsoap/etc/reliableMessage/store_pushTest_with_reliableServer1
/opt/FJSVsoap/etc/reliableMessage/store_pullTest_with_pullClient1
/opt/FJSVsoap/etc/reliableMessage/store_pullTest_with_reliableServer1
Notes on Use

This section describes the notes about the use of the sample programs.

- The display of the [Start] menu of Windows may slightly differ from system to system.

  **Windows**
  - Execute the `soapmodifydd` command as the user who belongs to the Administrators local group.

  **Solaris OE/Linux**
  - Execute the `soapmodifydd` command as the super user.
The SOAP version, HTTP version, and client function log output destination of the Web service container can be customized in the [WorkUnit setting] dialog box.

To open the [WorkUnit setting] dialog, select [System] > [WorkUnit], and the [New] tab, or select the [Environment setup] tab on an existing IJServer WorkUnit.

The following property values in "JavaVM option" at [WorkUnit setting] can be set:

- SOAP version
- HTTP version
- Client function log
- Return value of void

**Note**

If an invalid value or character (one that cannot be specified or that is out of the allowable range) is specified for a property, the default value is used.
SOAP Version Setting

Set the SOAP version using the following methods.

**Table 12-1 SOAP Version Settings**

<table>
<thead>
<tr>
<th>Functions used</th>
<th>Method of specifying the SOAP version (In descending order of priority)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server function</td>
<td>1. SOAP version received from the client application</td>
<td></td>
</tr>
<tr>
<td>Client function (excluding the reliable messaging function)</td>
<td>1. Setting in the source program&lt;br&gt;2. Setting from the Interstage Management Console (&quot;JavaVM option&quot; at [WorkUnit Settings])</td>
<td>If no setting is made, SOAP1.1 is assumed.</td>
</tr>
<tr>
<td>The reliable messaging function</td>
<td>1. Setting from the Interstage Management Console (&quot;JavaVM option&quot; at [WorkUnit Settings])</td>
<td></td>
</tr>
<tr>
<td>Server and client functions</td>
<td>1. Setting in the source program&lt;br&gt;2. SOAP version received from the client application</td>
<td>When using the client function to transmit a SOAP message received with the server function.</td>
</tr>
</tbody>
</table>

**Setting from Interstage Management Console in "JavaVM option" at [WorkUnit setting]**

Set the SOAP version as shown below.

- Property name
  
  com.fujitsu.interstage.soapx.soap.version

- Values that can be set
  
  1.1 or 1.2
  
  The default is 1.1

**Description example**: The Soap version is set to 1.1.

  com.fujitsu.interstage.soapx.soap.version = 1.1
Setting from the Source Program

RPC Method Web Service

When the SOAP version is set in the source program, the SOAP version is set as a property for the Call object.

```java
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Service;
import javax.xml.rpc.Call;

......
Service service = ServiceFactory.newInstance()
  .createService(proxySampleServiceQName);
Call call = service.createCall();

//when SOAP1.1 is set
Call.call.setProperty("com.fujitsu.interstage.soapx.soap.version", "1.1");
......
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value (java.lang.String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.fujitsu.interstage.soapx.soap.version</td>
<td>“1.1” or “1.2”</td>
</tr>
</tbody>
</table>

Web Service in the Messaging Method (excluding the reliable messaging function)

By casting the javax.xml.soap.MessageFactory object to the com.fujitsu.interstage.soapx.soap.MessageFactoryImpl class, the following methods can be used. These methods create SOAP message objects of the SOAP version 1.1 or 1.2.

Table 12-2 Package name: com.fujitsu.interstage.soapx.soap

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>javax.xml.soap.SOAPMessage createMessage12Impl()</td>
<td>Creates a SOAP1.2 message.</td>
</tr>
</tbody>
</table>

The SOAP version of the created SOAP message object is not changed until the object is abandoned. However, in the following case the SOAP version of the SOAP message object (SOAP envelope) is updated.

- When the data of the SOAP envelope of a version is different from that of the SOAP message object, for the void setContent(javax.xml.transform.Source envSrc) method of the javax.xml.soap.SOAPPart object.
Note

Use caution on the use of SOAP1.2 in the following conditions.

- When the Fault information is set in the server application of the Web service, only the Fault code is transmitted.

- The RPC client application and the CORBA/SOAP client gateway cannot obtain Fault information. JAXRPCException has a message indicating that SOAPFault of SOAP1.2 cannot be interpreted is generated.

- When SOAPFault is received from the RPC server application, the CORBA/SOAP client gateway throws COMM_FAILURE of the CORBA system exception to the CORBA client application.

- The Messaging method application cannot obtain Fault information. SOAPException having a message indicating that SOAPFault of SOAP1.2 cannot be interpreted is generated.

- When the reliable messaging function is used, a message indicating that SOAPFault of SOAP1.2 cannot be interpreted is output to the log. Though there is no problem in operation, it is recommended that SOAP1.1 be used in the reliable messaging function.
Setting the HTTP Version

Specify the version of the HTTP that is transmitted from the client system. The server system is dependent on the environment of the Web server used.

- Property name
  com.fujitsu.interstage.soapx.http.version

- Available value
  1.0 or 1.1
  The default is 1.1

**Description example**: The HTTP version is set to 1.1.

```
com.fujitsu.interstage.soapx.http.version = 1.1
```
Setting the Logs of the Client Function

When the client function of the Interstage SOAP service is used, logs are not output. This section describes the method for changing the log output destination when the client function is used.

**Setting the Log Output Destination**

- **Property name**
  
  com.fujitsu.interstage.soapx.log.system.out

- **Available value**

  If no value is set, “nothing” is assumed.
  - nothing (default) : No log is output.
  - stdout : Logs are output to the standard output.
  - logfile : A log file is created under the directory specified with the property name “com.fujitsu.interstage.soapx.log.system.path.”

**Description example**: Setting of outputting logs to the standard output.

com.fujitsu.interstage.soapx.log.system.out = stdout
Outputting Logs to a File

Setting the Name of the Directory in which a Log File is Created.

- **Property name**
  
  `com.fujitsu.interstage.soapx.log.system.path`

- **Available value**
  - No setting: The default path name is assumed.
  - Directory name in which a log file is created (full path specification)

  **Windows**
  
  The default path name is “%IS_HOME%\F3FMsoap\log”.

  **Solaris 0E**  **Linux**
  
  The default path name is “/opt/FJSVsoap/log”.

**Description example:** The directory name in which a log file is created is set to “C:\Interstage\F3FMsoap\log”.

  - `com.fujitsu.interstage.soapx.log.system.path = C:\Interstage\F3FMsoap\log`

  **Solaris 0E**  **Linux**
  
  **Description example:** The directory name in which a log file is created is set to “/opt/FJSVsoap/log”.

  - `com.fujitsu.interstage.soapx.log.system.path = /opt/FJSVsoap/log`

**Note**

- The name of the log file created is "message.log". The file name cannot be changed.
- If a removable disk drive is specified for the path name and the specified disk drive does not exist, a dialog box may be displayed depending on the system.
Specifying the Maximum Size of the File to which Logs are Output.

- Property name
  com.fujitsu.interstage.soapx.log.system.maxsize

- Available value
  0 to 2GB (The default is 10MB.)

  Set the maximum size of the file with only a numeric value (in bytes) or a numeric value and a unit ("KB", "MB", or "GB"). The maximum size is 2GB.

  **Description example:** The maximum size of the log file is set to 2 MB.
  com.fujitsu.interstage.soapx.log.system.maxsize = 2MB

Setting the Maximum Number of Generations of the File to which Logs are Output.

- Property name
  com.fujitsu.interstage.soapx.log.system.backup

- Available value
  A value between 0 and 100.
  (The default is 5.)

  **Description example:** The maximum number of generations of the log file is set to 3.
  com.fujitsu.interstage.soapx.log.system.backup = 3
Setting the Return Value of “void”

Sometimes, when interconnecting with products of another vendor, mutual connection is difficult due to the difference in the handling (implementation) of the return value of “void”.

Usually, avoiding the use of “void” as a return value of the RPC server application can improve interconnection performance. If it is difficult to modify the RPC server application, change the method of handling the return value of “void”.

Setting the Method of Handling the Return Value of “void”

- Property name
  com.fujitsu.interstage.soapx.simpleVoid
- Available value
  - true (case insensitive): If the return value is “void,” the return value of SOAP data is not transmitted.
  - Other (default): Even if the return value is “void,” dummy information is transmitted as the return value of SOAP data.

Description example: Setting is made so that the return value is “void” is not sent if the return value is “void.”

com.fujitsu.interstage.soapx.simpleVoid = true
Increasing the Speed of Calling the RPC Application

This section describes the method of increasing the speed of the remote procedure calling processing of the Web service.

**Table 12-3 Package Name: com.fujitsu.interstage.soapx.client**

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>public void setStubUse(boolean flag)</td>
<td>The speed of calling the RPC application is increased.</td>
</tr>
<tr>
<td></td>
<td>implements javax.xml.rpc.Call</td>
<td></td>
</tr>
</tbody>
</table>

- **DII method RPC client application**
  
  By casting the javax.xml.rpc.Call object to the com.fujitsu.interstage.soapx.client.Call class, the above methods can be used.
  
  Describe the method before the execution of remote procedure calling (invoke method).

- **Stub method RPC client application.**
  
  For the stub method RPC client application, describe the method before the execution of remote procedure calling (invoke method) in the stub source created by IDE.
  
  For further details, refer to the IDE documents used such as Apworks.

**Note**

Before making a setting to increase the speed of calling the RPC application, make sure that the service name and parameters (parameter name, data type, etc.) of the Web service are correct. If a parameter is incorrect, SOAPFault or JAXRPCException will be generated.

**Setting Method**

If “true” is specified for parameter “flag,” the SOAP message is optimized (the speed of the remote procedure calling of the Web service is increased). In this case, the parameter of the RPC application and the data type excluding the Vector type included in the return value is single reference (an instance is generated for each variable).

If “false” is set to parameter “flag,” optimization of SOAP messages is not performed.
Selecting a Certificate Used for Client Authentication

Secure Sockets Layer (SSL) client authentication certificates can be selected by the following method. This method casts the javax.xml.rpc.Call object to the com.fujitsu.interstage.soapx.client.Call class.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>public void setSSLAuthKeyAlias( String authKeyAlias )</td>
<td>Selects the site certificate used for SSL client authentication</td>
</tr>
</tbody>
</table>

Method of Selection

For parameter authKeyAlias, specify the alias of the site certificate used for SSL client authentication. Call this method before the invoke method of the same class.

Example

```java
import javax.xml.rpc.*
import javax.xml.namespace.QName;

Service service = ServiceFactory.newInstance().createService(
    new QName( "urn:sample-countRequest", "count" ));
javax.xml.rpc.Call call = service.createCall();
call.setTargetEndpointAddress(targetURL);
call.setOperationName( new QName( targetURN, targetOPName ));
call.setReturnType(datatype);

// Specification of certificate: The certificate registered with alias //"mysecondcert" is specified.

((com.fujitsu.interstage.soapx.client.Call)call).setSSLAuthKeyAlias("mysecondcert");
call.invoke( new Object[] {} );
```
Editing the Web Application File

The Web service container can be customized by editing the following items in the Web application file (web.xml) in the WAR file (soap.war or soap_dev.war) and deploying the file:

- Session timeout time
- Customization of Web service information management file
  - Update authority of Web service information management
  - Location of Web service information management file
  - Web service information read mode

Changing the Session Time Out

The session timeout time (minutes) applicable when server system applications use session management can be changed.

Specify the session parameter in the session-config tag, the session timeout time is in the "session-timeout" tag. The default is 30 minutes.

Example

```xml
<web-app>
  :<session-config>
    <session-timeout>
      10
    </session-timeout>
  </session-config>
  :</web-app>
```
Customizing the Method of Managing the Web Service Information Management File

By changing the following items, the method of accessing the Web service information management file by the Web service manager can be customized.

**Access Authority of Web Service Information Management File (web_app_kid)**

- **admin**
  According to an update request (-deploy, -undeploy, or replace option) from the soapmodifydd command, the Web service manager can update the Web service information registered.

- **user**
  The Web service manager cannot update the Web service information registered. It can read the Web service information registered, according to a reference request (-list or -query option) from the soapmodifydd command.

**Changing Web Service Information Read Mode (operation_mode)**

Specify the read mode of the Web service information of the Web service container. The default is "develop".

- **develop**
  Each time a request is received from the client application, the Web service manager checks whether the Web service information has been updated (registered, modified, or deleted) and, if it has been updated, reads the updated Web service information.

- **employ**
  When the Web service container starts, all Web service information is read from the Web service information management file only one time.
Appendix A

Describing Web Service Information

Use the following conventions to describe the Web service information within XML as shown below.

- Enclose the definition with a start tag (<tag-name>) and an end tag (</tag-name> or </>):
  when defining other tags or parameters in a tag:

  ```xml
  <tag-name>
    ...
  </tag-name>
  
  when not defining other tags or parameters in a tag:

  ```xml
  <tag-name ... />
  ```

- Describe the tag attribute next to the start tag:
  - when defining other tags or parameters for the tag:

  ```xml
  <tag-name attribute-name-1= "attribute value 1" attribute-name-2= "attribute value 2" ...> ...
  </tag-name>
  ```

  - when not defining other tags or parameters in a tag.

  ```xml
  <tag-name attribute-name-1= "attribute value 1" attribute-name-2= "attribute value 2" .../>
  ```

- Write a comment between "<![CDATA[" and "-->":

  ```xml
  <!--  This is a comment. -->
  ```

Notes

Notes on the description of tags are shown below.

- If “x” is written in the Tag omission column of a tag in the types and description of tags, the tag and its attributes cannot be omitted.
  
  If the tag is omitted, its definition becomes invalid and an error is output at the time of registration.
• If “x” is written in the Specifying two or more tags column of a tag in the types and description of
tags, the tag can be specified only once. If the tag or its attribute is specified more than once, an
error is output when registering the Web service information

• Case sensitive.

• All tags must belong to the name space “urn:xmlns-fujitsu-com:soapx-wsdd-2002-10”. If a tag that
belongs to another name space or that does not belong to any name space is specified, it is not
recognized as a tag of the Interstage SOAP service and ignored.

• The following characters cannot be written as value:
  • ‘&’, ‘’, ‘”’, ‘<’, ‘>’

• The following characters can be used for a path name:
  Alphanumeric characters, ‘+’, ‘-’, ‘.’, ‘\’, ‘\’, ‘\’, ‘\’, ‘\’, ‘\’, ‘\’

• The maximum length of a path name is as follows:

  Windows
  – 255bytes

  Solaris 0E, Linux
  – 1023bytes
XML Tags Describing Web Service Information

The hierarchy of XML tags that describe Web service information is shown below.

XML tag hierarchy of Web service information

Figure A-1 XML Tag Hierarchy of Web Service Server Application Information

Figure A-2 XML Tag Hierarchy of Web Service Client Application Information

Figure A-3 Undeployment Tag Information
## Types of XML tags of Web service information

### Table A-1  Types of XML tags of Web service information

<table>
<thead>
<tr>
<th>Tag name</th>
<th>Description</th>
<th>Tag omission</th>
<th>Specifying two or more tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment</td>
<td>Defines the start and end of the registration of Web service information.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>undeployment</td>
<td>Defines the start and end of the deletion of Web service information.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>service</td>
<td>Defines Web service.</td>
<td>X</td>
<td>0 (*1)</td>
</tr>
<tr>
<td>parameter</td>
<td>Defines the parameters of the Java method of the Web service.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>operation</td>
<td>Defines the Java method.</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>typeMapping</td>
<td>Defines the mapping information of the parameter/return value.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>requestFlow</td>
<td>Indicates the handler that is executed before provider processing.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>responseFlow</td>
<td>Indicates the handler that is executed after provider processing.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>handler</td>
<td>Indicates that it is the definition of the handler.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beanMapping</td>
<td>Defines the mapping information of the parameter/return value. (dedicated to javaBean).</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*1: Only when the soapmodifייdd “-set” option is specified.
Contents of XML Tags in Web Service Information

This section describes the contents of the XML tags that are described in Web service information files.

**deployment tag**

**Table A-2 Deployment Tag Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns</td>
<td>An attribute that specifies the name space used in the “deployment” service. It specifies the following three name spaces. Do not change this name space. xmlns=&quot;urn:xmlns-fujitsu-com:soapx-wsdd-2002-10&quot; xmlns:java=&quot;urn:xmlns-fujitsu-com:soapx-wsdd-providers-java-2002-10&quot; xmlns:handler=&quot;urn:xmlns-fujitsu-com:soapx-wsdd-providers-handler-2002-10&quot;(*1)</td>
<td>None</td>
<td>X</td>
</tr>
</tbody>
</table>

*1: “handler” is mandatory only when the high-reliability Web service function is operated.

**undeployment tag**

**Table A-3 Undeployment Tag Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns</td>
<td>Specifies the name space used in the “undeployment” service. Do not change this name space. xmlns=&quot;urn:xmlns-fujitsu-com:soapx-wsdd-2002-10&quot;</td>
<td>None</td>
<td>X</td>
</tr>
</tbody>
</table>
service tag

Table A-4 Service Tag Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the Web service identifier.</td>
<td>None</td>
<td>0(*1)</td>
</tr>
<tr>
<td>provider</td>
<td>Specifies the provider.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java:MSG</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies that it is a Messaging method Web service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java:RPC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies that it is an RPC (Remote Procedure Calling) method Web service (including the CORBA/SOAP gateway).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- In addition, the following provider name can be specified.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the reliable messaging function is used</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java:ISSOAPFactoryProvider</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When linking with the Factory method CORBA application</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java:ISSOAPStatefulEJBProvider</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When linking with EJB (STATEFUL SessionBean)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java:ISSOAPStatelessEJBProvider</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When linking with EJB (STATELESS SessionBean)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1: It cannot be specified when "-deploy" and "-replace" are specified in the soapmodiyfydd command.

parameter tag (when it is a child element of the “service” tag)

The parameter tag has the attribute names “name,” “value,” or “key.”

Define the values of the “name” and “value” attributes of the “parameter” tag shown in the following table in the following format.

Example

When the value of the “name” attribute is “corbaName” and the value of the “value” attribute is “NSbind”:
### Table A-5 Parameter Tag Attributes

<table>
<thead>
<tr>
<th>Value of the “name” attribute</th>
<th>Value of the “value” attribute</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>corbaName</td>
<td>CORBA naming service registration name</td>
<td>Specify the CORBA naming service registration name during linking with CORBA.</td>
<td>None(*1)</td>
<td>0</td>
</tr>
</tbody>
</table>
| scope                         | request session application | Specify the life cycle of the instance of the server application. The following values can be specified.  
   • - request  
     It generates and abandons the instance of the server application each time a request is made.  
   • - session  
     Specify this value when using the session management. The instance of the server application is retained until is performed explicitly by the server application the request acceptance termination processing or while there are requests from the client application before the session management timeout.  
   • - application  
     It generates the instance of the server application when the first request is made from the client application. This instance is retained until the Interstage SOAP service is stopped.  
     **Note:** If the value of the name="static" attribute of the parameter tag is "true," these settings become invalid. | request | 0        |
| className                     | Class name                    | Specify the Java class name of the server application when the “provider” attribute is other than Handler (Web service of reliable messaging). | None | X (*3)    |
| handlerClass                  | Class name                    | Specify the Java class name of Handler when the “provider” attribute is Handler (Web service of reliable messaging). | None | X (*4)    |
### Appendix A: Describing Web Service Information

<table>
<thead>
<tr>
<th>Value of the “name” attribute</th>
<th>Value of the “value” attribute</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>true / false</td>
<td>Specify whether the instance of the server application is to be generated. The following values can be specified.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- true</td>
<td>The server application is not instantiated. (Specify it when registering only the class method.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- false</td>
<td>The server application is instantiated.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong>: If “true” is specified, the name=&quot;scope&quot; attribute of the “parameter” tag becomes invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>allowedMethods</td>
<td>Method name</td>
<td>The use of the specified method is permitted. Use of all methods is permitted by specifying an asterisk “*” or not specifying the allowedMethods attribute. If specifying two or more methods, delimit each method by a comma “,”.</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>option</td>
<td>Key</td>
<td>Define the option information used by the provider. In “option” setting, the “key” attribute is also used. The XML statement is as shown below.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;parameter name=&quot;option&quot; key=&quot;key value&quot; value=&quot;value value&quot;/&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For the “key” attribute, specify the information name to be added as option information. For the “value” attribute, specify the information value to be added as option information.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>deleteProcessedHeader( &quot;2)</td>
<td>true / false</td>
<td>Specify whether the processed HeaderElement of the SOP message is to be deleted. If “true” is set, it is deleted; if “false” is set, it is not deleted.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>pushId</td>
<td>messageID</td>
<td>Set the message type ID of the push method.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>pullId</td>
<td>messageID</td>
<td>Set the message type ID of the pull method.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>role</td>
<td>actor name</td>
<td>Define the actor name of the Web service (SOAP header destination). If it is not specified, SOAP messages are sent to the Web service as they are.</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>
*1: To link with the CORBA application, specify the CORBA naming service registration name. Specify the Enterprise Bean name for linking with EJB.

*2: The deleteProcessedHeader tag is information required for operating the high reliability Web service function.

The attribute information of the deleteProcessedHeader tag is managed by the Web service. Do not change it.

*3: It cannot be omitted if the “provider” attribute is other than “Handler.”

*4: It cannot be omitted if the “provider” attribute is “Handler.”

**operation tag**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify the Java method name of the Web service.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>returnType</td>
<td>Specify the type of the return value of the Java method.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>returnQName</td>
<td>Specify the tag name of the return value of the Java method.</td>
<td>return</td>
<td>0</td>
</tr>
<tr>
<td>responseSuffix</td>
<td>Specify the tag name of the value returned by the Java method.</td>
<td>Response</td>
<td>0</td>
</tr>
<tr>
<td>xmlns</td>
<td>Specify the name space used in the “operation” tag.</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>
parameter tag (when it is a child element of the “operation” tag)

Table A-7 Parameter Tag Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specify the parameter name of the Java method of the Web service.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>type</td>
<td>Specify the parameter type specified in the Java method of the Web service in XML data type.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>For the structure and enumeration type equivalent to the User-defined type, description of the typeMapping tag is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For the Bean type equivalent to the User-defined type, description of the beanMapping tag is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When specifying an array type (Array), description of the arrayType attribute is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For the usable XML data type, see Chapter 8 Support of Data Types.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arrayType</td>
<td>If the array type (Array) is specified in “type,” specify the Java data type that is to be set to an array type.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>If the argument type is an array, put as many brackets “[” as the number of nets of the array at the head and add the full specification name. If it is of non-primitive type, insert “L” between the last bracket “[” and the full specification name and put a semicolon “;” after the full specification name.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>int-type one-dimensional array: [I</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data-type two-dimensional array: [[Ljava.lang.String;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>Specify the parameter attribute (in/out/inout). The following values can be specified.</td>
<td>IN</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>- IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- INOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Description Example of Web Service Information.
### typeMapping tag

#### Table A-8  TypeMapping Tag Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>encodingStyle</td>
<td>Specify the encoding method of data (*1). <a href="http://schemas.xmlsoap.org/soap/encoding/(default)">http://schemas.xmlsoap.org/soap/encoding/(default)</a></td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>xmlns</td>
<td>Specify the name space of the XML data type.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>qname</td>
<td>Specify the name of the XML data type corresponding to the Java data type.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>languageSpecificType</td>
<td>Specify the class name indicating the Java data type. For the class name, specify in full specification name including the package name.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>serializer</td>
<td>Specify the Java class name of serializer that converts the Java data type into XML data type. The following class names can be specified. com.fujitsu.interstage.soapx.encoding.ser.EnumSerializerFactory (for enumeration) com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory (for structure)</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>serializer</td>
<td>Specify the Java class name of deserializer that converts the XML data type into the Java data type. The following class names can be specified. com.fujitsu.interstage.soapx.encoding.ser.EnumSerializerFactory (for enumeration) com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory (for structure)</td>
<td>None</td>
<td>X</td>
</tr>
</tbody>
</table>

(*1) Regardless of whether the encoding method defined in SOAP1.1 or that defined in SOAP1.2 is specified, the system assumes that the encoding formats of both SOAP1.1 and SOAP1.2 are specified.

For the description example of the typeMapping tag, see the example of the structure in
Description Example of Web Service Information.

**requestFlow tag**

**Table A-9 RequestFlow Tag Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>There is no attribute.</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>
responseFlow tag

Table A-10  ResponseFlow Tag Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>There is no attribute.</td>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>

handler tag

The Header tag contains information required for operating the high-reliability Web service function. The attribute information of the "handler" tag is managed by the Web service. Do not change it. The attributes that can be referenced are listed below.

Table A-11  Handler Tag Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>It shows the handler name used by the high-reliability Web service function.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>- username</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>User name added handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The user name/password is added to a SOAP message.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- sign</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Signature added handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The SOAP digital signature is added to a SOAP message.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- encrypt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encryption handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A SOAP message is encrypted by the XML cipher.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- security</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security processing handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>It obtains the user name/password added to a SOAP message, verifies the SOAP digital signature, and decrypts XML cipher.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- permission</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reliable messaging permission handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>It authorizes the reliable messaging function.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute name</td>
<td>Description</td>
<td>Default</td>
<td>Omission</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>type</td>
<td>Specify the handler class name.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>- User name added</td>
<td>java:com.fujitsu.interstage.soapx.wssecurity.handler.UsernameTokenSender</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Signature added handler</td>
<td>java:com.fujitsu.interstage.soapx.wssecurity.handler.SignHandler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Encryption handler</td>
<td>java:com.fujitsu.interstage.soapx.wssecurity.handler.EncryptHandler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Reliable messaging permission handler</td>
<td>java:com.fujitsu.interstage.soapx.reliable.connection.axis.MessagePermissionHandler</td>
<td></td>
</tr>
</tbody>
</table>

**parameter tag**

parameter tag (when it is a child element of the “handler” tag)

The “parameter” tag contains information required for operating the high-reliability Web service function. The attribute information of the “parameter” tag is managed by the Web service. Do not change it. The attributes that can be referenced are listed in the table below.

**Table A-12 Parameter Tag Attributes For Server System Environment**

<table>
<thead>
<tr>
<th>Handler</th>
<th>Value of the “name” attribute</th>
<th>Value of the “value” attribute</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP signature generation</td>
<td>actor</td>
<td>actorURL</td>
<td>URI of the intermediary that verifies the SOAP digital signature.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mustUnderstand</td>
<td>true / false</td>
<td>Process type</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>XML encryption</td>
<td>actor</td>
<td>actorURL</td>
<td>URI of the intermediary that decrypts the data encrypted by XML encryption</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mustUnderstand</td>
<td>true / false</td>
<td>Process type</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Handler</td>
<td>Value of the &quot;name&quot; attribute</td>
<td>Value of the &quot;value&quot; attribute</td>
<td>Description</td>
<td>Default</td>
<td>Omission</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Security processing</td>
<td>verify</td>
<td>true / false</td>
<td>Set &quot;true&quot; when verifying the SOAP digital signature.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>decrypt</td>
<td>true / false</td>
<td>Set &quot;true&quot; when verifying the SOAP digital signature.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>Reliable messaging authorization</td>
<td>messageType</td>
<td>partyId1,partyId2,....</td>
<td>Set the message type ID to “name” and set the sender ID (PULL)/the receiver ID (PUSH) separated by a comma to “value.”</td>
<td>None</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table A-13 Parameter Tag Attributes For Client System Environment**

<table>
<thead>
<tr>
<th>Handler</th>
<th>Value of the &quot;name&quot; attribute</th>
<th>Value of the &quot;value&quot; attribute</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name addition</td>
<td>actor</td>
<td>actorURL</td>
<td>URI of the intermediary that processes the user name/password added to the SOAP message.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mustUnderstandd</td>
<td>true / false</td>
<td>Flag that indicates whether the receiver is always required to verify the SOAP digital signature.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>SOAP signature generation</td>
<td>actor</td>
<td>actorURL</td>
<td>URI of the intermediary that decrypts the data encrypted by XML encryption.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mustUnderstandd</td>
<td>true / false</td>
<td>Flag that indicates whether the receiver is always required to decrypt data encrypted by the XML encryption.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>XML encryption</td>
<td>actor</td>
<td>actorURL</td>
<td>URI of the intermediary that decrypts the data encrypted by XML encryption.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Handler</td>
<td>Value of the “name” attribute</td>
<td>Value of the “value” attribute</td>
<td>Description</td>
<td>Default</td>
<td>Omission</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>mustUnderstand</td>
<td>true / false</td>
<td></td>
<td>Flag that indicates whether the receiver is always required to decrypt data encrypted by the XML encryption.</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Security processing</td>
<td>verify</td>
<td>true / false</td>
<td>Set “true” when verifying the SOAP digital signature.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>decrypt</td>
<td>true / false</td>
<td>Set “true” when decrypting data encrypted by XML encryption.</td>
<td>false</td>
<td>0</td>
</tr>
</tbody>
</table>

**BeanMapping**

**Table A-14  BeanMapping Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Default</th>
<th>Omission</th>
</tr>
</thead>
<tbody>
<tr>
<td>qname</td>
<td>Specify the name of XML data type that maps Java data type.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>xmlns</td>
<td>Specify the name space of the XML data type name.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td>languageSpecificType</td>
<td>Specify the class name of the Java data type.</td>
<td>None</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>For the type name specified for the data type, specify in full specification, the name (including the package name).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Describing Web Service Information

Description Example of Web Service Information

**Description example of basic Web service information**

A description example of the Web service information of the sample program Calc is shown below.

- Method name: `add`
- Web service identifier: `urn:sample-calc`
- Parameter name: `arg1` (int-type “in” parameter), `arg2` (int-type “in” parameter)

```xml
<deployment xmlns="urn:xmlns-fujitsu-com:soapx-wsdd-2002-10"
            xmlns:java="urn:xmlns-fujitsu-com:soapx-wsdd-providers-java-2002-10">
    <service provider="java:RPC">
        <parameter name="scope" value="application"/>
        <parameter name="className" value="Calc.server.CalcServer"/>
        <operation name="add">
            <parameter name="arg1" type="n1:int" mode="in"
                        xmlns:n1="http://www.w3.org/2001/XMLSchema"/>
            <parameter name="arg2" type="n1:int" mode="in"
                        xmlns:n1="http://www.w3.org/2001/XMLSchema"/>
        </operation>
    </service>
</deployment>
```

**Description example of RPC method Web service information using the User-defined type**

A description example of the Web service information for the RPC server application that has the structure type as its parameter is described below.

- Method name: `op01`
- Web service identifier: `urn:sample`
- Parameter name: `in_p` (structure structF-type “in” parameter), `inout_p` (structure-type structF-type “inout” parameter), `out_p` (structure-type structF-type “out” parameter)

```xml
<deployment xmlns="urn:xmlns-fujitsu-com:soapx-wsdd-2002-10"
            xmlns:java="urn:xmlns-fujitsu-com:soapx-wsdd-providers-java-2002-10">
    <service name="sample" provider="java:RPC">
        <parameter name="scope" value="request"/>
        <parameter name="className" value="_INTF_DTstr2SOAP"/>
        <parameter name="allowedMethods" value="*"/>
        <operation name="op01" returnQName="return" returnType="RTypeNS:structF"
                   xmlns:RTypeNS="urn:Fujitsu-Soap-Service-Data"
                   responseSuffix="Response">
            <parameter name="in_p" type="tns:structF" mode="IN"
                        xmlns:tns="urn:Fujitsu-Soap-Service-Data"/>
            <parameter name="inout_p" type="tns:structF" mode="INOUT"
                        xmlns:tns="urn:Fujitsu-Soap-Service-Data"/>
            <parameter name="out_p" type="tns:structF"/>
        </operation>
    </service>
</deployment>
```
mode="OUT" xmlns:tns="urn:Fujitsu-Soap-Service-Data"/>
  </operation>
  <typeMapping xmlns:x="urn:Fujitsu-Soap-Service-Data" qname="x:structF" languageSpecificType="INTF_DTstr2Package.structFSOAP"
    serializer="com.fujitsu.interstage.soapx.encoding.ser.BeanSerializerFactory"
    deserializer="com.fujitsu.interstage.soapx.encoding.ser.BeanDeserializerFactory"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</service>
</deployment>
Index

Access limit of Reliable messaging function, 7-19
Adding HTTP connection information including proxy and authentication information (DII method), 5-25
Adding HTTP connection information including proxy and authentication information (stub method), 5-11
Agreement between Client System and Server System, 7-2
Agreement necessary in PULL model, 7-12
Agreement necessary in PUSH model, 7-5
Allocation of receiver application, 7-7, 7-15
Allocation of sender application, 7-13
allowedMethods, A-8
Analyzing fault information, 5-33
Any Type, 8-38
Array type, 8-20, 8-41
Array/sequence type, 8-36
Association of conversion mapping for conversion from Java values to XML values and vice versa (CORBA), 6-5
Association of conversion mapping for conversion from Java values to XML values and vice versa (EJB), 6-21
Basic Data Types, 8-31
Basic RPC client application (DII method), 5-17
Basic RPC client application (stub method), 5-7
Bean type, 8-17
Before Using Sample Programs, 11-4
Class and exception types, 8-40
className, A-7
CORBA Object/interface type, 8-37
CORBA/SOAP client gateway, 10-6
CORBA/SOAP server gateway, 10-3
corbaName, A-7
Correspondence of Conversion Mapping Required for Conversion between Java and XML Values, 8-28
Defining and Analyzing Fault Information, 5-31
Defining fault information, 5-31
deleteProcessedHeader, A-8
Deletion of messages, 7-22
Deployment in the Client System Environment, 9-18
Destination of reliable messages, 7-20
Edit, 9-2
Enumeration type, 8-8, 8-32
Environment Setting for using SSL with InfoProvider Pro, 8-2
Execution of sender application, 7-10
Generation of Java Class Files for CORBA/SOAP Gateway, 6-4
Generation of receiver application on PUSH model Receiver server, 7-6
Generation of sender application on PULL model Receiver client, 7-14
Generation of sender application on PULL model sender server, 7-13
Generation of sender application on PUSH model sender client, 7-7
handler tag, A-12
handlerClass, A-7
Handling attachment files, 5-3
Increasing the Speed of Calling the RPC Application, 12-10
java.util.Vector type, 8-24
Linkage with CORBA server application that uses any type, 6-17
Linkage with CORBA server application that uses TypeCode type, 6-16
Listing of reliable message IDs, 7-21
Log, 10-9
Log output, 10-9
Logs of the client function, 10-12
Logs of the CORBA/SOAP client gateway, 10-10
Management of stored messages, 7-20
Mapping of XML and CORBA Data Types, 8-31
Mapping of XML and EJB Data Types, 8-39
Mapping of XML and Java Data Types, 8-2
Message type ID, 7-2
Messaging Method Sample Programs, 11-29
Multiple reference, 8-28
Notes on Use, 11-34
Obtaining CORBA exception from Fault details item, 6-6
Obtaining reliable messages, 7-21
Obtaining the exception class from Fault details item, 6-27
Operating the CORBA/SOAP Gateway, 10-3
operation tag, A-9
option
setting, A-8
Other precautions, 6-18
Outputting logs to a file, 12-7
parameter tag
as child element of operation tag, A-10
as child element of the handler tag, A-13
Precautions on Linkage with CORBA Application, 6-5
Precautions on Linkage with EJB Application, 6-19
Preparation of key information, 7-7, 7-13, 7-14
Preparation of key pair and public-key, 7-6
Preparation of receiver application, 7-6, 7-14
Preparation of sender application, 7-8, 7-13
Primitive types, 8-39
Provision of parameters to be sent to method of EJB server application, 6-19
PULL Model where Client System Receives Messages, 7-12
pullld, A-8
PUSH Model where Server System Receives Messages, 7-5
pushld, A-8
Registering the User PIN, 8-2
role, A-8
RPC client application that uses user-defined types (DII method), 5-22
RPC Client Application Using the DII Method, 5-17
RPC Client Application Using the Stub Method, 5-7
RPC Method Sample Programs, 11-6
RPC Server Application, 5-4
RPC-based client application that links with Factory interface, 6-12
RPC-based client application that links with STATEFUL SessionBean, 6-22
Sample program Calc, 11-6
Sample program configuration, 11-2
Sample program ConvertName, 11-7
Sample program ConvertStrBean, 11-24
Sample program CountRequest, 11-10
Sample program Reliable, 11-29
Sample program ReturnArray, 11-11
Sample program ReturnMonth, 11-13
Sample program SimpleInout, 11-15
Sample program StringLength, 11-16
Sample program that extends PlusJ2EE component models to Web services, 11-20
Sample program Young3Persons, 11-18, 11-20
scope, A-7
Sender ID, Receiver ID, 7-3
Setting of client system, 7-19
Setting of Reliable messaging function, 7-10, 7-13
Setting of server system, 7-19
Setting of the Reliable messaging function, 7-7, 7-15
Setting the HTTP version, 12-5
Setting the log output destination, 12-6
Setting the Logs of the Client Function, 12-6
Setting the return value of, 12-9
SOAP Service operation commands
  soapSetSSL, 6-2, 10-2, 12-2
SOAP signature verification ID, 7-4
soapSetSSL, 6-2, 10-2, 12-2
Starting Reliable messaging function, 7-16
static
  true or false, A-8
Structure and exception types, 8-34
Structure of application using a combination of PUSH and PULL models, 7-17
Structure of application using PULL model, 7-16
Structure of application using PUSH model, 7-11
Structure type, 8-14
Stub method and DII method, 5-2
TypeCode type, 8-38
typeMapping tag, A-11
URL of Receiver server/sender server, 7-4
Using session control, 5-14, 5-27
Web, A-1
Web service identifier
  client system deployment, 9-21
Web service identifier and URL, 9-20

Web Service Information
  Managing, 9-19
XML tag hierarchy, A-3
XML tags
  content, A-5